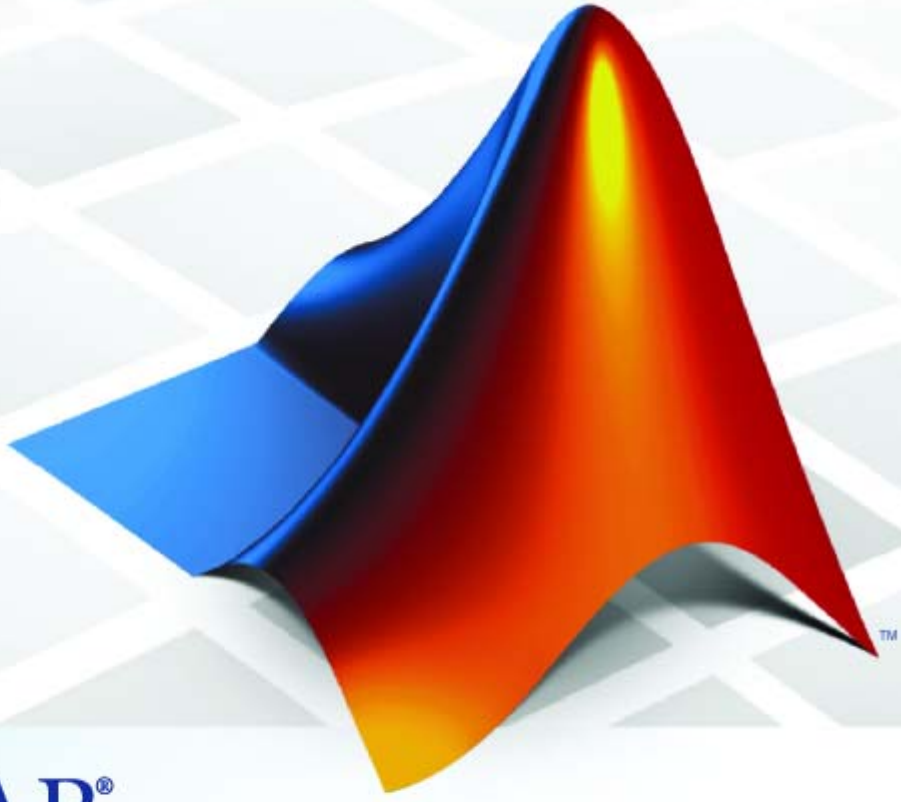
Simulink® 7
Reference



MATLAB®
&SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science

How to Contact The MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Simulink® Reference

Trademarks

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

Patents

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)
March 2006	Online only	Revised for Simulink 6.4 (Release 2006a)
September 2006	Online only	Revised for Simulink 6.5 (Release 2006b)
March 2007	Online only	Revised for Simulink 6.6 (Release 2007a)
September 2007	Online only	Revised for Simulink 7.0 (Release 2007b)
March 2008	Online only	Revised for Simulink 7.1 (Release 2008a)
October 2008	Online only	Revised for Simulink 7.2 (Release 2008b)
March 2009	Online only	Revised for Simulink 7.3 (Release 2009a)
September 2009	Online only	Revised for Simulink 7.4 (Release 2009b)

Contents

1 Block Reference

2 Blocks — Alphabetical List

3 Function Reference

9 Model File Format

10 Model Advisor Checks

1

Block Reference

Commonly Used

Bus Creator	Create signal bus
Bus Selector	Select signals from incoming bus
Constant	Generate constant value
Data Type Conversion	Convert input signal to specified data type
Demux	Extract and output elements of vector signal
Discrete-Time Integrator	Perform discrete-time integration or accumulation of signal
Gain	Multiply input by constant
Ground	Ground unconnected input port
Inport	Create input port for subsystem or external input
Integrator	Integrate signal
Logical Operator	Perform specified logical operation on input
Mux	Combine several input signals into vector
Outport	Create output port for subsystem or external output
Product	Multiply and divide scalars and nonscalars or multiply and invert matrices
Relational Operator	Perform specified relational operation on inputs
Saturation	Limit range of signal
Scope and Floating Scope	Display signals generated during simulation

Subsystem, Atomic Subsystem, Nonvirtual Subsystem, CodeReuse Subsystem	Represent system within another system
Sum, Add, Subtract, Sum of Elements	Add or subtract inputs
Switch	Switch output between first input and third input based on value of second input
Terminator	Terminate unconnected output port
Unit Delay	Delay signal one sample period

Continuous

Derivative	Output time derivative of input
Integrator	Integrate signal
PID Controller	Simulate continuous- or discrete-time PID controllers
PID Controller (2 DOF)	Simulate continuous- or discrete-time two-degree-of-freedom PID controllers
State-Space	Implement linear state-space system
Transfer Fcn	Model linear system by transfer function
Transport Delay	Delay input by given amount of time
Variable Time Delay, Variable Transport Delay	Delay input by variable amount of time
Zero-Pole	Model system by zero-pole-gain transfer function

Discontinuities

Backlash	Model behavior of system with play
Coulomb and Viscous Friction	Model discontinuity at zero, with linear gain elsewhere
Dead Zone	Provide region of zero output
Dead Zone Dynamic	Set inputs within bounds to zero
Hit Crossing	Detect crossing point
Quantizer	Discretize input at specified interval
Rate Limiter	Limit rate of change of signal
Rate Limiter Dynamic	Limit rising and falling rates of signal
Relay	Switch output between two constants
Saturation	Limit range of signal
Saturation Dynamic	Bound range of input
Wrap To Zero	Set output to zero if input is above threshold

Discrete

Difference	Calculate change in signal over one time step
Discrete Derivative	Compute discrete time derivative
Discrete Filter	Model Infinite Impulse Response (IIR) direct form II filters
Discrete FIR Filter	Model FIR filters
Discrete State-Space	Implement discrete state-space system
Discrete Transfer Fcn	Implement discrete transfer function

Discrete Zero-Pole	Model system defined by zeros and poles of discrete transfer function
Discrete-Time Integrator	Perform discrete-time integration or accumulation of signal
First-Order Hold	Implement first-order sample-and-hold
Integer Delay	Delay signal N sample periods
Memory	Output input from previous time step
PID Controller	Simulate continuous- or discrete-time PID controllers
PID Controller (2 DOF)	Simulate continuous- or discrete-time two-degree-of-freedom PID controllers
Tapped Delay	Delay scalar signal multiple sample periods and output all delayed versions
Transfer Fcn First Order	Implement discrete-time first order transfer function
Transfer Fcn Lead or Lag	Implement discrete-time lead or lag compensator
Transfer Fcn Real Zero	Implement discrete-time transfer function that has real zero and no pole
Unit Delay	Delay signal one sample period
Weighted Moving Average (Obsolete)	Implement weighted moving average (obsolete)
Zero-Order Hold	Implement zero-order hold of one sample period

Logic and Bit Operations

Bit Clear	Set specified bit of stored integer to zero
Bit Set	Set specified bit of stored integer to one
Bitwise Operator	Perform specified bitwise operation on inputs
Combinatorial Logic	Implement truth table
Compare To Constant	Determine how signal compares to specified constant
Compare To Zero	Determine how signal compares to zero
Detect Change	Detect change in signal's value
Detect Decrease	Detect decrease in signal's value
Detect Fall Negative	Detect falling edge when signal's value decreases to strictly negative value, and its previous value was nonnegative
Detect Fall Nonpositive	Detect falling edge when signal's value decreases to nonpositive value, and its previous value was strictly positive
Detect Increase	Detect increase in signal's value
Detect Rise Nonnegative	Detect rising edge when signal's value increases to nonnegative value, and its previous value was strictly negative
Detect Rise Positive	Detect rising edge when signal's value increases to strictly positive value, and its previous value was nonpositive

Extract Bits	Output selection of contiguous bits from input signal
Interval Test	Determine if signal is in specified interval
Interval Test Dynamic	Determine if signal is in specified interval
Logical Operator	Perform specified logical operation on input
Relational Operator	Perform specified relational operation on inputs
Shift Arithmetic	Shift bits or binary point of signal

Lookup Tables

Direct Lookup Table (n-D)	Index into N-dimensional table to retrieve element, column, or 2-D matrix
Interpolation Using Prelookup	Use precalculated index and fraction values to accelerate approximation of N-dimensional function
Lookup Table	Approximate one-dimensional function
Lookup Table (2-D)	Approximate two-dimensional function
Lookup Table (n-D)	Approximate N-dimensional function
Lookup Table Dynamic	Approximate one-dimensional function using dynamic table

Prelookup	Compute index and fraction for Interpolation Using Prelookup block
Sine, Cosine	Implement sine and/or cosine wave in fixed point using lookup table approach that exploits quarter wave symmetry

Math Operations

Abs	Output absolute value of input
Algebraic Constraint	Constrain input signal to zero
Assignment	Assign values to specified elements of signal
Bias	Add bias to input
Complex to Magnitude-Angle	Compute magnitude and/or phase angle of complex signal
Complex to Real-Imag	Output real and imaginary parts of complex input signal
Divide	Divide one input by another
Dot Product	Generate dot product of two vectors
Gain	Multiply input by constant
Magnitude-Angle to Complex	Convert magnitude and/or a phase angle signal to complex signal
Math Function	Perform mathematical function
Matrix Concatenate, Vector Concatenate	Concatenate input signals of same data type to create contiguous output signal
MinMax	Output minimum or maximum input value

MinMax Running Resettable	Determine minimum or maximum of signal over time
Permute Dimensions	Rearrange dimensions of multidimensional array dimensions
Polynomial	Perform evaluation of polynomial coefficients on input values
Product	Multiply and divide scalars and nonscalars or multiply and invert matrices
Product of Elements	Copy or invert one scalar input, or collapse one nonscalar input
Real-Imag to Complex	Convert real and/or imaginary inputs to complex signal
Reshape	Change dimensionality of signal
Rounding Function	Apply rounding function to signal
Sign	Indicate sign of input
Sine Wave Function	Generate sine wave, using external signal as time source
Slider Gain	Vary scalar gain using slider
Squeeze	Remove singleton dimensions from multidimensional signal
Sum, Add, Subtract, Sum of Elements	Add or subtract inputs
Trigonometric Function	Perform trigonometric function
Unary Minus	Negate input
Weighted Sample Time Math	Support calculations involving sample time

Model Verification

Assertion	Check whether signal is zero
Check Discrete Gradient	Check that absolute value of difference between successive samples of discrete signal is less than upper bound
Check Dynamic Gap	Check that gap of possibly varying width occurs in range of signal's amplitudes
Check Dynamic Lower Bound	Check that one signal is always less than another signal
Check Dynamic Range	Check that signal falls inside range of amplitudes that varies from time step to time step
Check Dynamic Upper Bound	Check that one signal is always greater than another signal
Check Input Resolution	Check that input signal has specified resolution
Check Static Gap	Check that gap exists in signal's range of amplitudes
Check Static Lower Bound	Check that signal is greater than (or optionally equal to) static lower bound
Check Static Range	Check that signal falls inside fixed range of amplitudes
Check Static Upper Bound	Check that signal is less than (or optionally equal to) static upper bound

Model-Wide Utilities

Block Support Table	View data type support for Simulink® blocks
DocBlock	Create text that documents model and save text with model
Model Info	Display revision control information in model
Time-Based Linearization	Generate linear models in base workspace at specific times
Trigger-Based Linearization	Generate linear models in base workspace when triggered

Ports & Subsystems

Action Port	Implement Action subsystems used by `if` and `switch` control flow statements in Simulink software
Configurable Subsystem	Represent any block selected from user-specified library of blocks
Enable	Add enabling port to subsystem
Enabled and Triggered Subsystem	Represent subsystem whose execution is enabled and triggered by external input
Enabled Subsystem	Represent subsystem whose execution is enabled by external input
For Iterator	Repeatedly execute contents of subsystem at current time step until iteration variable exceeds specified iteration limit

For Iterator Subsystem	Represent subsystem that executes repeatedly during simulation time step
Function-Call Generator	Execute function-call subsystem specified number of times at specified rate
Function-Call Subsystem	Represent subsystem that can be invoked as function by another block
If	Model `if-else` control flow
If Action Subsystem	Represent subsystem whose execution is triggered by If block
Inport	Create input port for subsystem or external input
Model	Include model as block in another model
Outport	Create output port for subsystem or external output
Subsystem, Atomic Subsystem, Nonvirtual Subsystem, CodeReuse Subsystem	Represent system within another system
Switch Case	Implement C-like `switch` control flow statement
Switch Case Action Subsystem	Represent subsystem whose execution is triggered by Switch Case block
Trigger	Add trigger port to subsystem or function-call model
Triggered Subsystem	Represent subsystem whose execution is triggered by external input
While Iterator Subsystem	Represent subsystem that executes repeatedly while condition is satisfied during simulation time step

Signal Attributes

Bus to Vector	Convert virtual bus to vector
Data Type Conversion	Convert input signal to specified data type
Data Type Conversion Inherited	Convert from one data type to another using inherited data type and scaling
Data Type Duplicate	Force all inputs to same data type
Data Type Propagation	Set data type and scaling of propagated signal based on information from reference signals
Data Type Scaling Strip	Remove scaling and map to built in integer
IC	Set initial value of signal
Probe	Output signal's attributes, including width, dimensionality, sample time, and/or complex signal flag
Rate Transition	Handle transfer of data between blocks operating at different rates
Signal Conversion	Convert signal to new type without altering signal values
Signal Specification	Specify desired dimensions, sample time, data type, numeric type, and other attributes of signal
Weighted Sample Time	Support calculations involving sample time
Width	Output width of input vector

Signal Routing

Bus Assignment	Replace specified bus elements
Bus Creator	Create signal bus
Bus Selector	Select signals from incoming bus
Data Store Memory	Define data store
Data Store Read	Read data from data store
Data Store Write	Write data to data store
Demux	Extract and output elements of vector signal
Environment Controller	Create branches of block diagram that apply only to simulation or only to code generation
From	Accept input from Goto block
Goto	Pass block input to From blocks
Goto Tag Visibility	Define scope of Goto block tag
Index Vector	Switch output between different inputs based on value of first input
Manual Switch	Switch between two inputs
Merge	Combine multiple signals into single signal
Multiport Switch	Choose between multiple block inputs
Mux	Combine several input signals into vector
Selector	Select input elements from vector, matrix, or multidimensional signal
Switch	Switch output between first input and third input based on value of second input

Sinks

Display	Show value of input
Outport	Create output port for subsystem or external output
Scope and Floating Scope	Display signals generated during simulation
Stop Simulation	Stop simulation when input is nonzero
Terminator	Terminate unconnected output port
To File	Write data to file
To Workspace	Write data to MATLAB® workspace
XY Graph	Display X-Y plot of signals using MATLAB figure window

Sources

Band-Limited White Noise	Introduce white noise into continuous system
Chirp Signal	Generate sine wave with increasing frequency
Clock	Display and provide simulation time
Constant	Generate constant value
Counter Free-Running	Count up and overflow back to zero after maximum value possible is reached for specified number of bits
Counter Limited	Count up and wrap back to zero after outputting specified upper limit
Digital Clock	Output simulation time at specified sampling interval

Enumerated Constant	Generate enumerated constant value
From File	Read data from MAT-file
From Workspace	Read data from workspace
Ground	Ground unconnected input port
Inport	Create input port for subsystem or external input
Pulse Generator	Generate square wave pulses at regular intervals
Ramp	Generate constantly increasing or decreasing signal
Random Number	Generate normally distributed random numbers
Repeating Sequence	Generate arbitrarily shaped periodic signal
Repeating Sequence Interpolated	Output discrete-time sequence and repeat, interpolating between data points
Repeating Sequence Stair	Output and repeat discrete time sequence
Signal Builder	Create and generate interchangeable groups of signals whose waveforms are piecewise linear
Signal Generator	Generate various waveforms
Sine Wave	Generate sine wave, using simulation time as time source
Step	Generate step function
Uniform Random Number	Generate uniformly distributed random numbers

User-Defined Functions

Embedded MATLAB Function	Include MATLAB code in models that generate embeddable C code
Fcn	Apply specified expression to input
Level-2 M-File S-Function	Use Level-2 M-file S-function in model
MATLAB Fcn	Apply MATLAB function or expression to input
S-Function	Include S-function in model
S-Function Builder	Create S-function from C code that you provide

Additional Math & Discrete

Additional Discrete (p. 1-17)	Provide additional discrete math support
Additional Math: Increment — Decrement (p. 1-18)	Increment or decrement value of signal by one

Additional Discrete

Fixed-Point State-Space	Implement discrete-time state space
Transfer Fcn Direct Form II	Implement Direct Form II realization of transfer function
Transfer Fcn Direct Form II Time Varying	Implement time varying Direct Form II realization of transfer function
Unit Delay Enabled	Delay signal one sample period, if external enable signal is on

Unit Delay Enabled External IC	Delay signal one sample period, if external enable signal is on, with external initial condition
Unit Delay Enabled Resettable	Delay signal one sample period, if external enable signal is on, with external Boolean reset
Unit Delay Enabled Resettable External IC	Delay signal one sample period, if external enable signal is on, with external Boolean reset and initial condition
Unit Delay External IC	Delay signal one sample period, with external initial condition
Unit Delay Resettable	Delay signal one sample period, with external Boolean reset
Unit Delay Resettable External IC	Delay signal one sample period, with external Boolean reset and initial condition
Unit Delay With Preview Enabled	Output signal and signal delayed by one sample period, if external enable signal is on
Unit Delay With Preview Enabled Resettable	Output signal and signal delayed by one sample period, if external enable signal is on, with external reset
Unit Delay With Preview Enabled Resettable External RV	Output signal and signal delayed by one sample period, if external enable signal is on, with external RV reset
Unit Delay With Preview Resettable	Output signal and signal delayed by one sample period, with external reset
Unit Delay With Preview Resettable External RV	Output signal and signal delayed by one sample period, with external RV reset

Additional Math: Increment — Decrement

2

Blocks — Alphabetical List

Abs

Purpose

Output absolute value of 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 selected, the absolute value of the data type saturates to the most positive representable value. If not selected, 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.

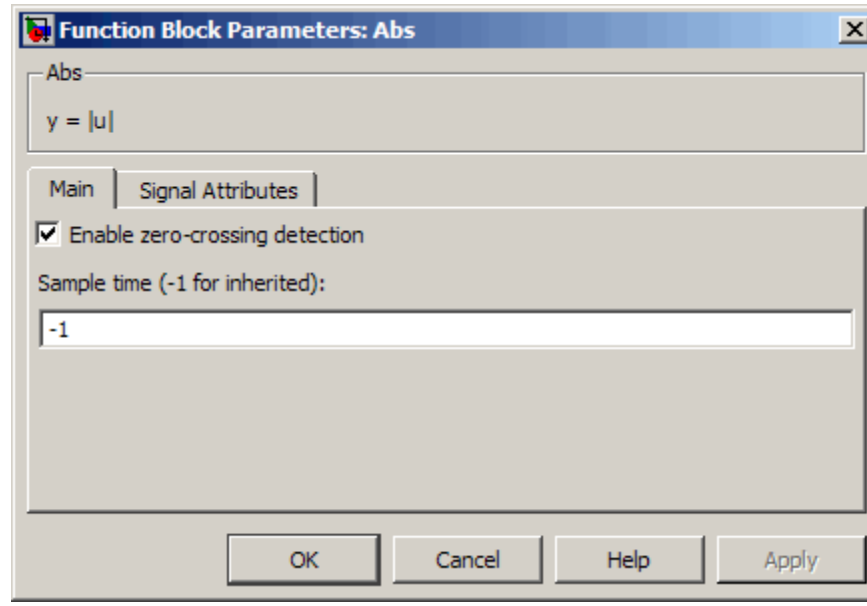
Data Type Support

The Abs block accepts real signals of any numeric data type supported by Simulink software, except Boolean. The Abs block supports real fixed-point data types. The block also accepts complex floating-point inputs.

For a discussion on the data types supported by Simulink software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

Parameters and Dialog Box

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



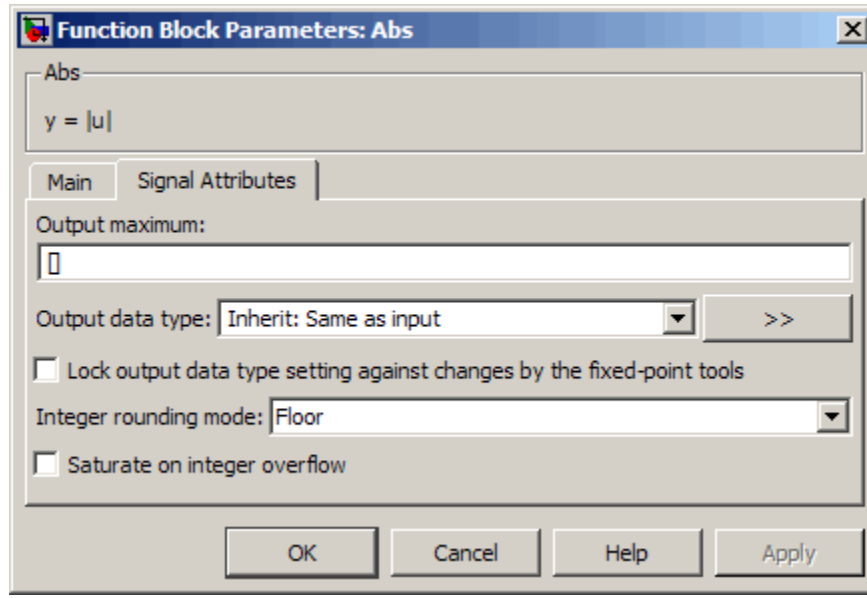
Enable zero-crossing detection

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

Sample time (-1 for inherited)

Enter the time interval between sample time hits or specify another appropriate sample time such as continuous. By default, the block inherits its sample time based upon its context within the model. See "Working with Sample Times" in the Simulink documentation.

The **Signal Attributes** pane of the Abs block dialog box appears as follows:



Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object

- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink® Fixed Point™ documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate on integer overflow

Select to have overflows saturate. If 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 127.
- For 16-bit integers, -32768 maps to 32767.
- For 32-bit integers, -2147483648 maps to 2147483647.

Otherwise, 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.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Characteristics

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

Purpose

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

Library

Ports & Subsystems

Description

Action Port blocks implement Action subsystems used in `if` and `switch` control flow statements. The Action Port block is available in the If Action Subsystem and the Switch Case Action Subsystem.

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 block 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 to the Action port on an Action subsystem.

   - if, else, or elseif ports for the If block
   - case or default ports for the Switch Case block

   When you make a connection, the icon changes for the subsystem and the Action Port block it contains to the name of the output port for the If or Switch Case block (for example, `if{ }`, `else{ }`, `elseif{ }`, `case{ }`, or `default{ }`).

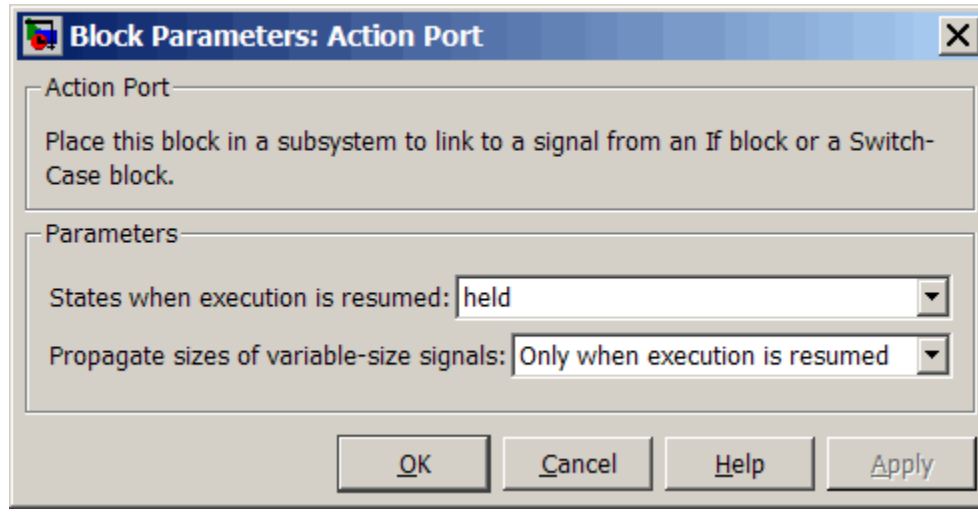
3 Open the new subsystem and add the diagram that you want to execute in response to the condition this subsystem covers.

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 Simulink resumes execution" on page 2-10
- "Propagate sizes of variable-size signals" on page 2-12

Action Port

States when Simulink resumes execution

Specify how to handle internal states when a subsystem with an Action Port block reenables.

Settings

Default: `held`

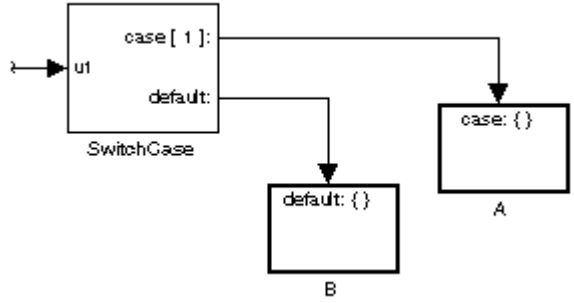
`held`

When the subsystem reenables, retains the states of the Action subsystem with their previous values. Retains the previous values of states between calls even if calling other member Action subsystems of an `if-else` or `switch` control flow statement.

`reset`

Reinitializes the states of the Action subsystem to initial values when the subsystem reenables.

Reenablement of a subsystem occurs when 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, call Action subsystem A. This result implies that the default condition is false. When later calling B for the default condition, its states are reset. In the same way, Action

subsystem A states are reset when calling A right after calling Action subsystem B.

Repeated calls to the Action subsystem of a case does not reset its states. If calling A again right after a previous call to A, this action does not reset the states of A. This behavior is because the condition of case[1] was not previously false. The same applies to B.

Command-Line Information

Parameter: `InitializeStates`
Type: string
Value: `'held' | 'reset' |`
Default: `'held'`

Propagate sizes of variable-size signals

Specify when to propagate a variable-size signal.

Settings

Default: `Only when execution is resumed`

`Only when execution is resumed`
: Propagates variable-size signals only when reenabling the subsystem containing the Action Port block.

`During execution`
: Propagates variable-size signals at each time step.

Command-Line Information

Parameter: `PropagateVarSize`
Type: string
Value: `'Only when execution is resumed' | 'During execution'`
Default: `'Only when execution is resumed'`

Characteristics

Sample Time	Inherited from driving If or Switch Case block

See Also

If, If Action Subsystem, Switch Case, and Switch Case Action Subsystem blocks for examples using Action Port blocks

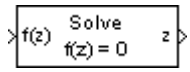
Purpose

Constrain 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 that produces a zero at the input. The output must affect the input through a direct feedback path, that is, the feedback path contains only blocks with direct feedthrough. For example, you can specify algebraic equations for index 1 differential-algebraic systems (DAEs).

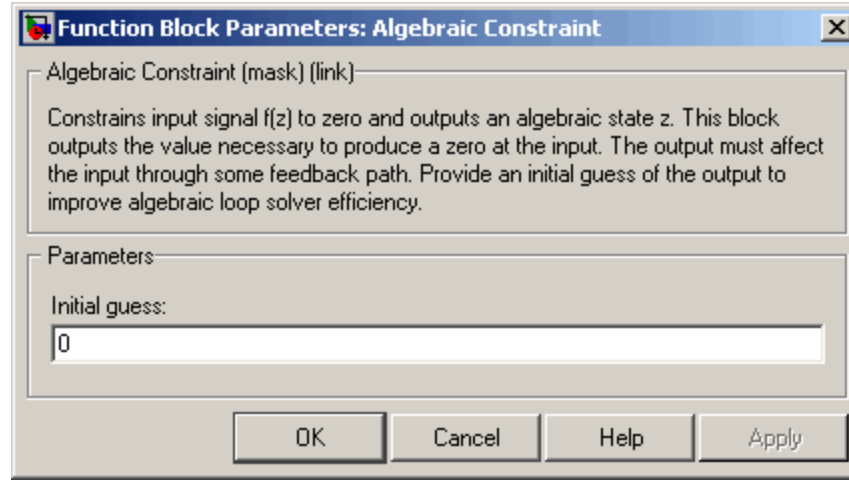
Algorithm

The Algebraic Constraint block uses a dogleg trust-region algorithm to solve algebraic loops [1], [2].

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`.

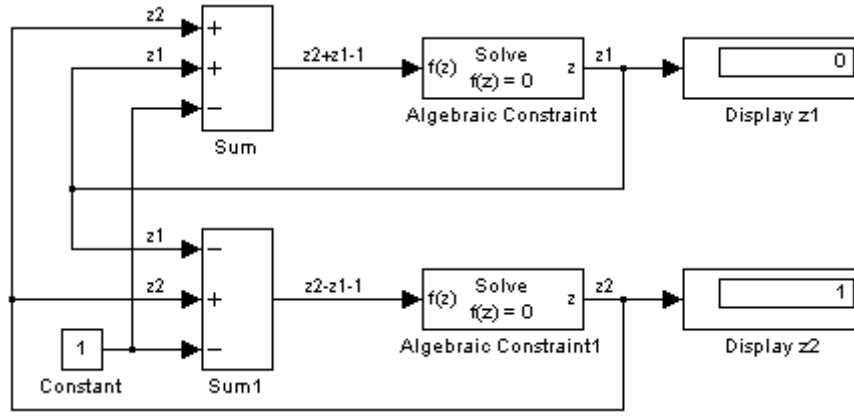
Example

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.



Characteristics

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

References

[1] Garbow, B. S., K. E. Hillstrom, and J. J. Moré. *User Guide for MINPACK-1.* Argonne, IL: Argonne National Laboratory, 1980.

[2] Rabinowitz, P. H. *Numerical Methods for Nonlinear Algebraic Equations*. New York, NY: Gordon and Breach, 1970.

Assertion

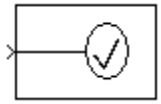
Purpose

Check whether signal is zero

Library

Model Verification

Description



The Assertion block checks whether any of the elements of the input signal is zero. If all elements are nonzero, the block does nothing. If any element is zero, the block halts the simulation, by default, and displays an error message. Use the block parameter dialog box to:

- Specify that the block should display an error message when the assertion fails but allow the simulation to continue.
- Specify a MATLAB expression to evaluate when the assertion fails.
- Enable or disable the assertion.

You can also use the **Model Verification block enabling** setting on the **Data Validity** 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.

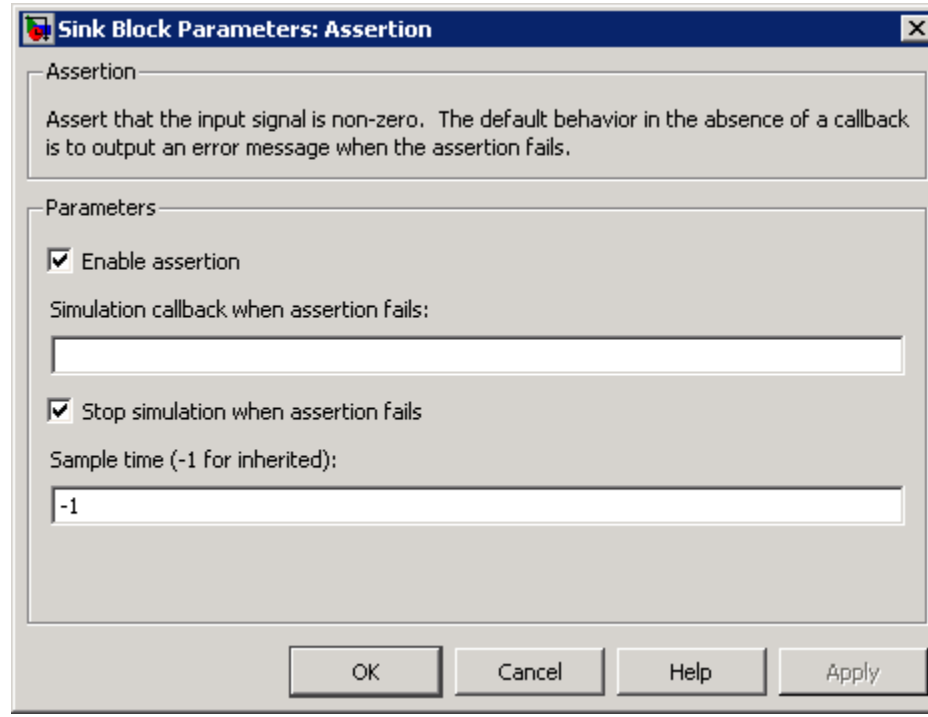
Note For information about how Real-Time Workshop® generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Enable assertion

Clearing this check box 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 under **Debugging** on the **Data Validity** diagnostics pane of the

Configuration Parameters dialog box lets you enable or disable all Assertion blocks in a model regardless of the setting of this option.

Simulation callback when assertion fails
A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails
Selecting this check box causes the Assertion block to halt the simulation when the block 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)
Enter the time interval between sample time hits or specify another appropriate sample time such as continuous. By default, the block inherits its sample time based upon its context within the model. See "Working with Sample Times" in the Simulink documentation.

Characteristics

Direct Feedthrough	No
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	Yes
Multidimensionalized	Yes
Zero-Crossing Detection	No

Purpose

Assign values to specified elements of 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 U, specifies values to be assigned to Y. The block replaces the specified elements of Y with elements from the data signal.

Based on the value you enter for the **Number of output dimensions** parameter, a table of index options is displayed. Each row of the table corresponds to one of the output dimensions in **Number of output dimensions**. For each dimension, you can define the elements of the signal to work with. Specify a vector signal as a 1-D signal and a matrix signal as a 2-D signal. When you configure the Assignment block for multidimensional signal operations, the block icon changes.

For example, assume a 5-D signal with a one-based index mode. The table in the Assignment block dialog changes to include one row for each dimension. If you define each dimension with the following entries:

- **1**

  Index Option, select `Assign all`

- **2**

  Index Option, select `Index vector (dialog)`

  Index, enter `[1 3 5]`

- **3**

  Index Option, select `Starting index (dialog)`

  Index, enter `4`

- **4**

  Index Option, select `Starting index (port)`

- **5**

  Index Option, select `Index vector (port)`

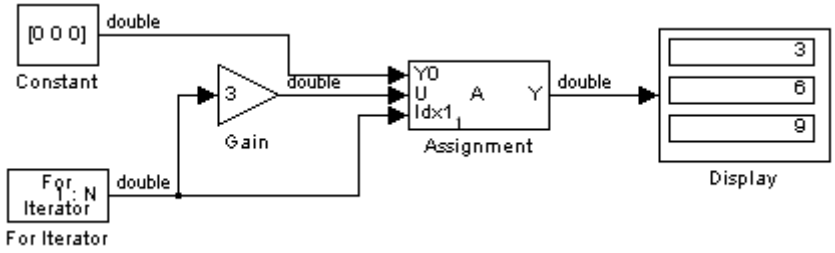
The assigned values will be `Y(1:end,[1 3 5],4:3+size(U,3),Idx4:Idx4+size(U,4)-1,Idx5)=U`, where `Idx4` and `Idx5` are the input ports for dimensions 4 and 5.

The Assignment block's data port is labeled U. The rest of this section refers to the data port as U to simplify the explanation of the block's usage.

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

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, matrix, or multidimensional 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.



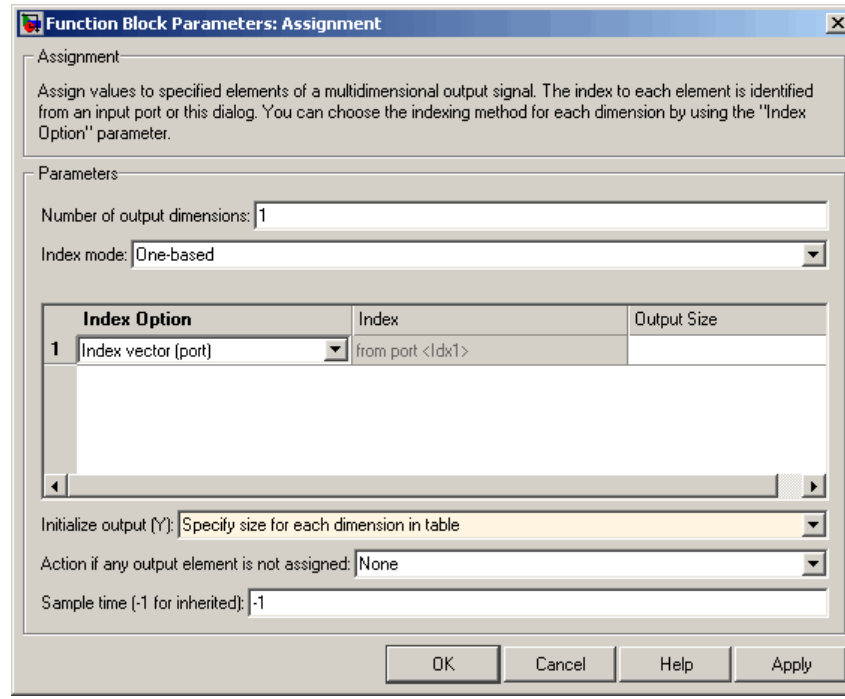
Iterated assignment uses an iterator (For or While) block to generate the indices required by the Assignment block. On the first iteration of an iterated assignment, the Assignment block copies the first input (Y0) to the output (Y) and assigns the second input (U) to the output $Y(E_1)$. On successive iterations, the Assignment block simply assigns the current value of U to $Y(E_i)$, i.e., without first copying Y0 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 software, including fixed-point and enumerated data types. The external indices port accepts any built-in data type, except Boolean data types.

For a discussion on the data types supported by Simulink software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

Parameters and Dialog Box



Number of output dimensions

Enter the number of dimensions of the output signal.

Index mode

Specifies the indexing mode: `One-based` or `Zero-based`. If `One-based` is selected, an index of 1 specifies the first element of

the input vector, 2, the second element, and so on. If `Zero-based` is selected, an index of 0 specifies the first element of the input vector, 1, the second element, and so on.

Index Option

Define, by dimension, how the elements of the signal are to be indexed. From the list, choose:

- `Assign all`

  This is the default. All elements are assigned.

- `Index vector (dialog)`

  Enables the **Index** column. Enter the indices of elements.

- `Index vector (port)`

  Disables the **Index** column. The index port defines the indices of elements.

- `Starting index (dialog)`

  Enables the **Index** column. Enter the starting index of the range of elements to be assigned values.

- `Starting index (port)`

  Disables the **Index** column. The index port defines the starting index of the range of elements to be assigned values.

If you choose `Index vector (port)` or `Starting index (port)` for any dimension in the table, you can specify the value for the **Initialize output (Y)** parameter to be one of the following:

- `Initialize using input port <Y0>`
- `Specify size for each dimension in table`

Otherwise, Y0 always initializes output port Y.

The **Index** and **Output Size** columns are displayed as relevant.

Index

If the **Index Option** is `Index vector (dialog)`, enter the index of each element you are interested in.

If the **Index Option** is `Starting index (dialog)`, enter the starting index of the range of elements to be selected. The number of elements from the starting point is determined by the size of this dimension at U.

Output Size

Enter the width of the block output signal. If you select `Specify size for each dimension in table` for the **Initialize output (Y)** parameter, this column is enabled.

Initialize output (Y)

Specify how to initialize the output signal. The **Initialize output** parameter appears only if you select `Index vector (port)` or `Starting index (port)` for the **Index Option** parameter.

- `Initialize using input port <Y0>`

  The signal at the input port Y0 initializes the output.

- `Specify size for each dimension in table`

  The block requires you to specify the width of the block's output signal in the **Output Size** parameter. If the output has unassigned elements, the value of those elements is undefined.

Action if any output element is not assigned

Specifies whether to produce a warning or error message if you have not assigned all output elements. Options include:

- `Error`
- `Warning`
- `None`

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample

Time" in the "How Simulink Works" chapter of the Simulink documentation.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified by Sample time parameter
Scalar Expansion	Yes
Dimensionalized	Yes
Multidimensionalized	Yes
Zero Crossing	No

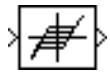
Purpose

Model behavior of system with play

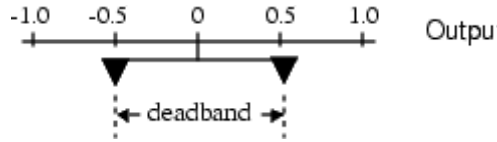
Library

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:

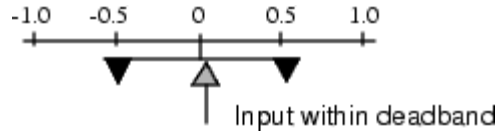
- Disengaged - In this mode, the input does not drive the output and the output remains constant.
- Engaged in a positive direction - In this mode, the input is increasing (has a positive slope) and the output is equal to the input *minus* half the deadband width.
- Engaged in a negative direction -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

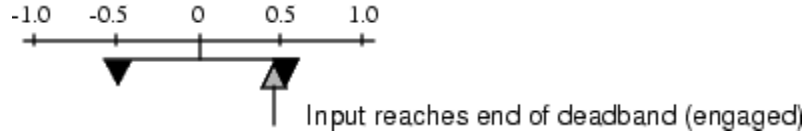
For example, the Backlash block can be used to model the meshing of two gears. The input and output are both shafts with a gear on one end, and the output shaft is driven by the input shaft. Extra space

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.

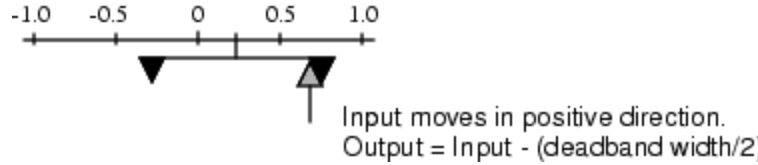
The following figures illustrate the block's operation when the initial input is within the deadband. The first figure shows the relationship between the input and the output while the system is in disengaged mode (and the default parameter values are not changed).



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.

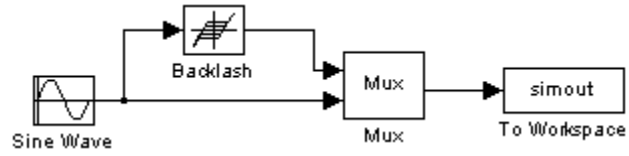


If the input reverses its direction, it disengages from the output. The output remains constant until the input either reaches the opposite end of the deadband or reverses its direction again and engages at the same end of the deadband. Now, as before, movement in the input causes equal movement in the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

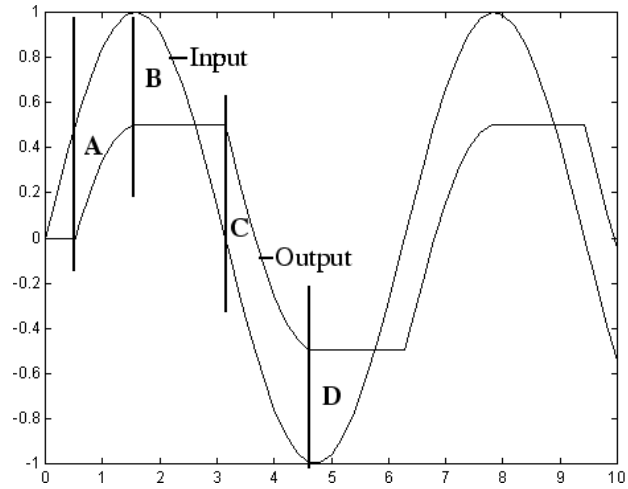
For example, if the deadband width is 2 and the initial output is 5, the output, *y*, at the start of the simulation is as follows:

- 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.

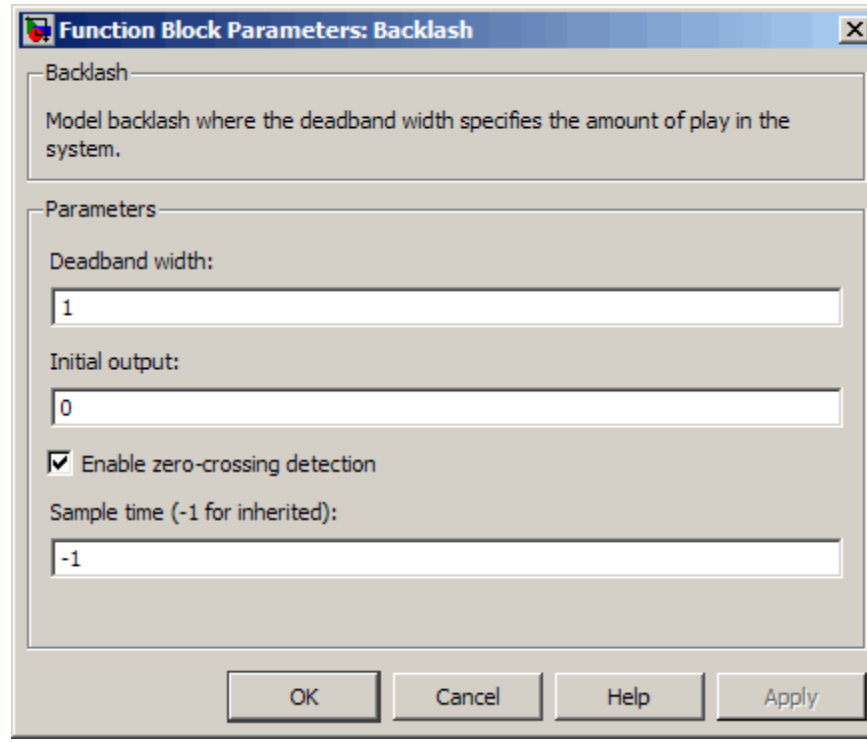
C Input engages in negative direction. Change in input causes equal change in output.

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 software does not allow the initial output of this block to be `inf` or `NaN`.

Enable zero-crossing detection

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

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in the "How Simulink Works" chapter of the Simulink documentation.

Characteristics

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

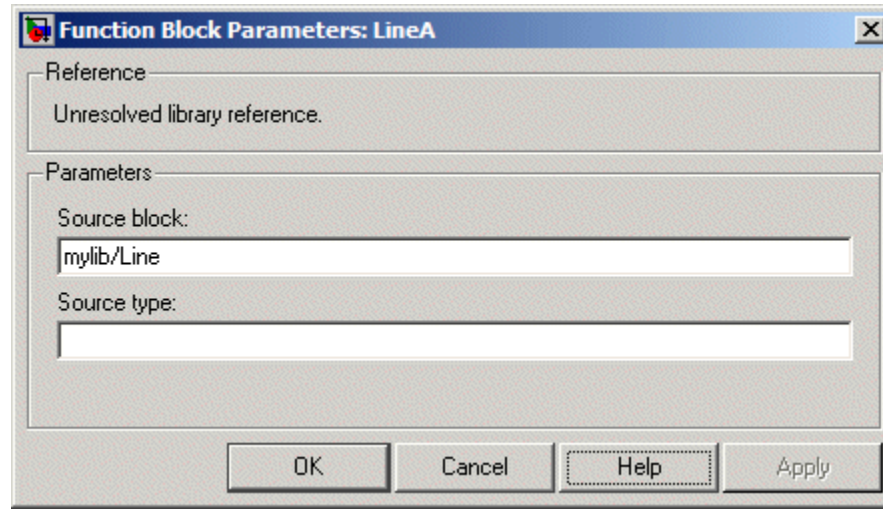
Purpose

Indicate unresolved reference to library block

Description

This block indicates an unresolved reference to a library block (see "Creating a Reference Block"). You can use this block's parameter dialog box to fix the reference to point to the actual location of the library block.

Parameters and Dialog Box



Source block

Path of the library block that this link represents. To fix a bad link, edit this field to reflect the actual path of the library block. Then select Apply or OK to apply the fix and close the dialog box.

Source type

Type of library block that this link represents.

Band-Limited White Noise

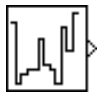
Purpose

Introduce white noise into 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 total energy 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 software, 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.

Algorithm

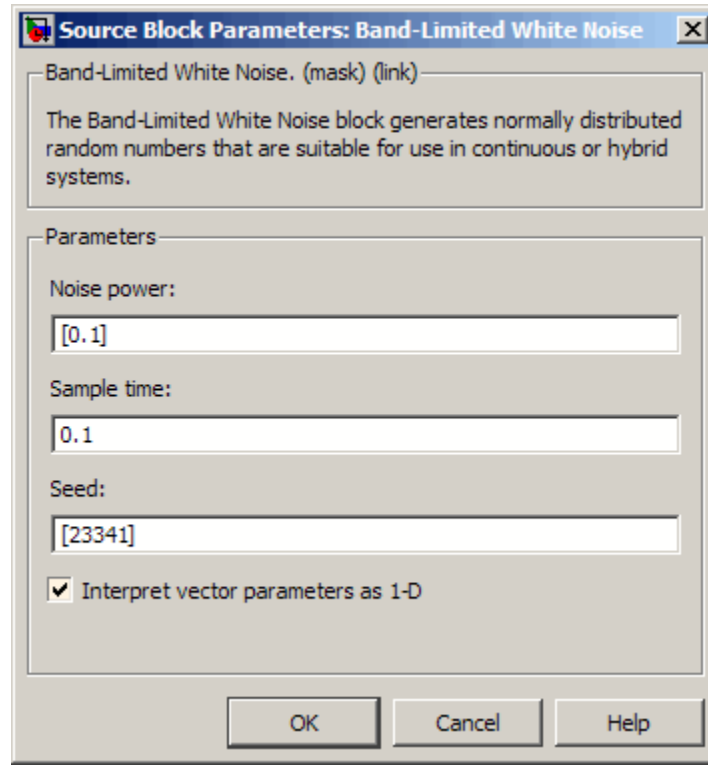
To produce the correct intensity of this noise, the covariance of the noise is scaled to reflect the implicit conversion from a continuous PSD to a discrete noise covariance. The appropriate scale factor is *1/tc*, where *tc* is the correlation time of the noise. This scaling ensures that the response 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. This block approximates the covariance of white noise as 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 During Simulation" in the "Working with Blocks" chapter of the Simulink documentation.

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 "How to Specify the Sample Time" in the "How Simulink Works" chapter of the Simulink documentation.

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`. See "Determining the Output Dimensions of Source Blocks" in the "Working with Signals" chapter of the Simulink documentation.

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

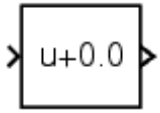
Add bias to input

Library

Math Operations

Description

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



$$Y = U + Bias$$

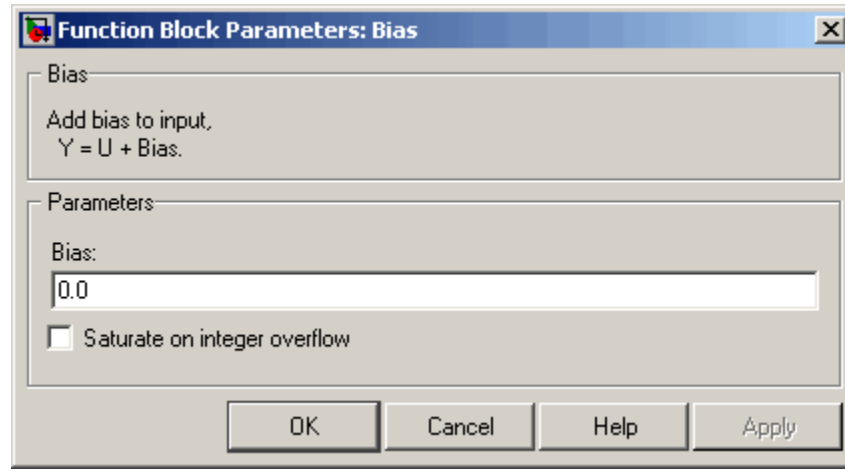
where U is the block input and Y is the output.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

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

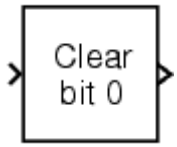
Purpose

Set specified bit of stored integer to zero

Library

Logic and Bit Operations

Description



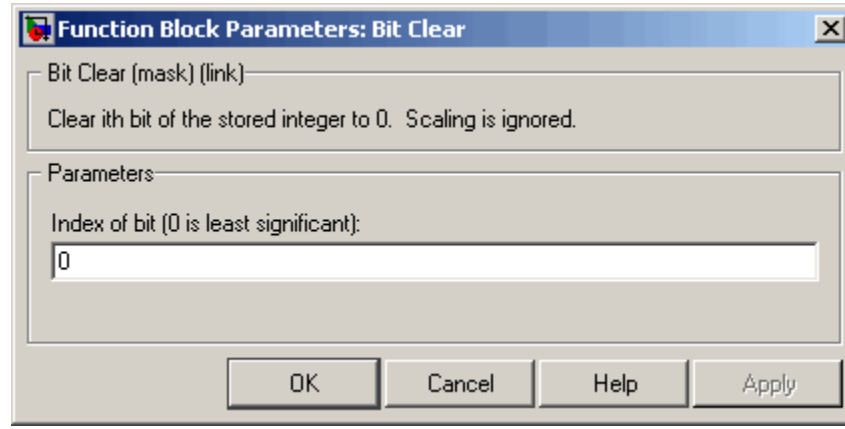
The Bit Clear block sets the specified bit, given by its index, of the stored integer to zero. Scaling is ignored.

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. The block does not support true floating-point data types or enumerated data types.

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

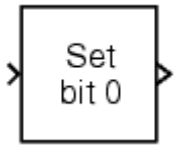
Set specified bit of stored integer to one

Library

Logic and Bit Operations

Description

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

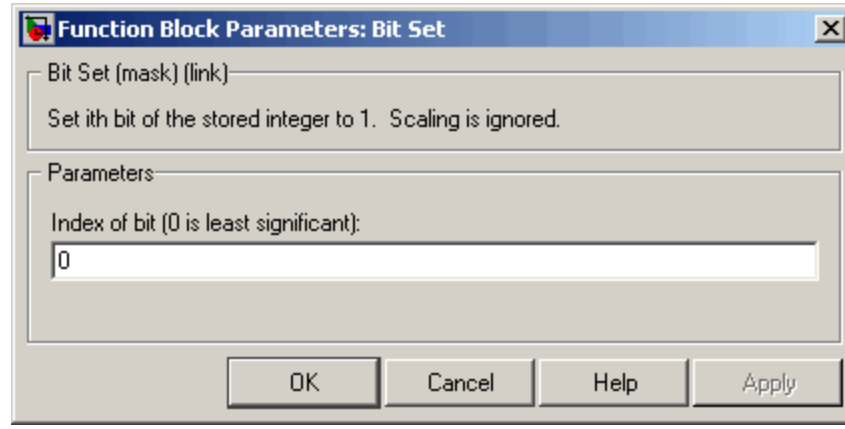


You can specify the bit to be set to one with the **Index of bit** parameter, where bit zero is the least significant bit.

Data Type Support

The Bit Set block supports Simulink integer, fixed-point, and Boolean data types. The block does not support true floating-point data types or enumerated data types.

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 specified bitwise operation on 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)

Restrictions on Block Operations

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

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.

Restrictions on Inputs and Outputs

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.

Tip 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. All inputs must have the same base data type.

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.

Tip 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.

Bit Set and Bit Clear Operations

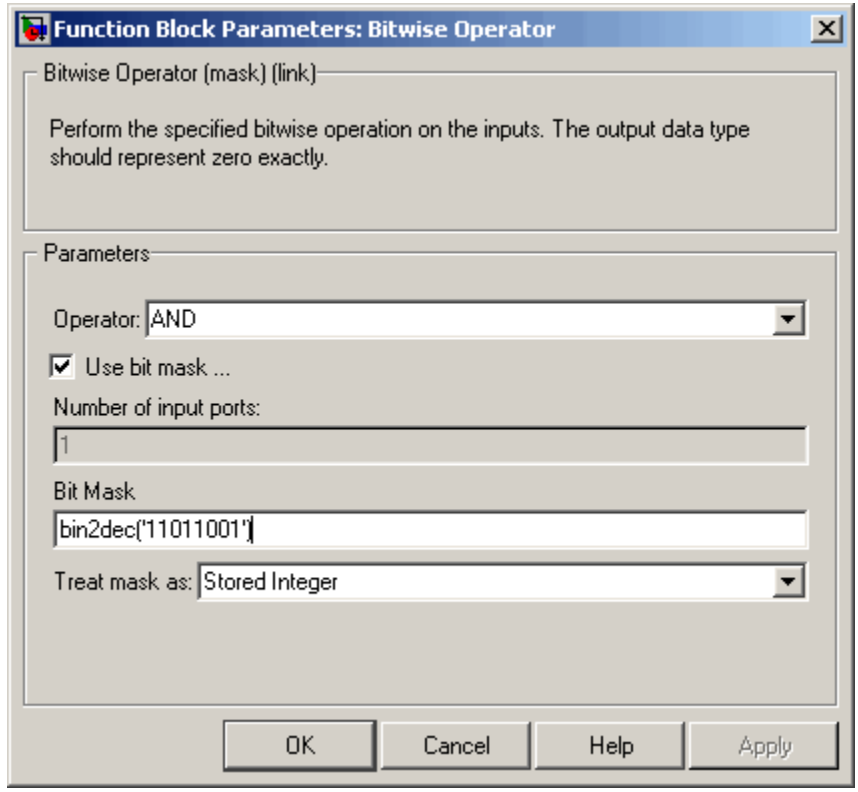
You can use the bit mask to perform a bit set or a bit clear on the input. To perform a bit set, set the **Operator** parameter list to `OR` and create a bit mask with a 1 for each corresponding input bit that you want to set to 1. To perform a bit clear, set the **Operator** parameter list to `AND`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

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 bit clear, the bit mask would be 11101111, which you can specify as `2^7+2^6+2^5+2^3+2^2+2^1+2^0` in the **Bit mask** parameter.

Data Type Support

The Bitwise Operator block supports Simulink integer, fixed-point, and Boolean data types. The block does not support true floating-point data types or enumerated data types.

Parameters and Dialog Box



Operator
The bitwise logical operator associated with the specified operands.

Use bit mask
Specify if the bit mask is used (single input only).

Number of input ports
The number of inputs.

Bit Mask

The bit mask to associate with a single input. The **Bit Mask** parameter is converted from a double to the input data type offline using round-to-nearest and saturation.

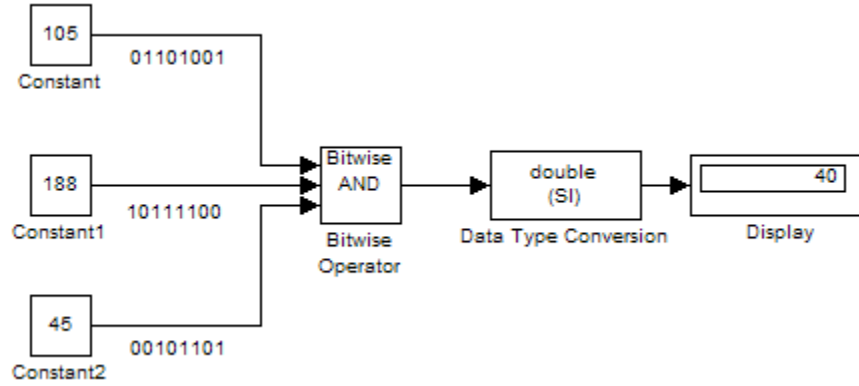
Treat mask as

Treat the mask as a real-world value or a stored integer.

Based on the encoding scheme described in "Scaling" in 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.

Examples

The following fixed-point model shows how the Bitwise Operator block works when inputs are unsigned.

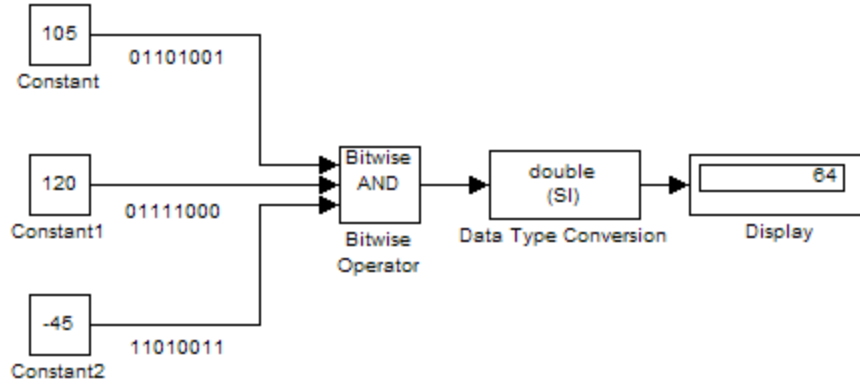


Each Constant block outputs an 8-bit unsigned integer (`uint8`). The results for all logic operations are shown below.

Operation	Binary Value	Decimal Value
AND	00101000	40
OR	11111101	253

Operation	Binary Value	Decimal Value
NAND	11010111	215
NOR	00000010	2
XOR	11111000	248
NOT	N/A	N/A

The following fixed-point model shows how the Bitwise Operator block works when inputs are signed.



Each Constant block outputs an 8-bit signed integer (`int8`). The results for all logic operations are shown below.

Operation	Binary Value	Decimal Value
AND	01000000	64
OR	11111011	-5
NAND	10111111	-65
NOR	00000100	4
XOR	11000010	-62
NOT	N/A	N/A

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of inputs
Multidimensionalized	Yes

Block Support Table

Purpose

View data type support for Simulink blocks

Library

Model-Wide Utilities

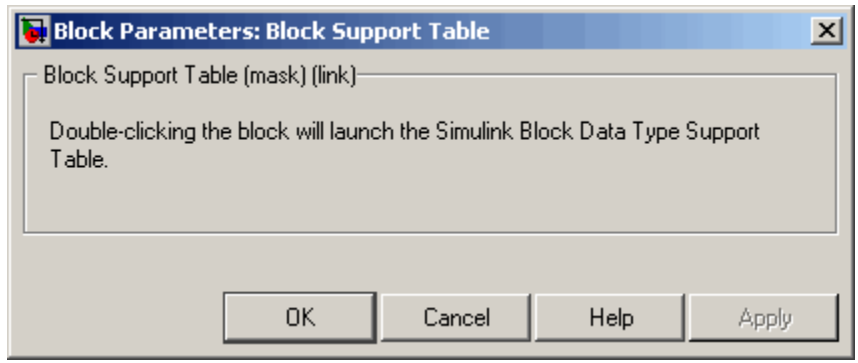
Description

The Block Support Table block helps you access a table that lists the data types that Simulink blocks support. Double-click the block to view the table.

Data Type Support

Not applicable

Parameters and Dialog Box



Characteristics

Not applicable

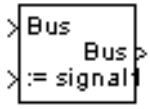
Purpose

Replace specified bus elements

Library

Signal Routing

Description



The Bus Assignment block assigns signals connected to its Assignment input ports (`:=`) to specified elements of the bus connected to its Bus input port, replacing the signals previously assigned to those elements. The change does not affect the signals themselves, it affects only the composition of the bus. Signals not replaced are unaffected by the replacement of other signals. See "Using Composite Signals" for information about buses.

Connect the bus to be changed to the first input port. Use the block's dialog box to specify the bus elements to be replaced. The block displays an assignment input port for each such element. The signal connected to the assignment port must have the same structure, data type, and numeric type as the bus element to which it corresponds.

You cannot use the Bus Assignment block to replace a bus that is nested within another bus. Thus no element selected in the dialog box for replacement can be a bus, and no signal connected to an Assignment port can be a bus.

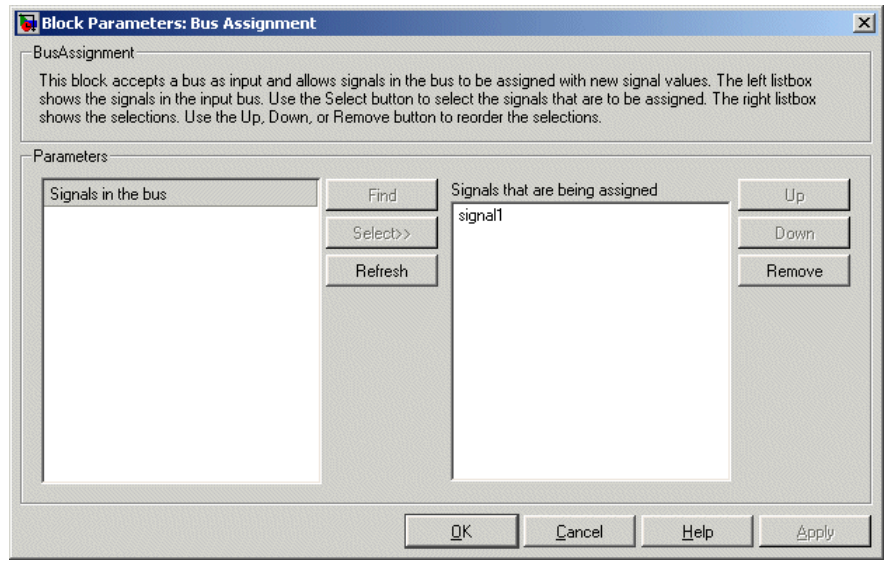
All signals in a nonvirtual bus must have the same sample time, even if the elements of the associated bus object specify inherited sample times. Any bus operation that would result in a nonvirtual bus that violates this requirement generates an error. All buses and signals input to a Bus Assignment block that modifies a nonvirtual bus must therefore have the same sample time. You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus, to allow the signal or bus to be included in a nonvirtual bus. See "Virtual and Nonvirtual Buses" for more information.

Data Type Support

The bus input port of the Bus Assignment block accepts and outputs real or complex values of any data type supported by Simulink software, including fixed-point and enumerated data types. The assignment input ports accept the same data types as the bus elements to which they correspond.

For a discussion on the data types supported by Simulink software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

Parameters and Dialog Box



Signals in the bus

Displays the names of the signals contained by the bus at the block’s Bus input port. Click any item in the list to select it. To find the source of the selected signal, click the adjacent **Find** button. Simulink software opens the subsystem containing the signal source and highlights the source’s icon. Use the **Select>>** button to move the currently selected signal into the adjacent list of signals to be assigned values (see **Signals that are being assigned** below). To refresh the display (e.g., to reflect modifications to the bus connected to the block), click the adjacent **Refresh** button.

Signals that are being assigned

Lists the names of bus elements to be assigned values. This block displays an assignment input port for each bus element in this

list. The label of the corresponding input port contains the name of the element. You can order the signals by using the **Up**, **Down**, and **Remove** buttons. Port connectivity is maintained when the signal order is changed.

Three question marks (`???`) before the name of a bus element indicate that the input bus no longer contains an element of that name, for example, because the bus has changed since the last time you refreshed the Bus Assignment block's input and bus element assignment lists. You can fix the problem either by modifying the bus to include a signal of the specified name or by removing the name from the list of bus elements to be assigned values.

Characteristics

Multidimensionalized	Yes
Virtual	Yes, if the input bus is virtual For more information, see "Virtual Blocks" in the Simulink documentation.

See Also

- "Using Composite Signals"
- Bus Creator
- Bus Selector
- Bus to Vector
- `Simulink.Bus`
- `Simulink.Bus.cellToObject`
- `Simulink.Bus.createObject`
- `Simulink.BusElement`
- `Simulink.Bus.objectToCell`
- `Simulink.Bus.save`

Bus Creator

Purpose

Create signal bus

Library

Signal Routing

Description

The Bus Creator block combines a set of signals into a bus. To bundle a group of signals with a Bus Creator block, set the block's **Number of inputs** parameter to the number of signals in the group. The block displays the number of ports that you specify. Connect the signals to be grouped to the resulting input ports. See "Using Composite Signals" for information about buses.

The signals in the bus will be ordered from the top input port to the bottom input port. See "How to Rotate a Block" in the Simulink documentation for a description of the port order for various block orientations.

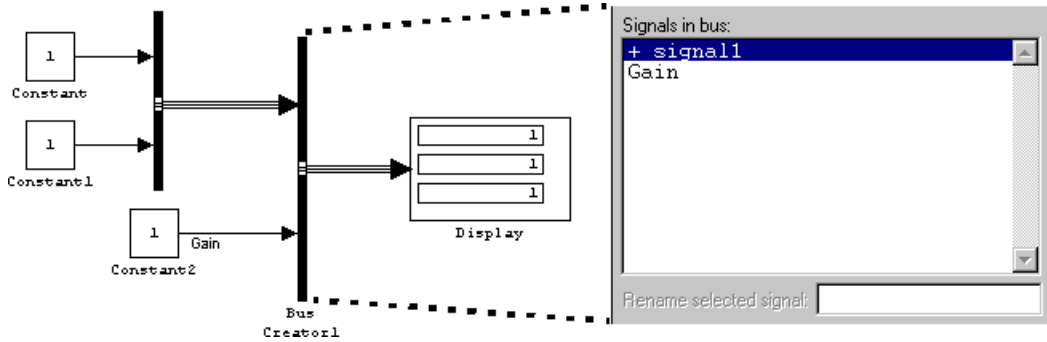
You can connect any type of signal to the inputs, including other bus signals. To ungroup the signals, connect the block's output port to a Bus Selector port.

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

Naming Signals

The Bus Creator block assigns a name to each signal on the bus that it creates. This allows you to refer to signals by name when searching for their sources (see "Browsing Bus Signals" on page 2-53) or selecting signals for connection to other blocks. The block offers two bus signal naming options. You can specify that each signal on the bus inherits the name of the signal connected to the bus (the default) or that each input signal must have a specific name.

To specify that bus signals inherit their names from input ports, select `Inherit bus signal names from input ports` from the list box on the block's parameter dialog box. The names of the inherited bus signals appear in the **Signals in bus** list box.



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 in 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 **Apply** or **OK**.

Browsing Bus Signals

The **Signals in bus** list on a Bus Creator block's parameter dialog box displays a list of the signals entering the block. A plus sign (+) next to a signal indicates that the signal is itself a bus. You can display its

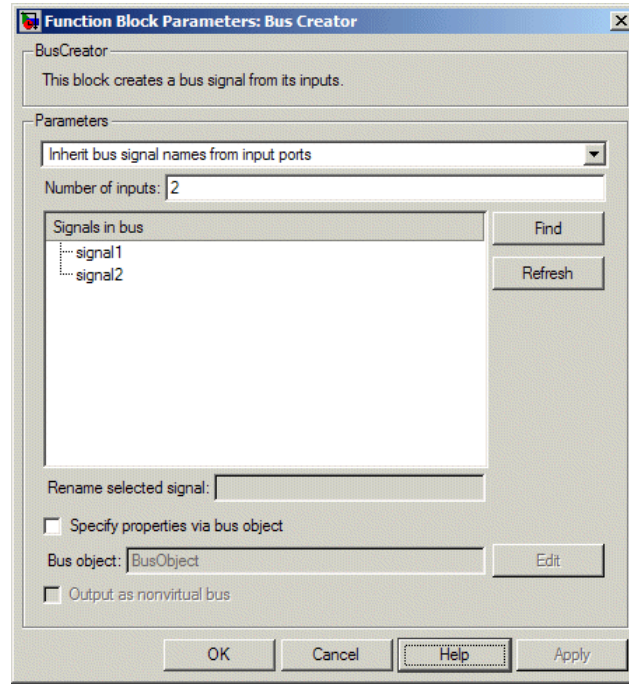
contents by clicking the plus sign. If the expanded input includes bus signals, plus signs appear next to the names of those bus signals. You can expand them as well. In this way, you can view all signals entering the block, including those entering via buses. To find the source of any signal entering the block, select the signal in the **Signals in bus** list and click the adjacent **Find** button. Simulink software opens the subsystem containing the signal source, if necessary, and highlights the source's icon.

Data Type Support

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

For a discussion on the data types supported by Simulink software, refer to "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Signal naming options

Assign input signal names to the corresponding bus signals.

Settings

Default: `Inherit bus signal names from input ports`

`Inherit bus signal names from input ports`
: Assign input signal names to the corresponding bus signals.

`Require input signal names to match signals below`
: Inputs must have the names listed in the **Signals in bus** list.

Dependencies

Selecting `Require input signal names to match signals below` enables **Rename selected signal**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Number of inputs

Specify the number of input ports on this block.

Settings

Default: 2

To bundle a group of signals, enter the number of signals in the group.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Signals in bus

Show the signals in the output bus.

Settings

When you modify the **Number of inputs** parameter, click **Refresh** to update the list of signals.

Tips

- 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 the name of an input signal.
- Click the **Find** button to highlight the source of the currently selected signal.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Rename selected signal

List the name of the signal currently selected in the **Signals in bus** list when you select the `Require input signal names to match signals below` option.

Settings

Default: `''`

Edit this field to change the name of the currently selected signal.

Dependencies

Selecting `Require input signal names to match signals below` for **Parameters** and `signal1` or `signal2` for **Signals in bus** enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Specify properties via bus object

Use a bus object to define the structure of the bus created by this block.

Settings

Default: Off

 On

Use a bus object to define the structure of the bus created by this block.

 Off

Do not use a bus object to define the structure of the bus created by this block.

Tips

Selecting this parameter is required when creating a nonvirtual bus, and optional when creating a virtual bus.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Bus object

Specify the name of the bus object used to define the structure of the bus created by this block.

Settings

Default: `BusObject`

The default value is a dummy bus object name. Type in the name of the bus object you want to use. If you need to create or change a bus object, click **Edit** to the left of the **Bus object** field to open the Simulink Bus Editor. See "Using the Bus Editor" for more information.

Tips

At the beginning of a simulation or when you update the model's diagram, Simulink software checks whether the signals connected to this Bus Creator block have the specified structure. If not, Simulink software displays an error message.

Dependencies

Specify properties via bus object enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output as nonvirtual bus

Output a nonvirtual bus.

Settings

Default: Off

 On

Output a nonvirtual bus.

 Off

Output a virtual bus.

Tips

- 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.
- All signals in a nonvirtual bus must have the same sample time, even if the elements of the associated bus object specify inherited sample times. Any bus operation that would result in a nonvirtual bus that violates this requirement generates an error. Therefore, if you select this option all signals entering the Bus Creator block must have the same sample time. You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus, to allow the signal or bus to be included in a nonvirtual bus.

Dependencies

Specify properties via bus object enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Characteristics

Multidimensionalized	Yes
Virtual	Yes, if the output bus is virtual For more information, see "Virtual Blocks" in the Simulink documentation.

See Also

- "Using Composite Signals"
- Bus Assignment
- Bus Selector
- Bus to Vector
- `Simulink.Bus`
- `Simulink.Bus.cellToObject`
- `Simulink.Bus.createObject`
- `Simulink.BusElement`
- `Simulink.Bus.objectToCell`
- `Simulink.Bus.save`

Bus Selector

Purpose

Select signals from incoming bus

Library

Signal Routing

Description

The Bus Selector block outputs a specified subset of the elements of the bus at its input. The block can output the specified elements as separate signals or as a new bus. See “Using Composite Signals” for information about buses.

When the block outputs separate elements, it outputs each element from a separate port from top to bottom of the block. See “How to Rotate a Block” for a description of the port order for various block orientations. See “Using Composite Signals” for information about buses.

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

Caution

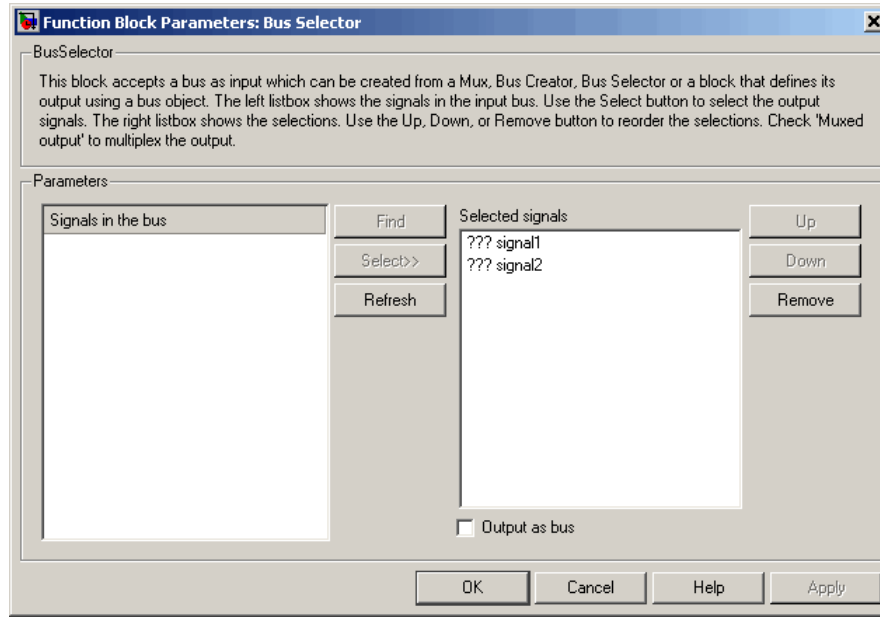
The MathWorks recommends not using Bus Selector blocks in library blocks, because such use complicates changing the library blocks and increases the likelihood of errors. See “Buses and Libraries” for more information.

Data Type Support

A Bus Selector block accepts and outputs real or complex values of any data type supported by Simulink software, including fixed-point and enumerated data types.

For a discussion on the data types supported by Simulink software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

Parameters and Dialog Box



Signals in the bus

Shows the signals in the input bus.

Settings

To refresh the display to reflect modifications to the bus connected to the block, click **Refresh**.

Tips

- Use **Select>>** to select signals to output.
- To find the source of any signal entering the block, select the signal in the list and click **Find**. The Simulink software opens the subsystem containing the signal source, and highlights the source's icon.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Selected signals

Shows the signals to be output.

Settings

Default: `signal1,signal2`

You can change the list by using the **Up**, **Down**, and **Remove** buttons.

Tips

- 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 (`???`).

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output as bus

Output the selected elements as a bus.

Settings

Default: Off

 On

Output the selected elements as a bus.

 Off

Output the selected elements as standalone signals, each from an output port that is labeled with the corresponding element's name.

Tips

The output bus is virtual if the input bus is virtual, or nonvirtual if the input bus is nonvirtual.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Multidimensionalized	Yes
Virtual	Yes, if the input bus is virtual For more information, see "Virtual Blocks" in the Simulink documentation.

See Also

- "Using Composite Signals"
- Bus Assignment
- Bus Creator
- Bus to Vector
- `Simulink.Bus`
- `Simulink.Bus.cellToObject`

- `Simulink.Bus.createObject`
- `Simulink.BusElement`
- `Simulink.Bus.objectToCell`
- `Simulink.Bus.save`

Bus to Vector

Purpose

Convert virtual bus to vector

Library

Signal Attributes

Description

The Bus to Vector block converts a virtual bus signal to a vector signal. The input bus signal must consist of scalar, 1-D, or either row or column vectors having the same data type, signal type, and sampling mode. If the input bus contains row or column vectors, this block outputs a row or column vector, respectively; otherwise, it outputs a 1-D array.

Use the Bus to Vector block only to replace an implicit bus-to-vector conversion with an equivalent explicit conversion. See “Bus signal treated as vector” and “Correcting Buses Used as Muxes” for more information.

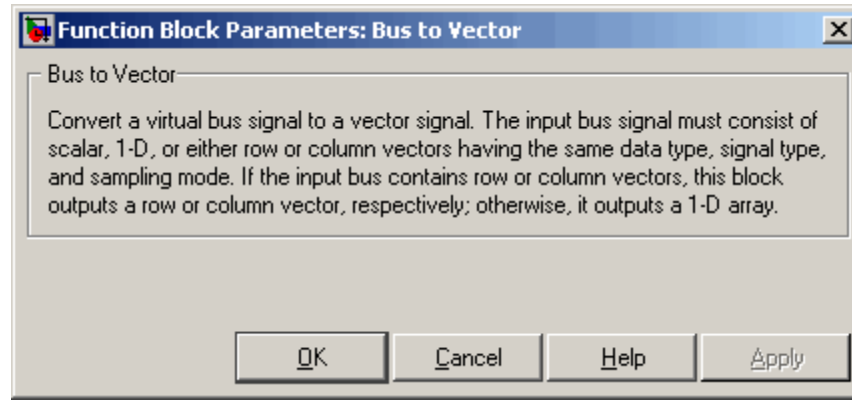
Note Simulink software hides the name of a Bus to Vector block when you copy it from the Simulink library to a model.

Data Type Support

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

For a discussion of the data types supported by Simulink software, refer to “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

Parameters and Dialog Box



This block has no user-accessible parameters.

Characteristics

Multidimensionalized	Yes

See Also

- "Using Composite Signals"
- Avoiding Mux/Bus Mixtures
- Bus Assignment
- Bus Creator
- Bus Selector
- `Simulink.BlockDiagram.addBusToVector`
- `Simulink.Bus`
- `Simulink.Bus.cellToObject`
- `Simulink.Bus.createObject`
- `Simulink.BusElement`
- `Simulink.Bus.objectToCell`
- `Simulink.Bus.save`

Check Discrete Gradient

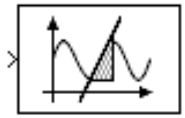
Purpose

Check that absolute value of difference between successive samples of discrete signal is less than 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. Use the block parameter dialog box 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 under **Debugging** on the **Data Validity** 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.

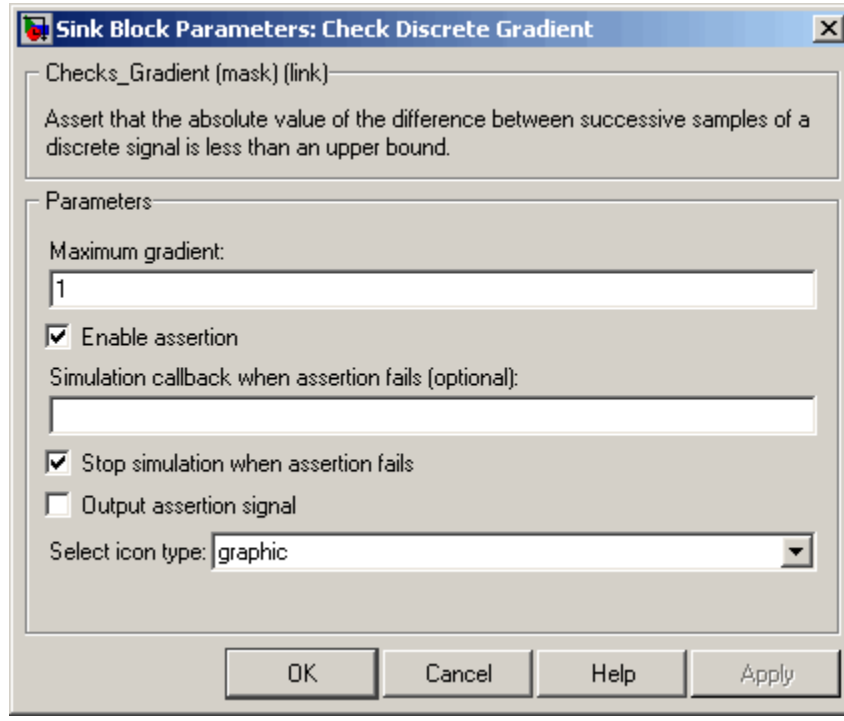
The Check Discrete Gradient block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied 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.

Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

The Check Discrete Gradient block accepts `single`, `double`, `int8`, `int16`, and `int32` input signals of any dimensions. This block also supports fixed-point data types.

Parameters and Dialog Box



Maximum gradient
Upper bound on the gradient of the discrete input signal.

Enable assertion
Clearing this check box 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 under **Debugging** on the **Data Validity** diagnostics pane of the Configuration Parameters dialog box allows you to enable or disable all model verification blocks in a model, including Check Discrete Gradient blocks, regardless of the setting of this option.

Simulation callback when assertion fails
A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails
Selecting this check box causes the Check Discrete Gradient block to halt the simulation when the block's output is zero and display an error message in the Simulink Simulation Diagnostics viewer. Otherwise, the block displays a warning message in the MATLAB Command Window and continues the simulation.

Output assertion signal
Selecting this check box causes the Check Discrete Gradient 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** check box on the **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 block.

Characteristics

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

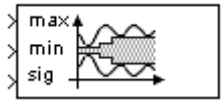
Purpose

Check that gap of possibly varying width occurs in range of 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.

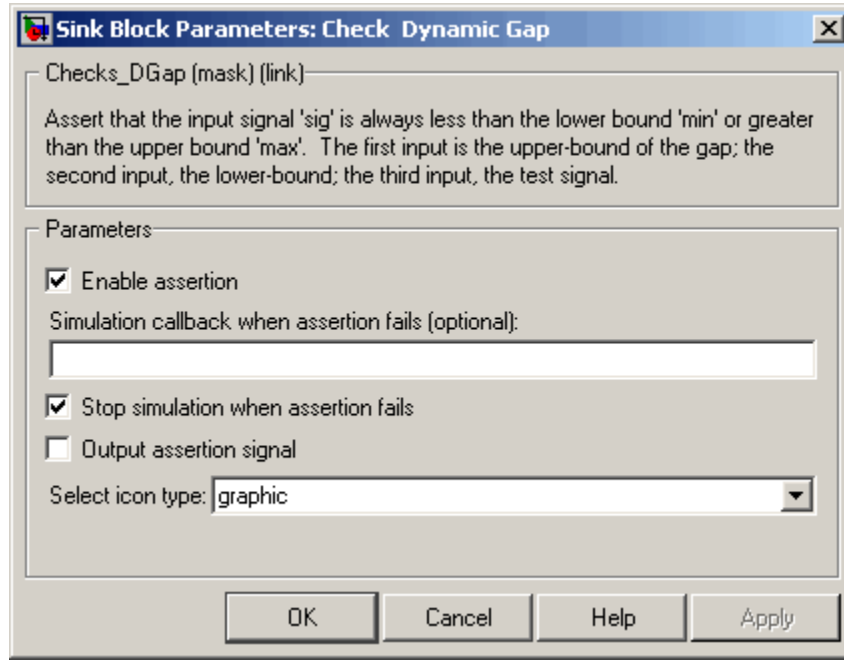
Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

The Check Dynamic Gap block accepts input signals of any dimensions and of any numeric data type supported by Simulink software. 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 software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

Parameters and Dialog Box



Enable assertion

Clearing this check box 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 under **Debugging** on the **Data Validity** 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

A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails

Selecting this check box causes the Check Dynamic Gap block to halt the simulation when the block's output 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

Selecting this check box causes the Check Dynamic Gap 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** check box on the **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 block.

Characteristics

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

Check Dynamic Lower Bound

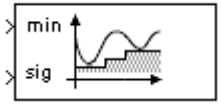
Purpose

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 reference signal is less than the amplitude of a test 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.

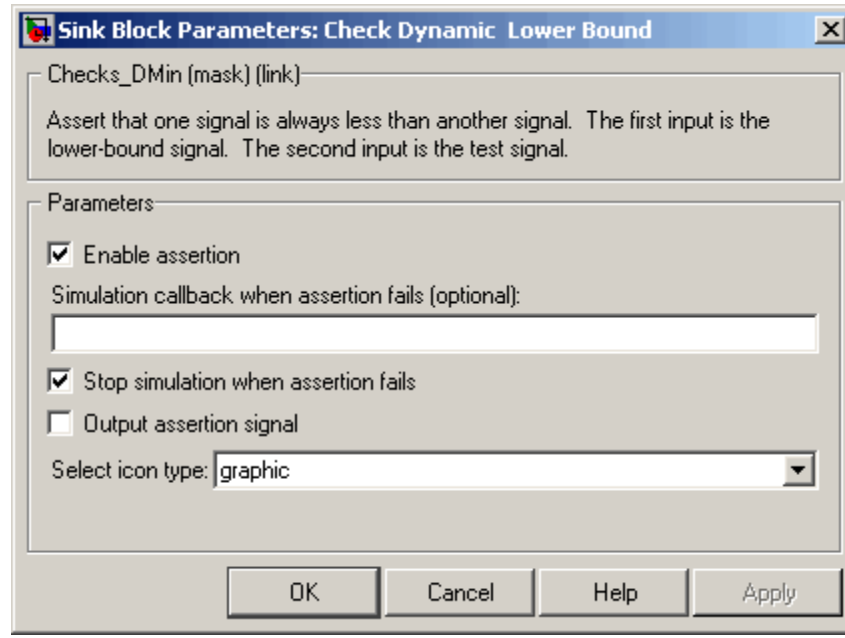
Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

The Check Dynamic Lower Bound block accepts input signals of any numeric data type supported by Simulink software. 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 software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Enable assertion

Clearing this check box 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 under **Debugging** on the **Data Validity** 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

A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails

Selecting this check box causes the Check Dynamic Lower Bound block to halt the simulation when the block's output 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

Selecting this check box causes the Check Dynamic Lower Bound 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** check box on the **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 block.

Characteristics

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

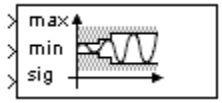
Purpose

Check that signal falls inside 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.

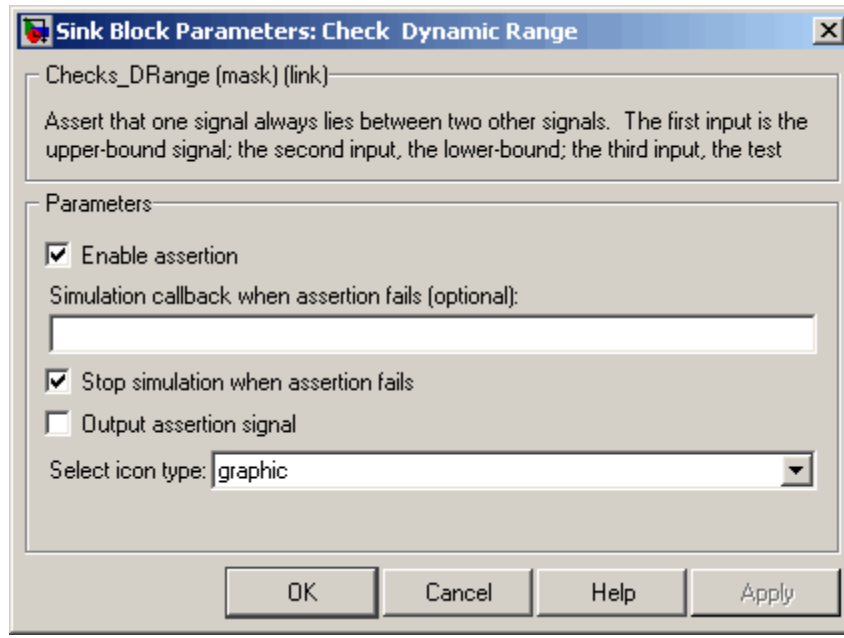
Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

The Check Dynamic Range block accepts input signals of any dimensions and of any numeric data type supported by Simulink software. 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 software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

Parameters and Dialog Box



Enable assertion

Clearing this check box 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 under **Debugging** on the **Data Validity** 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

A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails
Selecting this check box causes the Check Dynamic Range block to halt the simulation when the block's output 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
Selecting this check box causes the Check Dynamic Range 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 selected the **Implement logic signals as Boolean data** check box on the **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 block.

Characteristics

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

Check Dynamic Upper Bound

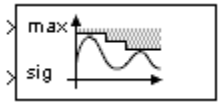
Purpose

Check that one signal is always greater than another signal

Library

Model Verification

Description



The Check Dynamic Upper Bound block checks that the amplitude of a reference signal is greater than the amplitude of a test 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 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.

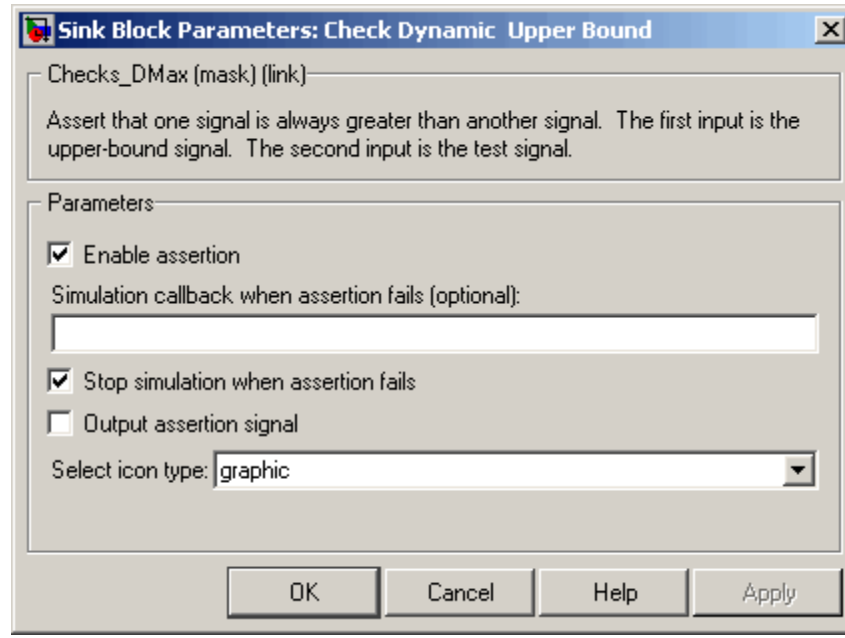
Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

The Check Dynamic Upper Bound block accepts input signals of any dimensions and of any numeric data type supported by Simulink software. The test and the reference signals must have the same dimensions and data type. If the inputs are nonscalar, the block compares 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 software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Enable assertion

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

Simulation callback when assertion fails

A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails

Selecting this check box causes the Check Dynamic Upper Bound block to halt the simulation when the block's output 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

Selecting this check box causes the Check Dynamic Upper Bound 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** check box on the **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 block.

Characteristics

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

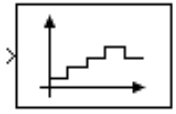
Purpose

Check that input signal has specified resolution

Library

Model Verification

Description



The Check Input Resolution block checks whether the input signal has a specified scalar or vector resolution (see Resolution). If the resolution is a scalar, the input signal must be a multiple of the resolution within a `10e-3` tolerance. If the resolution is a vector, the input signal must equal an element of the resolution vector. 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 Input Resolution 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.

Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

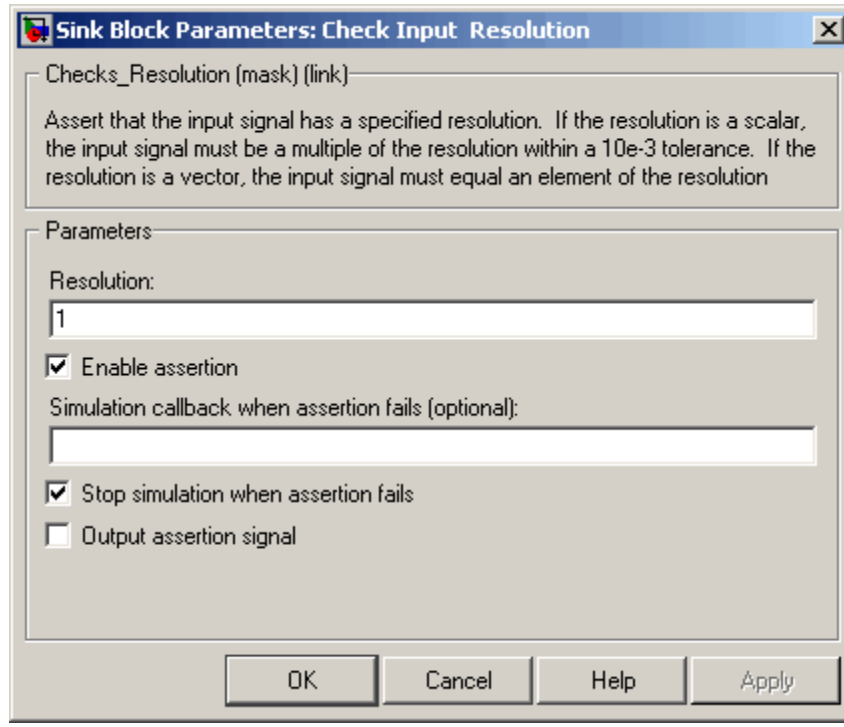
Data Type Support

The Check Input Resolution block accepts input signals of data type `double` and of any dimension. If the input signal is nonscalar, the block checks the resolution of each element of the input test signal.

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Check Input Resolution

Parameters and Dialog Box



Resolution

Resolution that the input signal must have.

Enable assertion

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

Simulation callback when assertion fails

A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails

Selecting this check box causes the Check Input Resolution block to halt the simulation when the block's output 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

Selecting this check box causes the Check Input Resolution 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** check box on the **Optimization** pane of the Configuration Parameters dialog box. Otherwise the data type of the output signal is `double`.

Characteristics

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

Check Static Gap

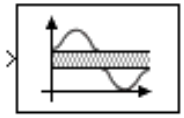
Purpose

Check that gap exists in signal's range of amplitudes

Library

Model Verification

Description



The Check Static Gap block checks that each element of the input signal is less than (or optionally equal to) a static lower bound or greater than (or optionally equal to) a static upper bound 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 Static 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.

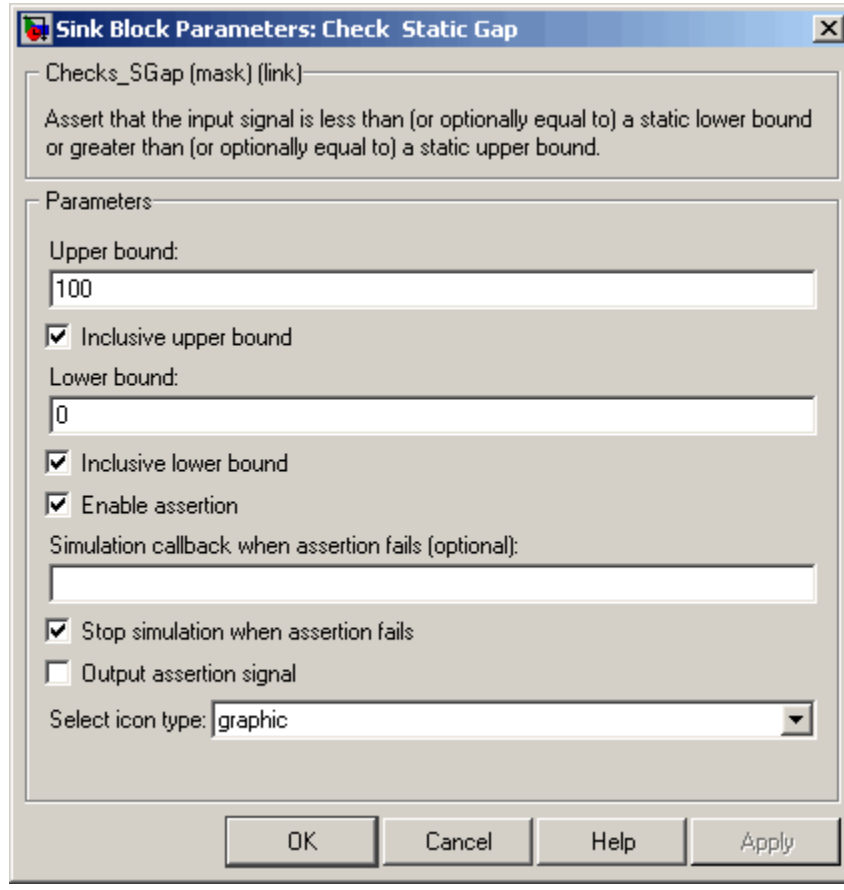
Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Upper bound

Upper bound of the gap in the input signal's range of amplitudes.

Inclusive upper bound

Selecting this check box specifies that the gap includes the upper bound.

Lower bound

Lower bound of the gap in the input signal's range of amplitudes.

Inclusive lower bound
Selecting this check box specifies that the gap includes the lower bound.

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

Simulation callback when assertion fails
A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails
Selecting this check box causes the Check Static Gap block to halt the simulation when the block's output 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
Selecting this check box causes the Check Static Gap 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** check box on the **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 block.

Characteristics

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

Check Static Lower Bound

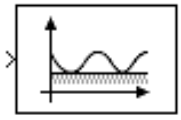
Purpose

Check that signal is greater than (or optionally equal to) 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. Use the block parameter dialog box 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.

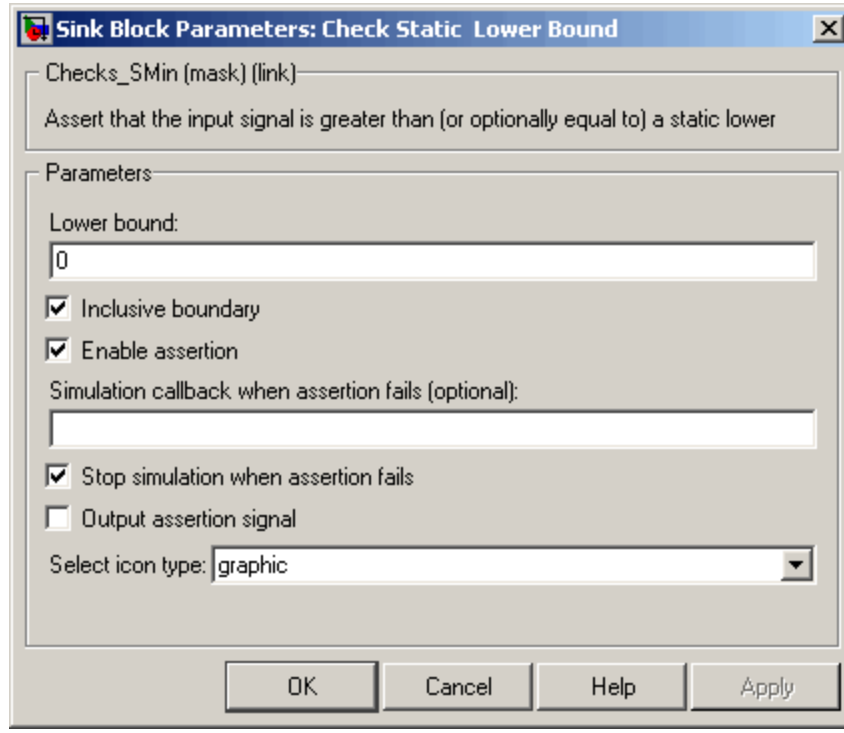
Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Lower bound

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

Inclusive boundary

Selecting this check box makes the range of valid input amplitudes include the lower bound.

Enable assertion

Clearing this check box 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 under **Debugging** on the **Data Validity** diagnostics pane of the Configuration Parameters dialog box allows you to enable or

disable all model verification blocks in a model, including Check Static Lower Bound blocks, regardless of the setting of this option.

Simulation callback when assertion fails
A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails
Selecting this check box causes the Check Static Lower Bound block to halt the simulation when the block's output 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
Selecting this check box causes the Check Static Lower Bound 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** check box on the **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 block.

Characteristics

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

Multidimensionalized	Yes
Zero-Crossing Detection	No

Check Static Range

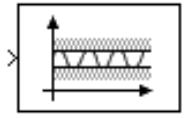
Purpose

Check that signal falls inside 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. Use the block parameter dialog box 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.

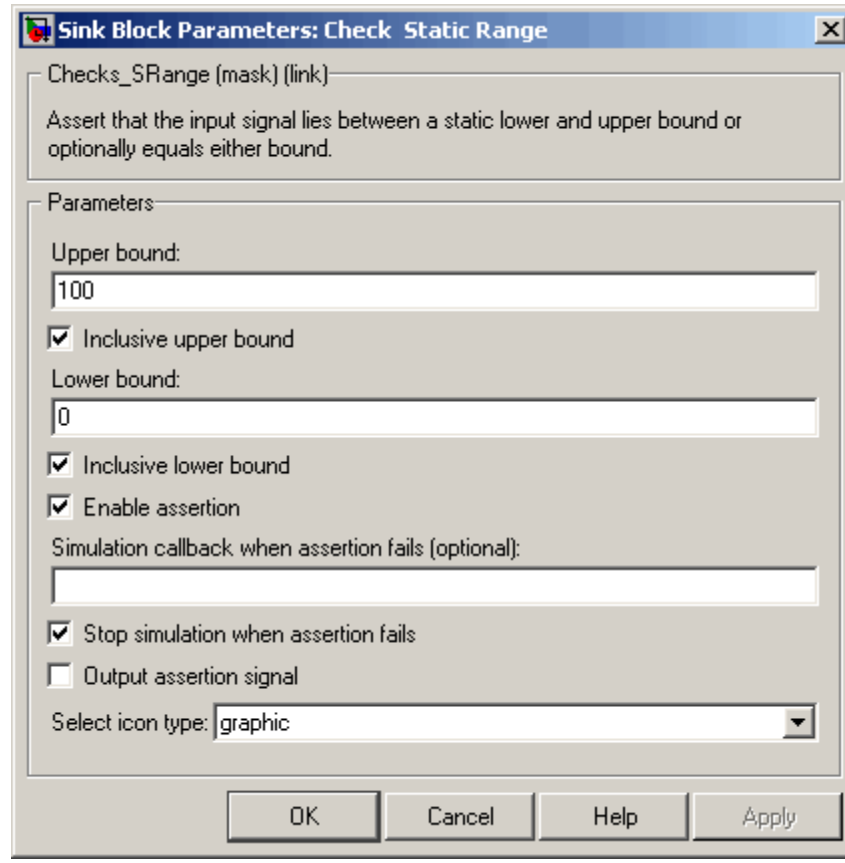
Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Upper bound

Upper bound of the range of valid input signal amplitudes.

Inclusive upper bound

Selecting this check box 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
Selecting this check box specifies that the valid signal range includes the lower bound.

Enable assertion
Clearing this check box 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 under **Debugging** on the **Data Validity** diagnostics pane of the Configuration Parameters dialog box allows you to enable or disable all model verification blocks in a model, including Check Static Range blocks, regardless of the setting of this option.

Simulation callback when assertion fails
A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails
Selecting this check box causes the Check Static Range block to halt the simulation when the block's output 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
Selecting this check box causes the Check Static Range 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** check box on the **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 block.

Characteristics

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

Check Static Upper Bound

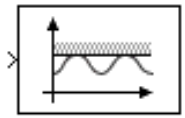
Purpose

Check that signal is less than (or optionally equal to) static upper 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 upper bound at the current time step. Use the block parameter dialog box 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.

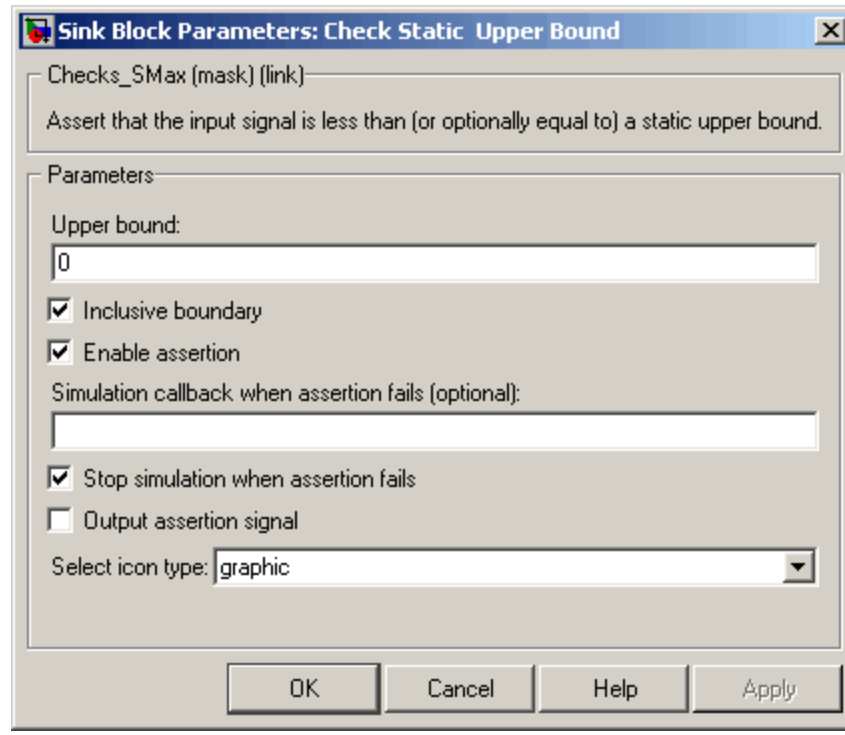
Note For information about how Real-Time Workshop generated code handles Model Verification blocks, see "Enabling Instrumentation for Debugging" in the Real-Time Workshop User's Guide.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Upper bound

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

Inclusive boundary

Selecting this check box makes the range of valid input amplitudes include the upper bound.

Enable assertion

Clearing this check box 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 under **Debugging** on the **Data Validity** diagnostics pane of the Configuration Parameters dialog box allows you to enable or

disable all model verification blocks in a model, including Check Static Upper Bound blocks, regardless of the setting of this option.

Simulation callback when assertion fails
A MATLAB expression to evaluate when the assertion fails.

Stop simulation when assertion fails
Selecting this check box causes the Check Static Upper Bound block to halt the simulation when the block's output 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
Selecting this check box causes the Check Static Upper Bound 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** check box on the **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 block.

Characteristics

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

Multidimensionalized	Yes
Zero-Crossing Detection	No

Chirp Signal

Purpose

Generate sine wave with increasing frequency

Library

Sources

Description

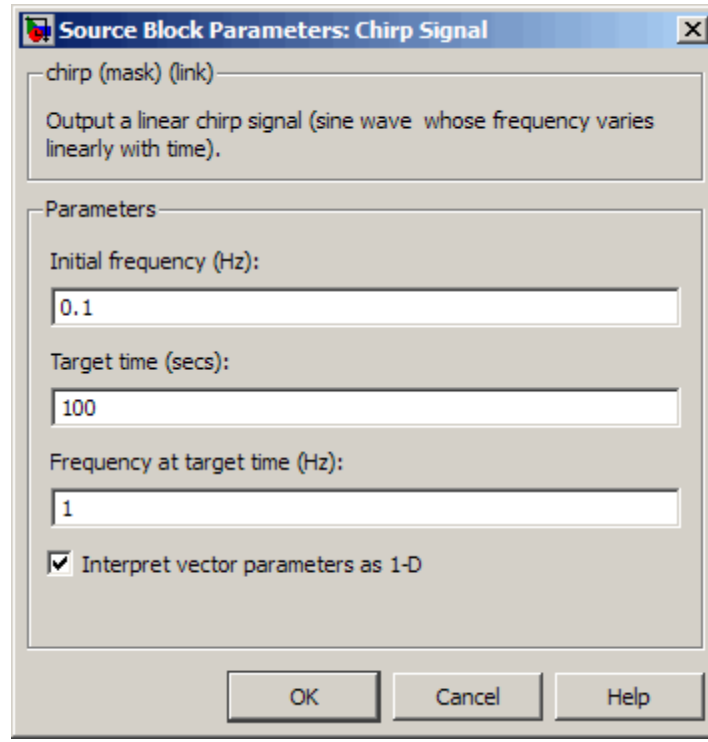
The Chirp Signal block generates a sine wave whose frequency increases at a linear rate with time. You can use this block for spectral analysis of nonlinear systems. The block generates a scalar or vector output.

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** check box. If you select this check box 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 During Simulation" in the "Working with Blocks" chapter of the Simulink documentation.

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. See "Determining the Output Dimensions of Source Blocks" in the "Working with Signals" chapter of the Simulink documentation.

Characteristics

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

Purpose

Display and provide simulation time

Library

Sources

Description

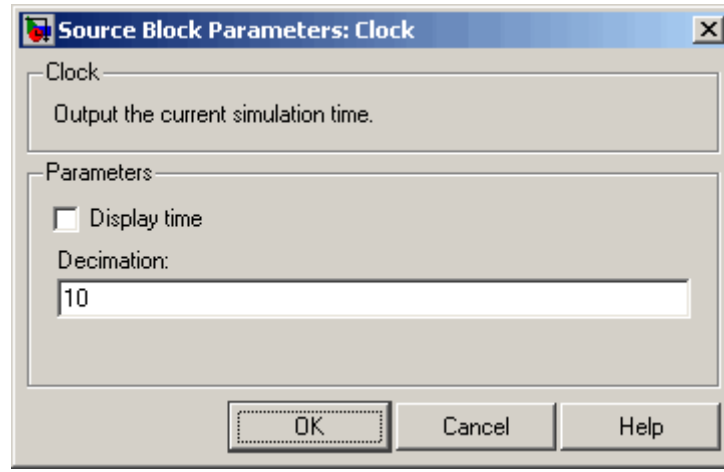
The Clock block outputs the current simulation time at each simulation step. This block is useful for other blocks that need the simulation time.

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 Simulink software updates the Clock icon when **Display time** is checked. Specify a positive integer (the default is 10). For

example, if the decimation is 1000, then, for a fixed integration step of 1 millisecond, the Clock icon updates at 1 second, 2 seconds, and so on.

Characteristics

Sample Time	Continuous
Scalar Expansion	N/A
Dimensionalized	No
Zero Crossing	No

Purpose

Implement 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

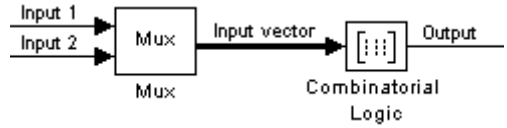
```
number of rows = 2 ^ (number of inputs)
```

Simulink software returns a row of the matrix by computing the row's index from the input vector elements. Simulink software 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:

```
(2^1*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

Inputs			Outputs	
a	b	c	c′	s
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.

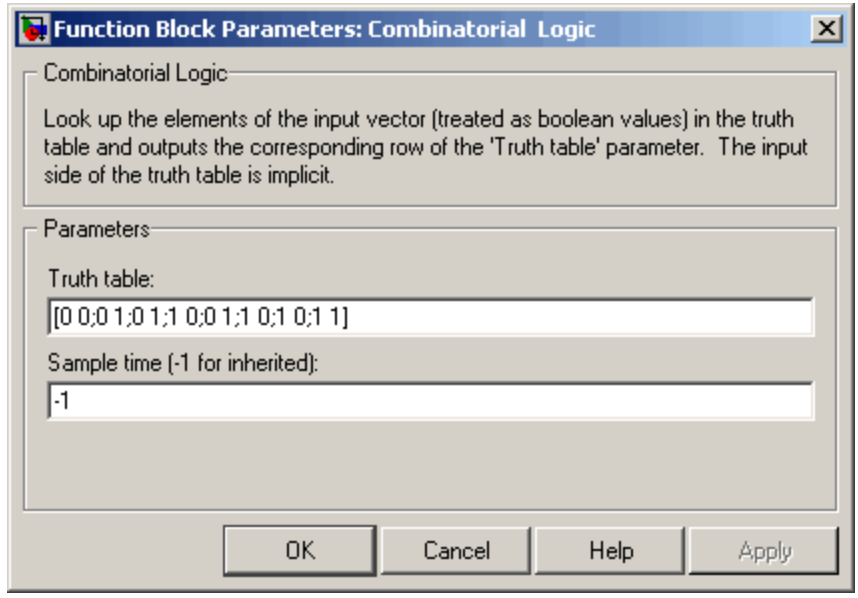
Data Type Support

The type of signals accepted by a Combinatorial Logic block depends on whether you selected the Boolean logic signals option (see "Implement logic signals as Boolean data (vs. double)"). If this option is enabled, the block accepts real signals of type `Boolean` or `double`. The **Truth table** parameter can have Boolean values (0 or 1) of any data type, including fixed-point data types. If the table contains non-Boolean values, the data type of the table 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 "How to Specify the Sample Time" in the "How Simulink Works" chapter of the Simulink documentation.

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

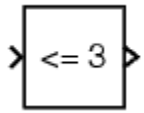
Purpose

Determine how signal compares to 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.

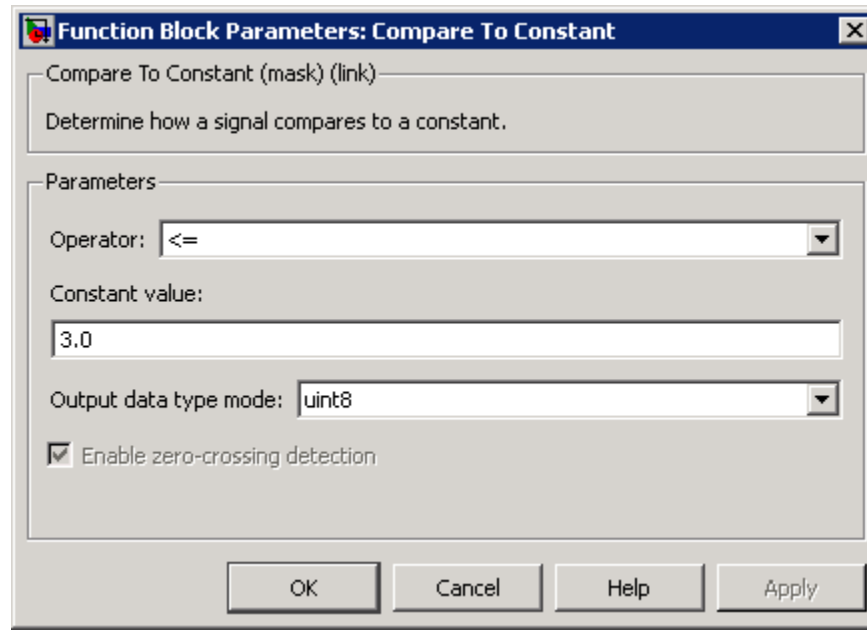
The output is `0` if the comparison is false, and `1` if it is true.

Data Type Support

The Compare To Constant block accepts inputs of any data type supported by Simulink software, including fixed-point and enumerated data types. The block first converts its **Constant value** parameter to the input data type, and then performs the specified operation. 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 software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink User’s Guide.

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 the Simulink documentation.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes
Multidimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled.

See Also

Compare To Zero

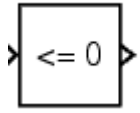
Purpose

Determine how 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.

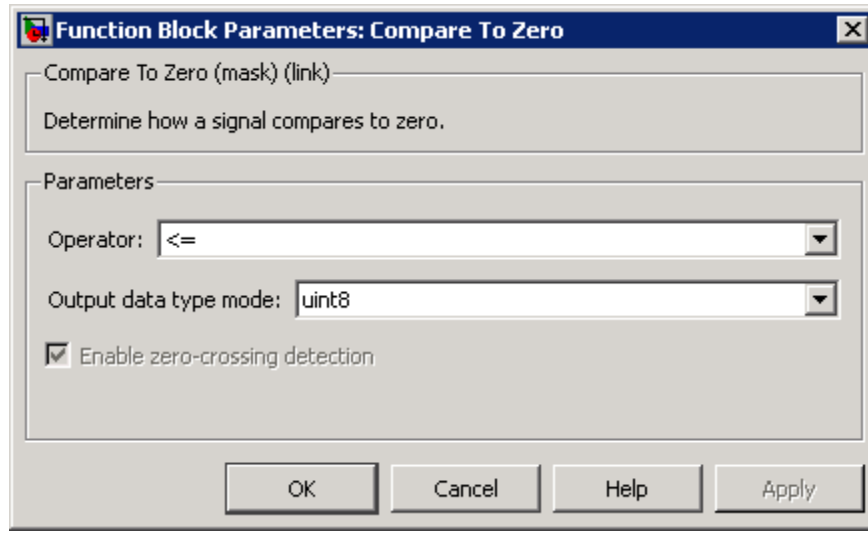
The output is `0` if the comparison is false, and `1` if it is true.

Data Type Support

The Compare To Zero block accepts inputs of any numeric data type supported by Simulink software, 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 software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink User’s Guide.

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 the Simulink documentation.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes
Multidimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled.

See Also

Compare To Constant

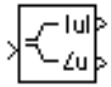
Purpose

Compute magnitude and/or phase angle of complex signal

Library

Math Operations

Description

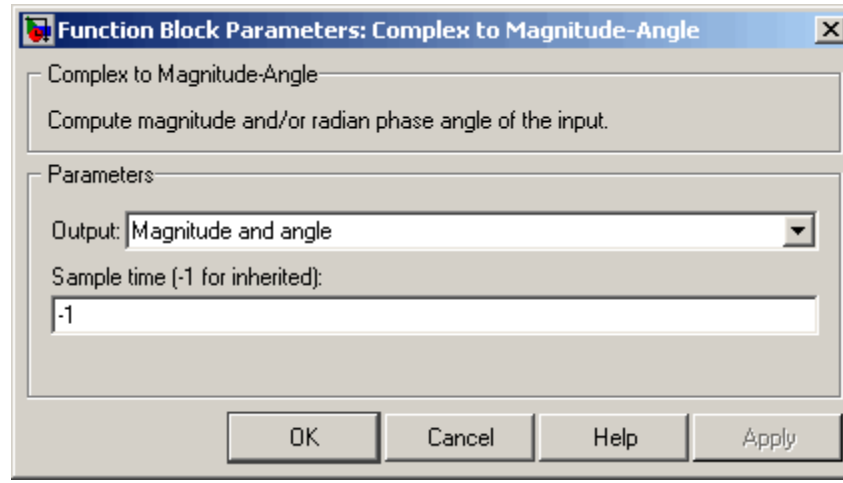


The Complex to Magnitude-Angle block accepts a complex-valued signal of type `double` or `single`. It outputs the magnitude and/or phase angle of the input signal, depending on the setting of the **Output** parameter. The outputs are real values of the same data type as the block input. The input can 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: `Magnitude and angle` (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 "How to Specify the Sample Time" in the "How Simulink Works" chapter of the Simulink documentation.

Characteristics

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

Purpose

Output real and imaginary parts of complex input signal

Library

Math Operations

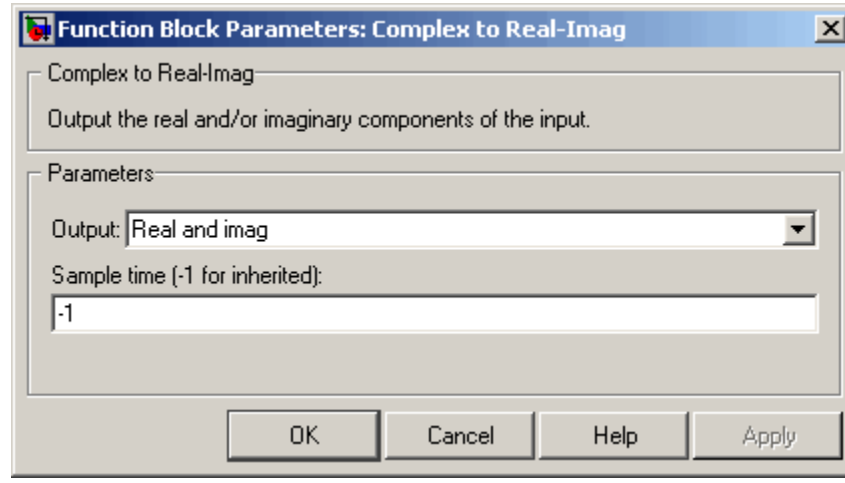
Description

The Complex to Real-Imag block accepts a complex-valued signal of any data type supported by Simulink software, 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 software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Output

Determines the output of this block. Choose from the following values: `Real and imag` (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 "How to Specify the Sample Time" in the "How Simulink Works" chapter of the Simulink documentation.

Characteristics

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

Purpose

Represent any block selected from 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 software derives the port names for the configurable subsystem by making a unique list from the port names of all the choices. Note that Simulink software uses default port names for non-subsystem block choices.

Note that Simulink software does not allow you to break library links in a configurable subsystem because Simulink software 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 Ports & Subsystems 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. Under **List of block choices** in the dialog box, select the blocks that represent the various configurations of the configurable subsystems you are creating.
6. Click the **OK** button to apply the changes and close the dialog box.
7. Select **Block Choice** from the Configurable Subsystem block's context menu.

   The context menu displays a submenu listing the blocks that the subsystem can represent.
8. Select the block that you want the subsystem to represent by default.
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', ...);
```

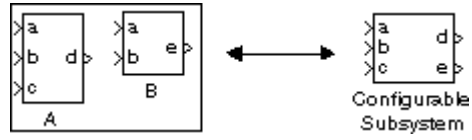
Mapping I/O Ports

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



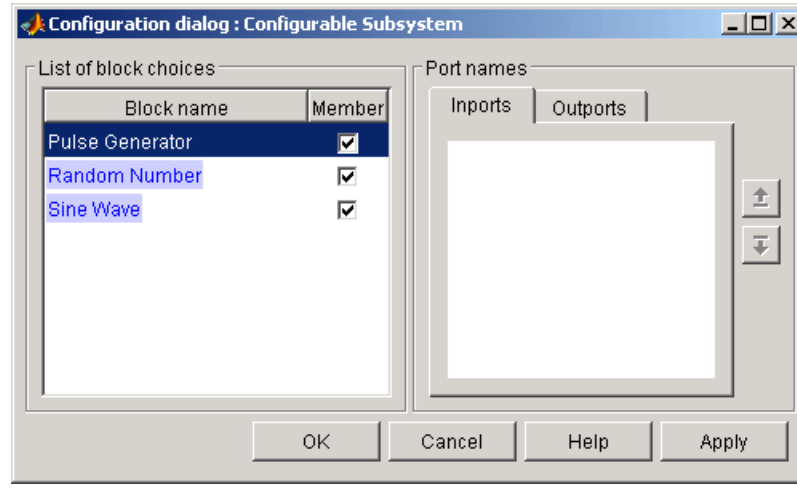
In this example, port `a` on the Configurable Subsystem block connects to port `a` of the selected library block no matter which block is selected. On the other hand, port `c` on the Configurable Subsystem block functions only if library block A is selected. Otherwise, it simply terminates.

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 software, 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.

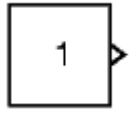
Purpose

Generate constant value

Library

Sources

Description



The Constant block generates a real or complex constant value. The block generates scalar, vector, or matrix output, depending on the dimensionality of the **Constant value** parameter and the setting of the **Interpret vector parameters as 1-D** parameter. Also, the block can generate either a sample-based or frame-based signal, depending on the setting of the **Sampling mode** 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, select the **Interpret vector parameters as 1-D** parameter. Otherwise, the block treats the **Constant value** parameter as a matrix.

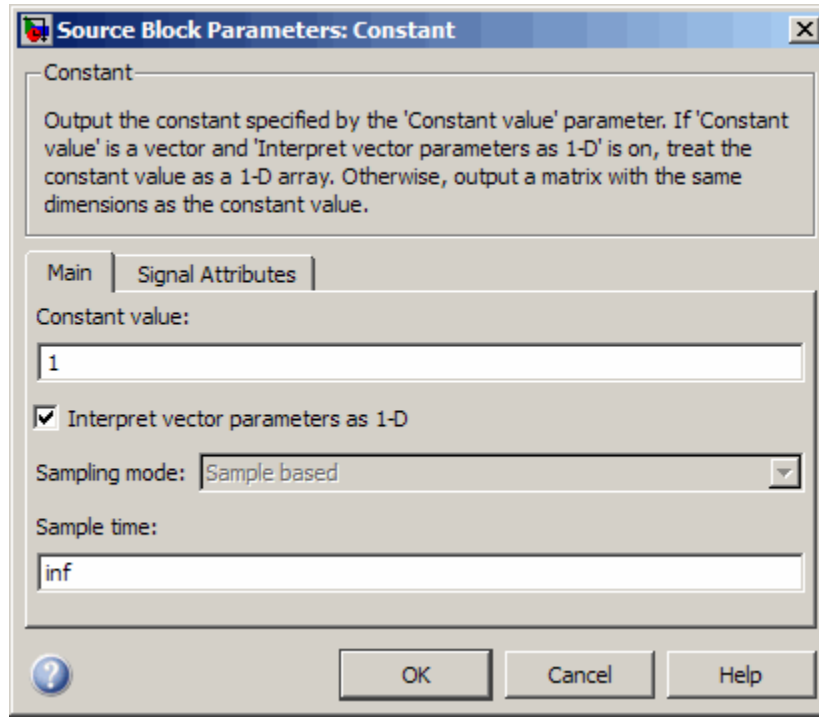
Data Type Support

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 data type supported by Simulink software, including fixed-point and enumerated data types. The Enumerated Constant block may be more convenient than the Constant block for outputting a constant enumerated value.

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

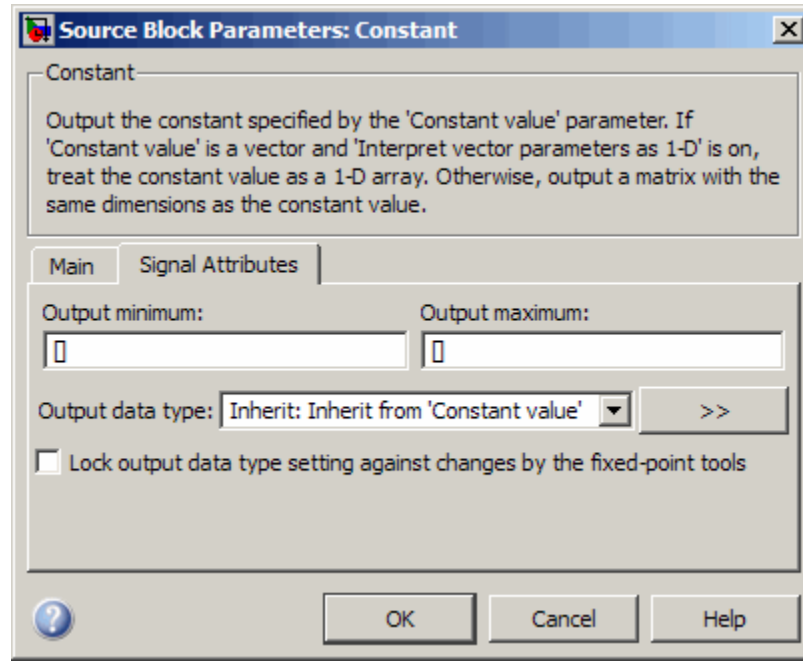
Parameters and Dialog Box

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



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters During Simulation" in the "Working with Blocks" chapter of the Simulink documentation.

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


Source Block Parameters: Constant
Constant
Output the constant specified by the 'Constant value' parameter. If 'Constant value' is a vector and 'Interpret vector parameters as 1-D' is on, treat the constant value as a 1-D array. Otherwise, output a matrix with the same dimensions as the constant value.
Main
Signal Attributes
Output minimum:
Output maximum:
[]
[]
Output data type:
Inherit: Inherit from 'Constant value'
>>
Lock output data type setting against changes by the fixed-point tools
OK
Cancel
Help

Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User's Guide* for more information.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Constant value

Specify the constant value output of the block.

Settings

Default: 1

Minimum: value from the **Output minimum** parameter

Maximum: value from the **Output maximum** parameter

- You can enter any expression that MATLAB evaluates as a matrix, including the Boolean keywords `true` and `false`.
- This parameter is converted from its data type to the specified output data type offline using round toward nearest and saturation.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Interpret vector parameters as 1-D

Select this check box to output a vector of length `N` if the **Constant value** parameter evaluates to an `N`-element row or column vector.

Settings

Default: On

 On

Outputs a vector of length `N` if the **Constant value** parameter evaluates to an `N`-element row or column vector. For example, the block outputs a matrix of dimension `1-by-N` or `N-by-1`.

 Off

Does not output a vector of length `N` if the **Constant value** parameter evaluates to an `N`-element row or column vector.

If you clear this check box, you can interact with the **Sampling mode** parameter.

Dependencies

This parameter enables **Sampling mode**.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Sampling mode

Specify whether the output signal is `Sample based` or `Frame based`.

Settings

Default: `Sample based`

`Sample based`
: The output signal is sample-based.

`Frame based`
: The output signal is frame-based.

Tips

To generate frame-based signals, you must have the Signal Processing Blockset™ product installed.

For more information, see "Sample-Based Signals" and "Frame-Based Signals" in the Signal Processing Blockset User's Guide.

Dependencies

Clearing **Interpret vector parameters as 1-D** enables this parameter.

Selecting `Sample based` enables the following parameter:

- **Sample time**

Selecting `Frame based` enables the following parameter:

- **Frame period**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sample time

Specify the interval between times that the Constant block output can change during simulation (for example, due to tuning the **Constant value** parameter).

Settings

Default: `inf`

This setting indicates that the block output can never change. This setting speeds simulation and generated code by avoiding the need to recompute the block output.

See "How to Specify the Sample Time" in the online documentation for more information.

Dependency

Sampling mode enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Frame period

Specify the interval between frames that the Constant block output can change during simulation (for example, due to tuning the **Constant value** parameter).

Settings

Default: `inf`

Dependency

Sampling mode enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Output minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `Inherit: Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block), `Inherit: Inherit from 'Constant value'` (Constant block), `Inherit: Inherit via back propagation` (Data Type Conversion block), `Inherit: Same as input` (Saturation block)

`Inherit: Inherit via internal rule`
: This option appears for some blocks. Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the calculated 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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 software displays an error message in the Simulation Diagnostics Viewer.

`Inherit: Inherit from 'Constant value'`
: Use data type of **Constant value** parameter. This option appears for some blocks.

`Inherit: Inherit via back propagation`
: Use data type of the driving block.

`Inherit: Same as input`
: Use data type of sole input signal. This option appears for some blocks.

`Inherit: Same as first input`
: Use data type of first input signal. This option appears for some blocks.

`Inherit: Same as accumulator`
: Output data type is the same as accumulator data type. This option appears for some blocks.

`double`
: Output data type is `double`.

`single`
: Output data type is `single`.

`int8`
: Output data type is `int8`.

`uint8`
: Output data type is `uint8`.

`int16`
: Output data type is `int16`.

`uint16`
: Output data type is `uint16`.

`int32`
: Output data type is `int32`.

`uint32`
: Output data type is `unt32`.

`fixdt(1,16,0)`
: Output data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Output data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
: Use an enumerated data type, for example, `Enum: BasicColors`. This option appears for some blocks.

`<data type expression>`
: Use a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<class name>`

`Expression`
: Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

See “Specifying a Fixed-Point Data Type” in the *Simulink User’s Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Characteristics

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

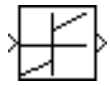
Purpose

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 block implementation is

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

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

The `Offset` corresponds to Coulomb friction. By default, the `Offset` is a vector with four elements: `[1 3 2 0]`. This default vector tests the same input value against four different offset values. You can specify a vector with a different number of elements, such as one.

The `Gain` corresponds to the signal gain for nonzero input values. By default, the `Gain` is 1.

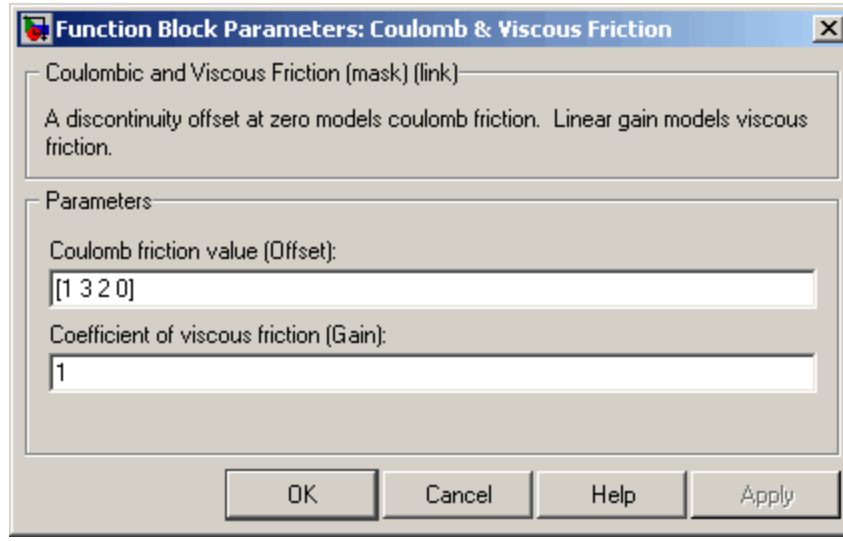
The block accepts one input and generates one output. The input can be a scalar, vector, or matrix. For a vector or matrix input, the `Offset` and `Gain` must have the same dimensions as the input or be scalars. For a scalar input, the output will be a scalar, vector, or matrix based on the dimensions of the `Offset` and `Gain`. For example, passing a scalar input to the block when using the default `Offset` produces an output vector with four elements.

Data Type Support

The Coulomb and Viscous Friction block accepts and outputs real signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point

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 for nonzero input values. The default is `1`.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	Yes, at the point where Coulomb (static) friction is overcome

Purpose

Count up and overflow back to zero after maximum value possible is reached for 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.

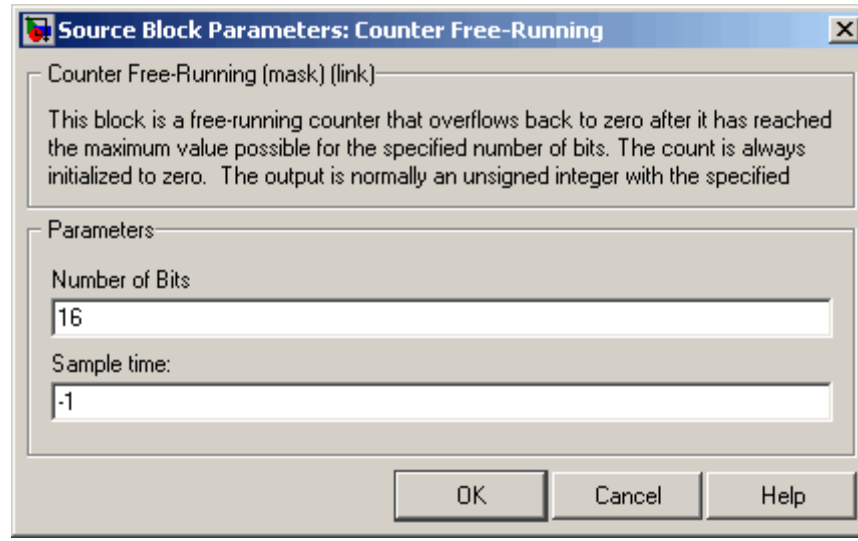
You can specify the sample time with the **Sample time** parameter.

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 outputs an unsigned integer.

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 "How to Specify the Sample Time" in the "How Simulink Works" chapter of the Simulink documentation.

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 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.

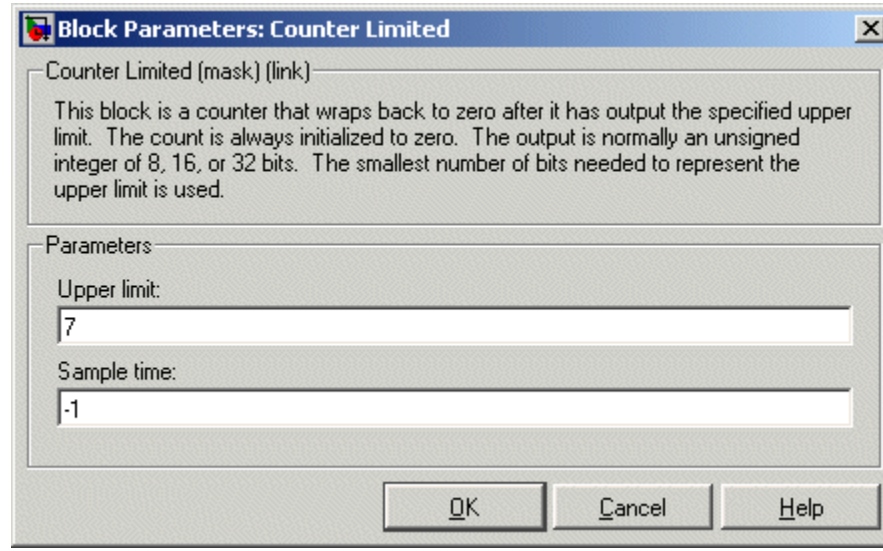
You can specify the sample time with the **Sample time** parameter. A **Sample time** of `-1` means that the sample time is inherited.

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 outputs an unsigned integer.

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 "How to Specify the Sample Time" in the "How Simulink Works" chapter of the Simulink documentation.

Characteristics

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

See Also

Counter Free-Running

Purpose

Define 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 that specify the same data store name.

The location of the Data Store Memory block that defines a data store determines which Data Store Read and Data Store Write blocks 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.

A Data Store Memory block in a referenced model cannot be accessed by Data Store Read or Data Store Write blocks in either a superior model or a subordinate model.

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.

Obtaining correct results from data stores requires ensuring that data store reads and writes occur in the expected order. See “Ordering Data Store Access” and “Using Data Store Diagnostics” for details.

You can use `Simulink.Signal` objects in addition to or instead of Data Store Memory blocks to define data stores. A data store defined in the base workspace using a signal object is accessible to every model,

including all referenced models. See “Working with Data Stores” for more information.

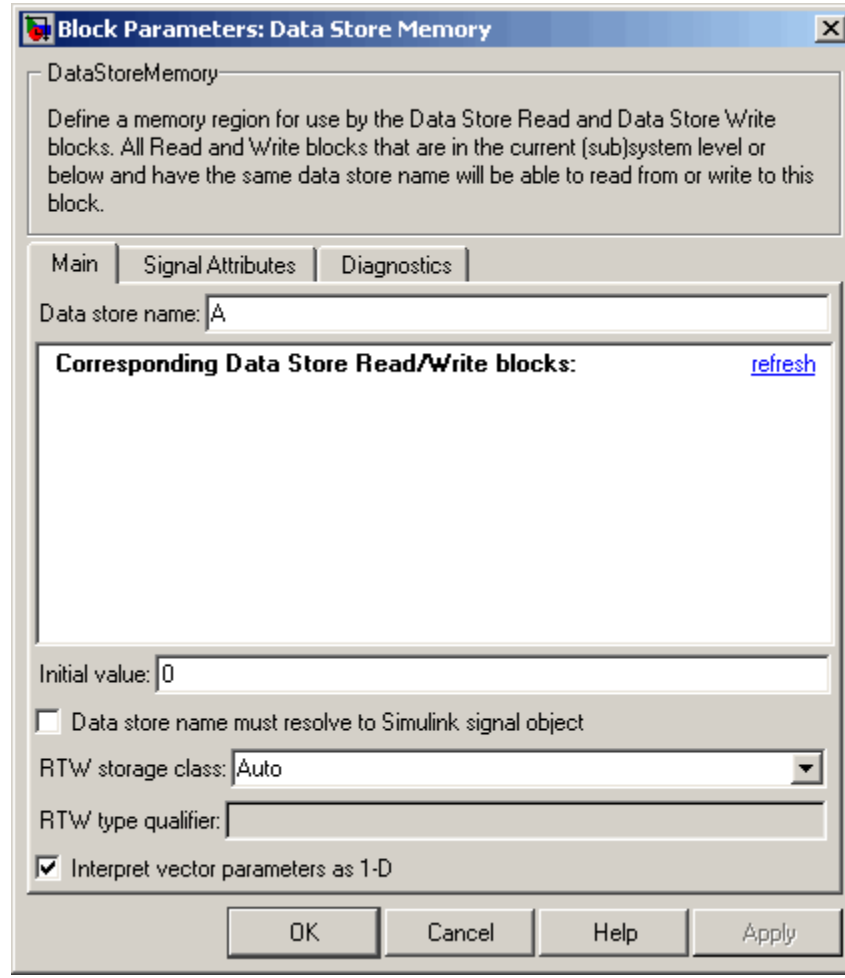
Data Type Support

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

For a discussion on the data types supported by Simulink software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

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. The **Minimum** parameter specifies the minimum value for this parameter, the **Maximum** parameter specifies the maximum value.

Data store must resolve to Simulink signal object
Causes Simulink software, when compiling the model, to search the model and base workspace for a `Simulink.Signal` object having the same name, as described in "Resolving Symbols". If such an object is not found, Simulink software halts the compilation and displays an error. Otherwise Simulink software 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 software halts model compilation and displays an error.

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

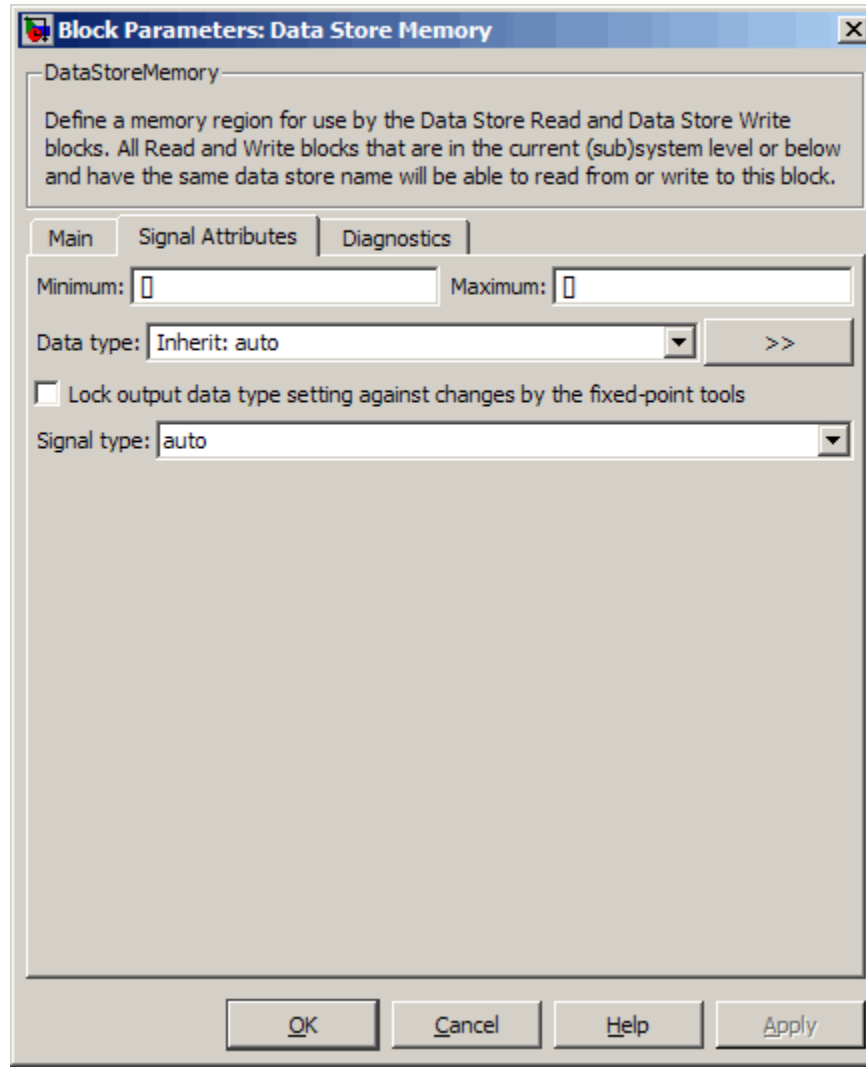
- **Data store name must resolve to Simulink signal object**
- **RTW storage class**
- **RTW type qualifier**

See “Block State Storage and Interfacing Considerations” 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. See “Determining the Output Dimensions of Source Blocks” in the “Working with Signals” chapter of the Simulink documentation.

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


Block Parameters: Data Store Memory
DataStoreMemory
Define a memory region for use by the Data Store Read and Data Store Write blocks. All Read and Write blocks that are in the current (sub)system level or below and have the same data store name will be able to read from or write to this block.
Main
Signal Attributes
Diagnostics
Minimum:
Maximum:
Data type:
Inherit: auto
>>
Lock output data type setting against changes by the fixed-point tools
Signal type:
auto
OK
Cancel
Help
Apply

Minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: auto`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

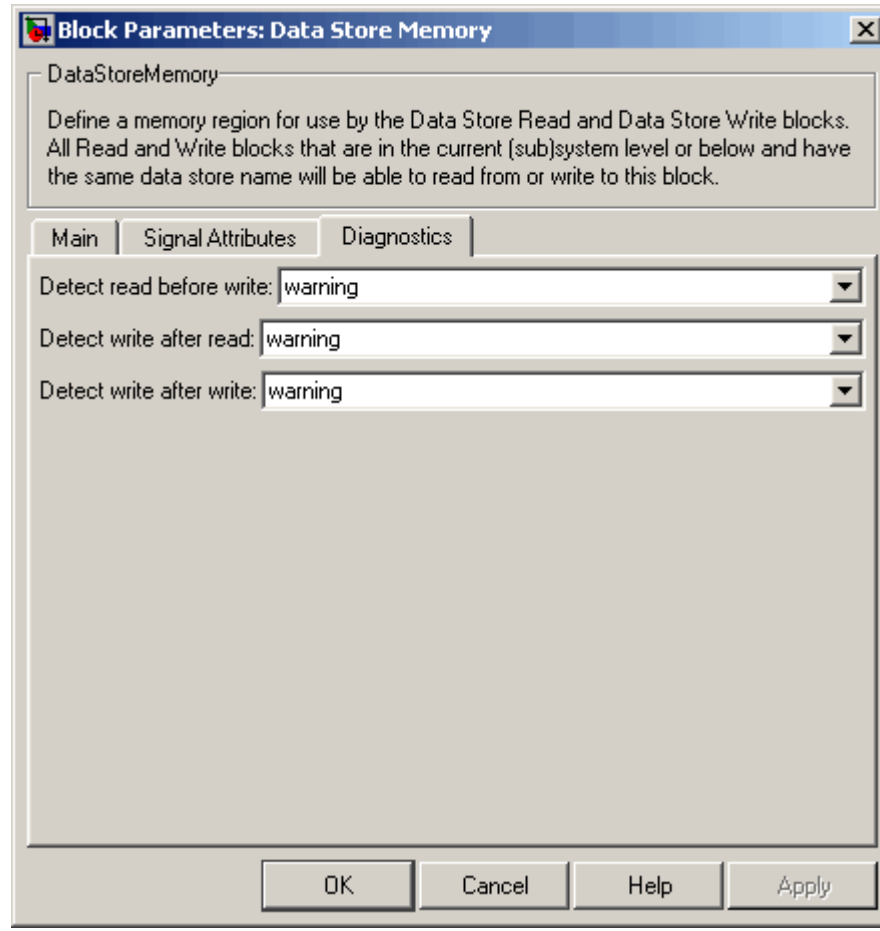
Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

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

Select the diagnostic action to take if the model attempts to read data from a data store to which it has not written data in this time step. See also "Detect read before write" in the **Data Store Memory Block** section of the **Configuration Parameters > Diagnostics > Data Validity** pane.

Default: `warning`

`none`
Simulink software takes no action.

`warning`
Simulink software displays a warning.

`error`
Simulink software terminates the simulation and displays an error message.

Detect write after read
Select the diagnostic action to take if the model attempts to write data to the data store after previously reading data from it in the current time step. See also "Detect write after read" in the **Data Store Memory Block** section of the **Configuration Parameters > Diagnostics > Data Validity** pane.

Default: `warning`

`none`
Simulink software takes no action.

`warning`
Simulink software displays a warning.

`error`
Simulink software terminates the simulation and displays an error message.

Detect write after write
Select the diagnostic action to take if the model attempts to write data to the data store twice in succession in the current time step. See also "Detect write after write" in the **Data Store Memory Block** section of the **Configuration Parameters > Diagnostics > Data Validity** pane.

Default: `warning`

`none`
: Simulink software takes no action.

`warning`
: Simulink software displays a warning.

`error`
: Simulink software terminates the simulation and displays an error message.

Characteristics

Sample Time	N/A
Dimensionalized	Yes
Multidimensionalized	Yes

See Also

- "Working with Data Stores"
- Data Store Memory
- Data Store Read

Data Store Read

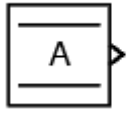
Purpose

Read data from data store

Library

Signal Routing

Description



The Data Store Read block copies data from the named data store to its output. More than one Data Store Read block can read from the same data store.

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.

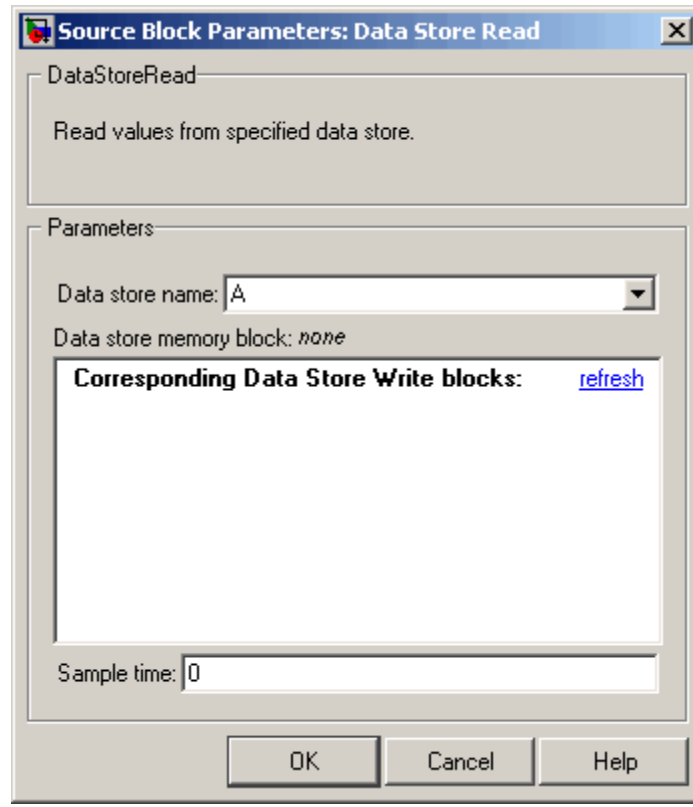
Obtaining correct results from data stores requires ensuring that data store reads and writes occur in the expected order. See "Ordering Data Store Access" and "Using Data Store Diagnostics" for details.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

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 software compiles the model containing this block, Simulink software searches the model upwards from this block's level for a Data Store Memory block having the specified

data store name. If Simulink software does not find such a block, it searches the model workspace and the MATLAB workspace for a `Simulink.Signal` object having the same name. See "Resolving Symbols" for more information about the search path.

If Simulink software 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 software 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 "How to Specify the Sample Time" for more information.

Characteristics

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

See Also

- "Working with Data Stores"
- Data Store Memory
- Data Store Read

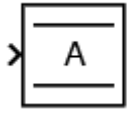
Purpose

Write data to 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 block 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.

Obtaining correct results from data stores requires ensuring that data store reads and writes occur in the expected order. See "Ordering Data Store Access" and "Using Data Store Diagnostics" for details.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

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 software compiles the model containing this block, Simulink software searches the model upwards from this block's level for a Data Store Memory block having the specified data store name. If Simulink software does not find such a block, it searches the model workspace and the MATLAB workspace for a `Simulink.Signal` object having the same name. See "Resolving Symbols" for more information about the search path.

If Simulink software 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 software 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 "How to Specify the Sample Time" for more information.

Characteristics

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

See Also

- "Working with Data Stores"
- Data Store Memory
- Data Store Read

Data Type Conversion

Purpose

Convert input signal to specified data type

Library

Signal Attributes

Description

The Data Type Conversion block converts an input signal of any Simulink software data type to the data type and scaling you specify for the **Output data type** parameter. 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`.

Suppose that 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 **Integer rounding mode** parameter to `Floor` and clear the **Saturate on integer overflow** check box.

Working with Enumerated Signals

You can use a Data Type Conversion block to cast signal of an enumerated type to a signal of any numeric type, provided that the underlying integers of all enumerated values input to the block are within the range of the numeric type. Otherwise, an error occurs during simulation.

You can use a Data Type Conversion block to cast a signal of any integer type to a signal of an enumerated type, provided that every value input to the Data Type Conversion block is the underlying integer of some value in the enumerated type. Otherwise, an error occurs during simulation.

You cannot use a Data Type Conversion block to cast a non-integer numeric signal an enumerated signal. You cannot cast a complex signal

to an enumerated signal regardless of the data types of its real and imaginary parts. See “Using Enumerated Data” for information about enumerated data types in Simulink.

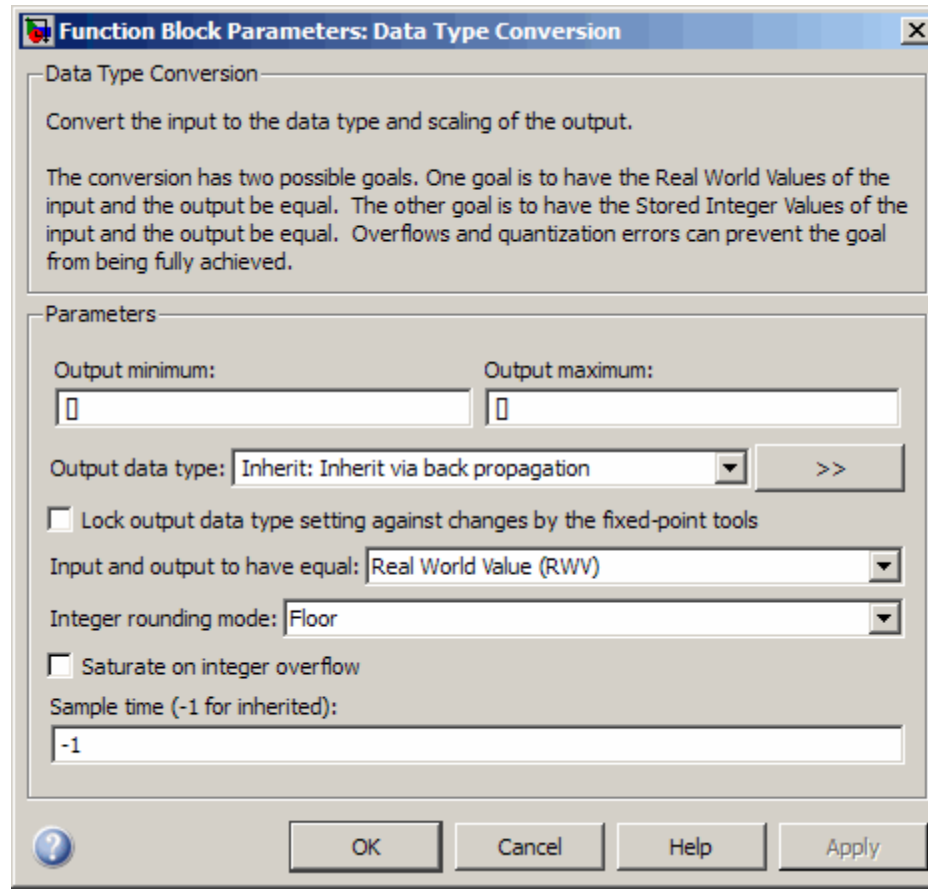
When you generate code for a Data Type Conversion block that casts to an enumerated type, the code uses safe casting if the block’s **Saturate on integer overflow** option is selected. If the option is cleared, the code does not use safe casting to enumerated types. See “Enumerated Type Safe Casting” for more information.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

Parameters and Dialog Box



Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Input and output to have equal

Specify which type of input and output should be equal.

Settings

Default: `Real World Value (RWV)`

`Real World Value (RWV)`
: Specifies the goal of making the `Real World Value (RWV)` of the input equal to the `Real World Value (RWV)` of the output.

`Stored Integer (SI)`
: Specifies the goal of making the `Stored Integer (SI)` value of the input equal to the `Stored Integer (SI)` value of the output.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Floor`

`Ceiling`
: Round both positive and negative numbers toward positive infinity.

`Convergent`
: Round number to the nearest representable value. If a tie occurs, round to the nearest even stored value.

`Floor`
: Round both positive and negative numbers toward negative infinity.

`Nearest`
: Round number to the nearest representable value. If a tie occurs, round toward positive infinity.

`Round`
: Round number to the nearest representable value. If a tie occurs, round positive numbers toward positive infinity and round negative numbers toward negative infinity.

`Simplest`
: This option provides for an optimization of the rounding code for several blocks.

`Zero`
: Round number toward zero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Rounding” in the *Simulink Fixed Point User’s Guide*.

Saturate on integer overflow

Specify whether overflows saturate.

Settings

Default: Off

 On

Overflows saturate.

 Off

Overflows do not saturate.

Tips

- When you select this check box, saturation applies to every internal operation on the block, not just the output or result.
- In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits or specify another appropriate sample time such as continuous or inherited.

Settings

Default: -1

By default, the block inherits its sample time based upon the context of the block within the model. To set a different sample time, enter a valid sample time based upon the table in "Types of Sample Time".

See also "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `Inherit: Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block), `Inherit: Inherit from 'Constant value'` (Constant block), `Inherit: Inherit via back propagation` (Data Type Conversion block), `Inherit: Same as input` (Saturation block)

`Inherit: Inherit via internal rule`
: This option appears for some blocks. Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the calculated 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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 software displays an error message in the Simulation Diagnostics Viewer.

`Inherit: Inherit from 'Constant value'`
: Use data type of **Constant value** parameter. This option appears for some blocks.

`Inherit: Inherit via back propagation`
: Use data type of the driving block.

`Inherit: Same as input`
: Use data type of sole input signal. This option appears for some blocks.

`Inherit: Same as first input`
: Use data type of first input signal. This option appears for some blocks.

`Inherit: Same as accumulator`
: Output data type is the same as accumulator data type. This option appears for some blocks.

`double`
: Output data type is `double`.

`single`
: Output data type is `single`.

`int8`
: Output data type is `int8`.

`uint8`
: Output data type is `uint8`.

`int16`
: Output data type is `int16`.

`uint16`
: Output data type is `uint16`.

`int32`
: Output data type is `int32`.

`uint32`
: Output data type is `unt32`.

`fixdt(1,16,0)`
: Output data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Output data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
: Use an enumerated data type, for example, `Enum: BasicColors`. This option appears for some blocks.

`<data type expression>`
: Use a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
Fixed-point data types.

`Enumerated`
Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `<class name>`

`Expression`
Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

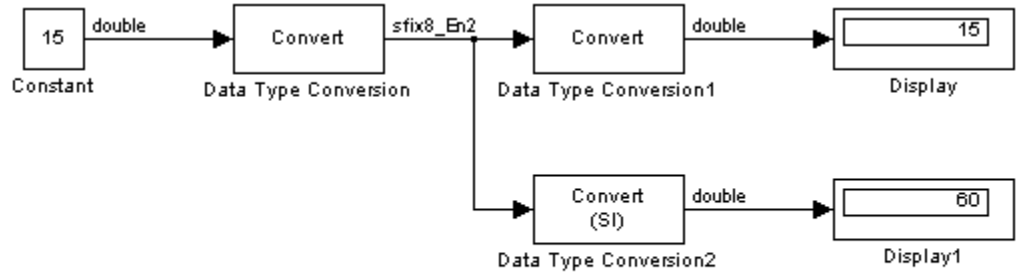
See Also

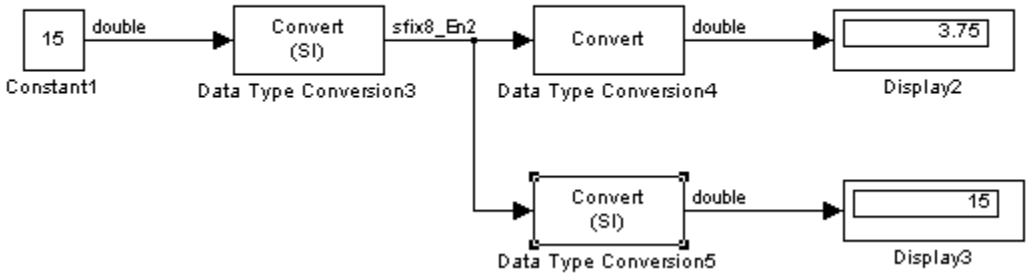
See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Examples

Example 1 — Real World Values Versus Stored Integers

This example uses the Data Type Conversion block to explain the difference between a real-world value and a stored integer. Consider the two fixed-point models shown below.





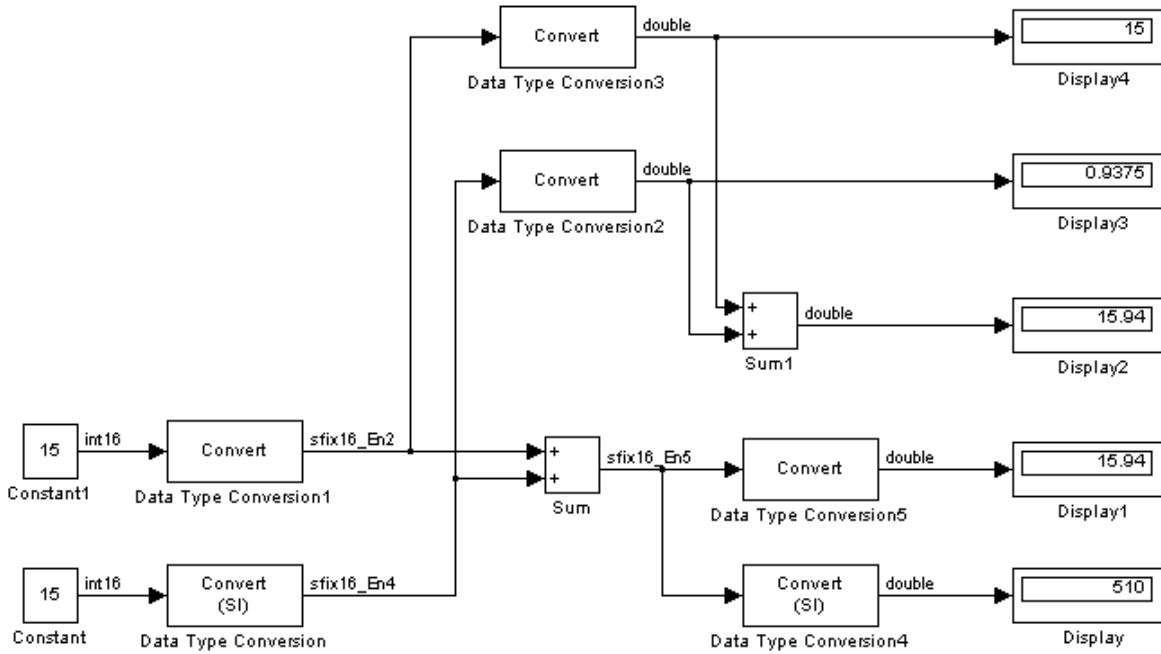
In the top model, the Data Type Conversion block treats the input as a real-world value, and maps that value to an 8-bit signed generalized fixed-point data type with a scaling of 2^{-2}. When the value is then output from the Data Type Conversion1 block as a real-world value, the scaling and data type information is retained and the output value is 001111.00, or 15. When the value is output from the Data Type Conversion2 block as a stored integer, the scaling and data type information is not retained and the stored integer is interpreted as 00111100, or 60.

In the bottom model, the Data Type Conversion3 block treats the input as a stored integer, and the data type and scaling information is not applied. When the value is then output from the Data Type Conversion4 block as a real-world value, the scaling and data type information is applied to the stored integer, and the output value is 000011.11, or 3.75.

When the value is output from the Data Type Conversion5 block as a stored integer, you get back the original input value of 15.

Example 2 — Real World Values and Stored Integers in Summations

The model shown below illustrates how a summation operation applies to real-world values and stored integers, and how scaling information is dealt with in generated code.



Note that the summation operation produces the correct result when the Data Type Conversion (2 or 5) block outputs a real-world value. This is because the specified scaling information is applied to the stored integer value. However, when the Data Type Conversion4 block outputs a stored integer value, then the summation operation produces an unexpected result due to the absence of scaling information.

If you generate code for the above model, then the code captures the appropriate scaling information. The code for the Sum block is shown below. The inputs to this block are tagged with the specified scaling information so that the necessary shifts are performed for the summation operation.

```
/* Sum Block: <Root>/Sum
   *
   *  y =  u0 + u1
   *
   * Input0  Data Type:  Fixed Point    S16  2^-2
   * Input1  Data Type:  Fixed Point    S16  2^-4
   * Output0 Data Type:  Fixed Point    S16  2^-5
   *
   * Round Mode: Floor
   * Saturation Mode: Wrap
   *
   */
  sum = ((in1) << 3);
  sum += ((in2) << 1);
```

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	N/A
Dimensionalized	Yes
Multidimensionalized	Yes
Zero-Crossing Detection	No

See Also

Data Type Conversion Inherited

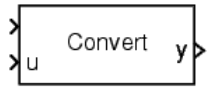
Purpose

Convert from one data type to another using inherited data type and scaling

Library

Signal Attributes

Description



The Data Type Conversion Inherited block forces dissimilar data types to be the same. The first input is used as the reference signal and the second input is converted to the reference type by inheriting the data type and scaling information. (See "How to Rotate a Block" in the Simulink User's Guide for a description of the port order for various block orientations.) Either input is scalar expanded such that the output has the same width as the widest input.

Inheriting the data type and scaling provides these advantages:

- It makes reusing existing models easier.
- It allows you to create new fixed-point models with less effort since you can avoid the detail of specifying the associated parameters.

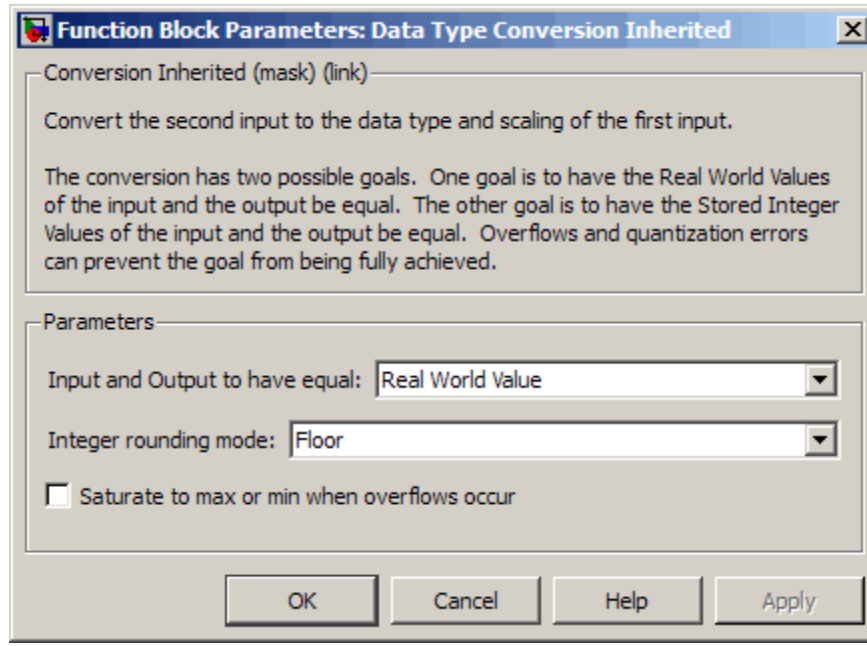
Data Type Support

The Data Type Conversion Inherited block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

For more information, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Input and Output to have equal

Specify whether the `Real World Value (RWV)` or the `Stored Integer (SI)` of the input and output should be the same. Refer to Description in the Data Type Conversion block reference page for more information about these choices.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate to max or min when overflows occur

Select to have overflows saturate.

Characteristics

Direct Feedthrough	Yes

See Also Data Type Conversion

Data Type Duplicate

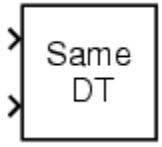
Purpose

Force all inputs to same data type

Library

Signal Attributes

Description



The Data Type Duplicate block forces all inputs to have exactly the same data type. Other attributes of input signals, such as dimension, complexity, and sample time, are completely independent.

You can use the Data Type Duplicate block to check for consistency of data types among blocks. If all signals do not have the same data type, the block returns an error message.

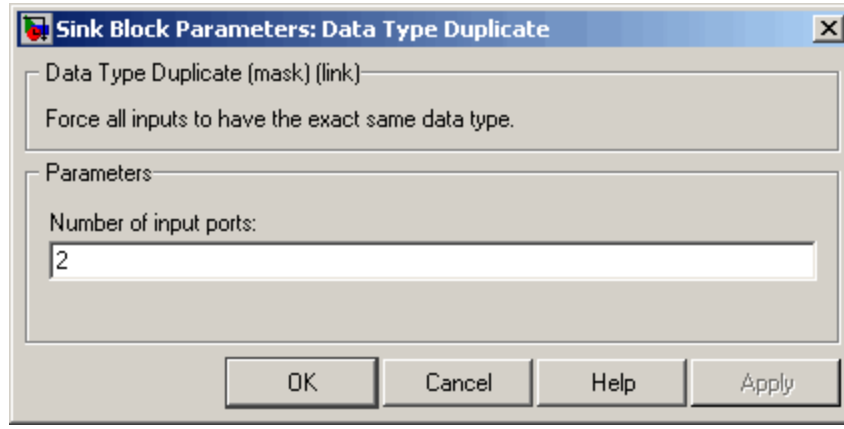
The Data Type Duplicate block is typically used such that one signal to the block controls the data type for all other blocks. The other blocks are set to inherit their data types via back propagation.

The block is also used in a user created library. These library blocks can be placed in any model, and the data type for all library blocks are configured according to the usage in the model. To create a library block with more complex data type rules than duplication, use the Data Type Propagation block.

Data Type Support

Except for enumerated data types, the Data Type Duplicate block accepts signals of any data type supported by Simulink, including fixed-point types.

Parameters and Dialog Box



Number of input ports

Number of input ports.

Characteristics

Scalar Expansion	Yes
States	0

Data Type Propagation

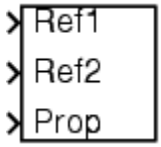
Purpose

Set data type and scaling of propagated signal based on information from reference signals

Library

Signal Attributes

Description



The Data Type Propagation block allows you to control the data type and scaling of signals in your model. You can use this block in conjunction with fixed-point blocks that have their **Output data type** parameter configured to `Inherit: Inherit via back propagation`.

The block has three inputs: Ref1 and Ref2 are the reference inputs, while the Prop input back propagates the data type and scaling information gathered from the reference inputs. This information is then passed on to other fixed-point blocks.

The block provides you with many choices for propagating data type and scaling information. For example, you can:

- Use the number of bits from the Ref1 reference signal, or use the number of bits from widest reference signal.
- Use the range from the Ref2 reference signal, or use the range of the reference signal with the greatest range.
- Use a bias of zero, regardless of the biases used by the reference signals.
- Use the precision of the reference signal with the least precision.

You specify how data type information is propagated with the **Propagated data type** parameter list. If the parameter list is configured as `Specify via dialog`, then you manually specify the data type via the **Propagated data type** edit field. If the parameter list is configured as `Inherit via propagation rule`, then you must use the parameters described in "Parameters and Dialog Box" on page 2-213.

You specify how scaling information is propagated with the **Propagated scaling** parameter list. If the parameter list is configured as `Specify via dialog`, then you manually specify the scaling via the **Propagated scaling** edit field. If the parameter list is configured as `Inherit via`

`propagation rule`, then you must use the parameters described in "Parameters and Dialog Box" on page 2-213.

After you use the information from the reference signals, you can apply a second level of adjustments to the data type and scaling by using individual multiplicative and additive adjustments. This flexibility has a variety of uses. For example, if you are targeting a DSP, then you can configure the block so that the number of bits associated with a MAC (multiply and accumulate) operation is twice as wide as the input signal, and has a certain number of guard bits added to it.

The Data Type Propagation block also provides a mechanism to force the computed number of bits to a useful value. For example, if you are targeting a 16-bit micro, then the target C compiler is likely to support sizes of only 8 bits, 16 bits, and 32 bits. The block will force these three choices to be used. For example, suppose the block computes a data type size of 24 bits. Since 24 bits is not directly usable by the target chip, the signal is forced up to 32 bits, which is natively supported.

There is also a method for dealing with floating-point reference signals. This makes it easier to create designs that are easily retargeted from fixed-point chips to floating-point chips or vice versa.

The Data Type Propagation block allows you to set up libraries of useful subsystems that will be properly configured based on the connected signals. Without this data type propagation process, a subsystem that you use from a library will almost certainly not work as desired with most integer or fixed-point signals, and manual intervention to configure the data type and scaling would be required. This block can eliminate the manual intervention in many situations.

Precedence Rules

The precedence of the dialog box parameters decreases from top to bottom. Additionally:

- Double-precision reference inputs have precedence over all other data types.

- Single-precision reference inputs have precedence over integer and fixed-point data types.
- Multiplicative adjustments are carried out before additive adjustments.
- The number of bits is determined before the precision or positive range is inherited from the reference inputs.

Data Type Support

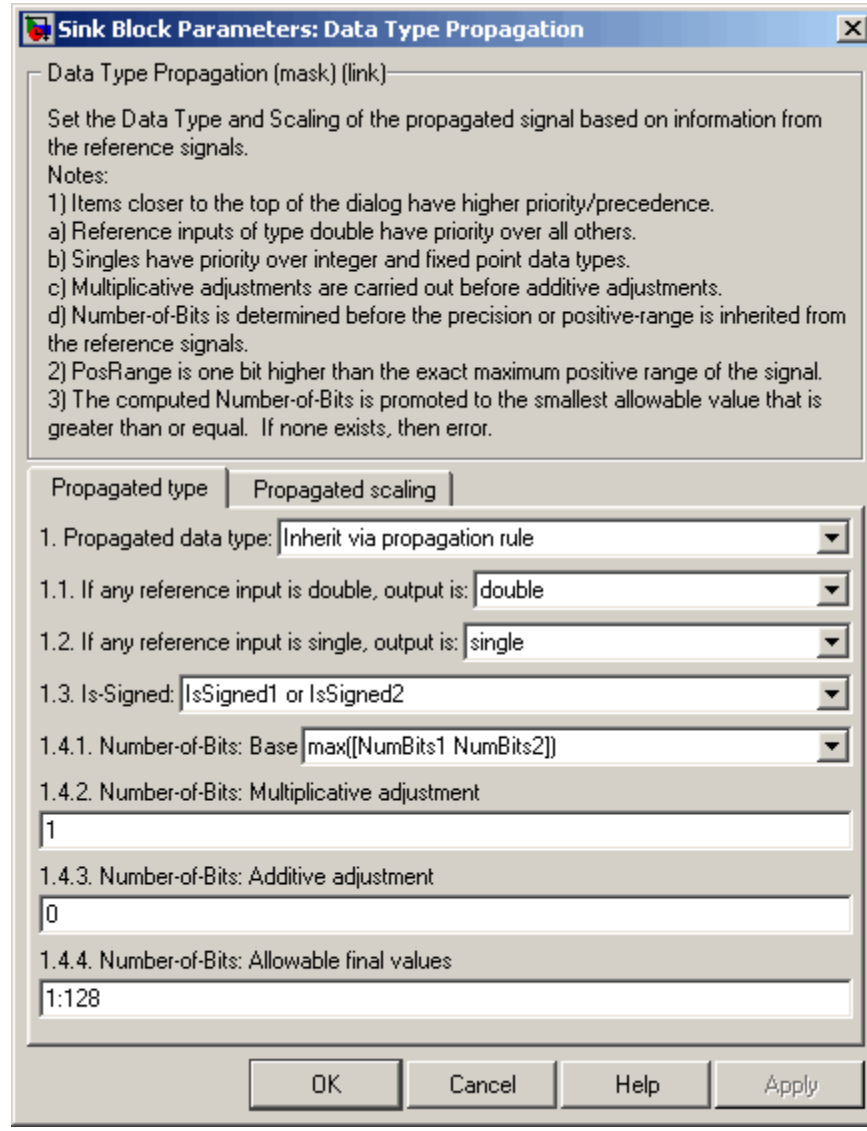
The Data Type Propagation block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

For more information, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box

The **Propagated type** pane of the Data Type Propagation block dialog appears as follows:



Propagated data type

Use the parameter list to propagate the data type via the dialog box, or inherit the data type from the reference signals. Use the edit field to specify the data type via the dialog box.

If any reference input is double, output is

Specify `single` or `double`. This parameter makes it easier to create designs that are easily retargeted from fixed-point chips to floating-point chips or vice versa.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated data type** parameter list.

If any reference input is single, output is

Specify `single` or `double`. This parameter makes it easier to create designs that are easily retargeted from fixed-point chips to floating-point chips or visa versa.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated data type** parameter list.

Is-Signed

Specify the sign of Prop as one of the following values:

Parameter Value	Description
`IsSigned1`	Prop is a signed data type if Ref1 is a signed data type.
`IsSigned2`	Prop is a signed data type if Ref2 is a signed data type.
`IsSigned1 or IsSigned2`	Prop is a signed data type if either Ref1 or Ref2 are signed data types.
`TRUE`	Ref1 and Ref2 are ignored, and Prop is always a signed data type.
`FALSE`	Ref1 and Ref2 are ignored, and Prop is always an unsigned data type.

For example, if the Ref1 signal is `ufix(16)`, the Ref2 signal is `sfix(16)`, and the **Is-Signed** parameter is `IsSigned1 or IsSigned2`, then Prop is forced to be a signed data type.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated data type** parameter list.

Number-of-bits: Base

Specify the number of bits used by Prop for the base data type as one of the following values:

Parameter Value	Description
`NumBits1`	The number of bits for Prop is given by the number of bits for Ref1.
`NumBits2`	The number of bits for Prop is given by the number of bits for Ref2.
`max([NumBits1 NumBits2])`	The number of bits for Prop is given by the reference signal with largest number of bits.
`min([NumBits1 NumBits2])`	The number of bits for Prop is given by the reference signal with smallest number of bits.
`NumBits1+NumBits2`	The number of bits for Prop is given by the sum of the reference signal bits.

Refer to Targeting an Embedded Processor in the *Simulink Fixed Point User's Guide* for more information about the base data type.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated data type** parameter list.

Number-of-bits: Multiplicative adjustment

Specify the number of bits used by Prop by including a multiplicative adjustment. For example, suppose you want to guarantee that the number of bits associated with a multiply and

accumulate (MAC) operation is twice as wide as the input signal. To do this, you configure this parameter to the value `2`.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated data type** parameter list.

Number-of-bits: Additive adjustment

Specify the number of bits used by Prop by including an additive adjustment. For example, if you are performing multiple additions during a MAC operation, the result might overflow. To prevent overflow, you can associate guard bits with the propagated data type. To associate four guard bits, you specify the value `4`.

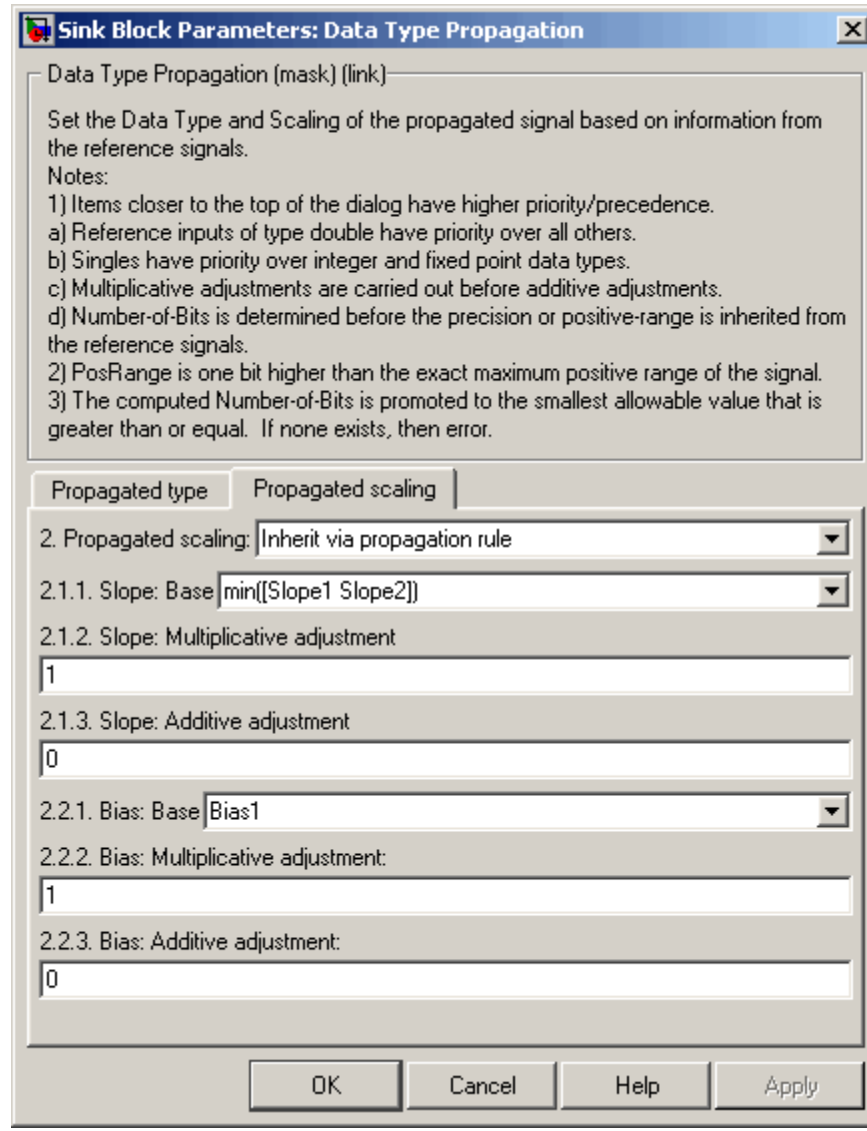
This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated data type** parameter list.

Number-of-bits: Allowable final values

Force the computed number of bits used by Prop to a useful value. For example, if you are targeting a processor that supports only 8, 16, and 32 bits, then you configure this parameter to `[8,16,32]`. The block always propagates the smallest specified value that fits. If you want to allow all fixed-point data types, you would specify the value `1:128`.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated data type** parameter list.

The **Propagated scaling** pane of the Data Type Propagation block dialog appears as follows:


Sink Block Parameters: Data Type Propagation
Data Type Propagation (mask) (link)
Set the Data Type and Scaling of the propagated signal based on information from the reference signals.
Notes:
1) Items closer to the top of the dialog have higher priority/precedence.
a) Reference inputs of type double have priority over all others.
b) Singles have priority over integer and fixed point data types.
c) Multiplicative adjustments are carried out before additive adjustments.
d) Number-of-Bits is determined before the precision or positive-range is inherited from the reference signals.
2) PosRange is one bit higher than the exact maximum positive range of the signal.
3) The computed Number-of-Bits is promoted to the smallest allowable value that is greater than or equal. If none exists, then error.
Propagated type
Propagated scaling
2. Propagated scaling: Inherit via propagation rule
2.1.1. Slope: Base min([Slope1 Slope2])
2.1.2. Slope: Multiplicative adjustment
1
2.1.3. Slope: Additive adjustment
0
2.2.1. Bias: Base Bias1
2.2.2. Bias: Multiplicative adjustment:
1
2.2.3. Bias: Additive adjustment:
0
OK
Cancel
Help
Apply

Propagated scaling

Use the parameter list to propagate the scaling via the dialog box, inherit the scaling from the reference signals, or calculate the scaling to obtain best precision.

Propagated scaling (Slope or [Slope Bias])

Specify the scaling as either a slope or a slope and bias.

This parameter is visible only if `Specify via dialog` is selected for the **Propagated scaling** parameter list.

Values used to determine best precision scaling

Specify any values to be used to constrain the precision, such as the upper and lower limits on the propagated input. Based on the data type, the scaling will automatically be selected such that these values can be represented with no overflow error and minimum quantization error.

This parameter is visible only if `Obtain via best precision` is selected for the **Propagated scaling** parameter list.

Slope: Base

Specify the slope used by Prop for the base data type as one of the following values:

Parameter Value	Description
`Slope1`	The slope of Prop is given by the slope of Ref1.
`Slope2`	The slope of Prop is given by the slope of Ref2.
`max([Slope1 Slope2])`	The slope of Prop is given by the maximum slope of the reference signals.
`min([Slope1 Slope2])`	The slope of Prop is given by the minimum slope of the reference signals.

Parameter Value	Description
`Slope1*Slope2`	The slope of Prop is given by the product of the reference signal slopes.
`Slope1/Slope2`	The slope of Prop is given by the ratio of the Ref1 slope to the Ref2 slope.
`PosRange1`	The range of Prop is given by the range of Ref1.
`PosRange2`	The range of Prop is given by the range of Ref2.
`max([PosRange1 PosRange2])`	The range of Prop is given by the maximum range of the reference signals.
`min([PosRange1 PosRange2])`	The range of Prop is given by the minimum range of the reference signals.
`PosRange1*PosRange2`	The range of Prop is given by the product of the reference signal ranges.
`PosRange1/PosRange2`	The range of Prop is given by the ratio of the Ref1 range to the Ref2 range.

You control the precision of Prop with `Slope1` and `Slope2`, and you control the range of Prop with `PosRange1` and `PosRange2`. Additionally, `PosRange1` and `PosRange2` are one bit higher than the maximum positive range of the associated reference signal.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated scaling** parameter list.

Slope: Multiplicative adjustment

Specify the slope used by Prop by including a multiplicative adjustment. For example, if you want 3 bits of additional precision (with a corresponding decrease in range), the multiplicative adjustment is `2^-3`.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated scaling** parameter list.

Slope: Additive adjustment

Specify the slope used by Prop by including an additive adjustment. An additive slope adjustment is often not needed. The most likely use is to set the multiplicative adjustment to `0`, and set the additive adjustment to force the final slope to a specified value.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated scaling** parameter list.

Bias: Base

Specify the bias used by Prop for the base data type. The parameter values are described as follows:

Parameter Value	Description
`Bias1`	The bias of Prop is given by the bias of Ref1.
`Bias2`	The bias of Prop is given by the bias of Ref2.
`max([Bias1 Bias2])`	The bias of Prop is given by the maximum bias of the reference signals.
`min([Bias1 Bias2])`	The bias of Prop is given by the minimum bias of the reference signals.
`Bias1*Bias2`	The bias of Prop is given by the product of the reference signal biases.
`Bias1/Bias2`	The bias of Prop is given by the ratio of the Ref1 bias to the Ref2 bias.
`Bias1+Bias2`	The bias of Prop is given by the sum of the reference biases.
`Bias1-Bias2`	The bias of Prop is given by the difference of the reference biases.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated scaling** parameter list.

Bias: Multiplicative adjustment

Specify the bias used by Prop by including a multiplicative adjustment.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated scaling** parameter list.

Bias: Additive adjustment

Specify the bias used by Prop by including an additive adjustment.

If you want to guarantee that the bias associated with Prop is zero, you should configure both the multiplicative adjustment and the additive adjustment to `0`.

This parameter is visible only if `Inherit via propagation rule` is selected for the **Propagated scaling** parameter list.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

Data Type Scaling Strip

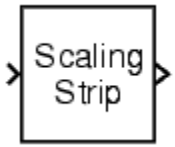
Purpose

Remove scaling and map to built in integer

Library

Signal Attributes

Description

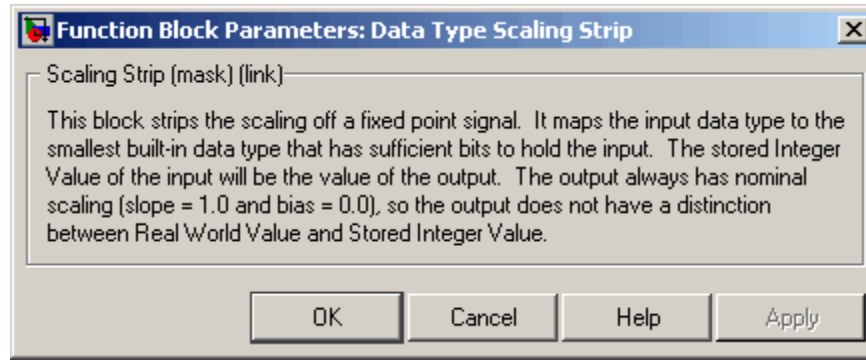


The Scaling Strip block strips the scaling off a fixed point signal. It maps the input data type to the smallest built in data type that has enough data bits to hold the input. The stored integer value of the input is the value of the output. The output always has nominal scaling (slope = 1.0 and bias = 0.0), so the output does not make a distinction between real world value and stored integer value.

Data Type Support

The Data Type Scaling Strip block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes
Multidimensionalized	Yes

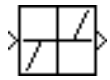
Purpose

Provide region of zero output

Library

Discontinuities

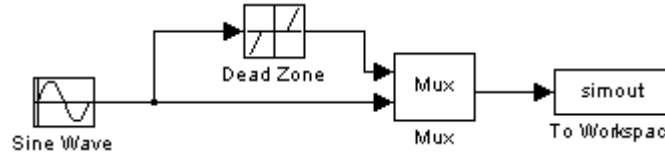
Description



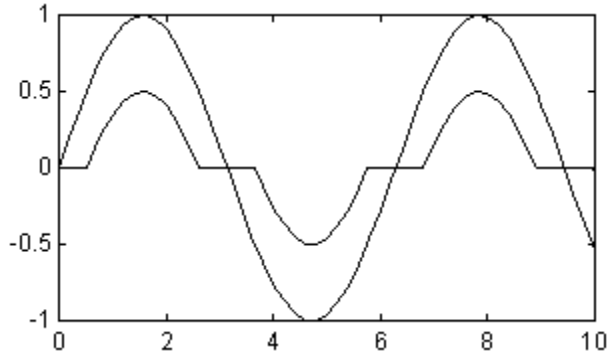
The Dead Zone block generates zero output within a specified region, called its dead zone. The lower and upper limits of the dead zone are specified as the **Start of dead zone** and **End of dead zone** parameters. The block output depends on the input and dead zone:

- If the input is within the dead zone (greater than the lower limit and less than the upper limit), the output is zero.
- If the input is greater than or equal to the upper limit, the output is the input minus the upper limit.
- If the input is less than or equal to the lower limit, the output is the input minus the lower limit.

This sample model uses lower and upper limits of -0.5 and +0.5, with a sine wave as input.



This plot shows the effect of the Dead Zone block on the sine wave. While the input (the sine wave) is between -0.5 and 0.5, the output is zero.

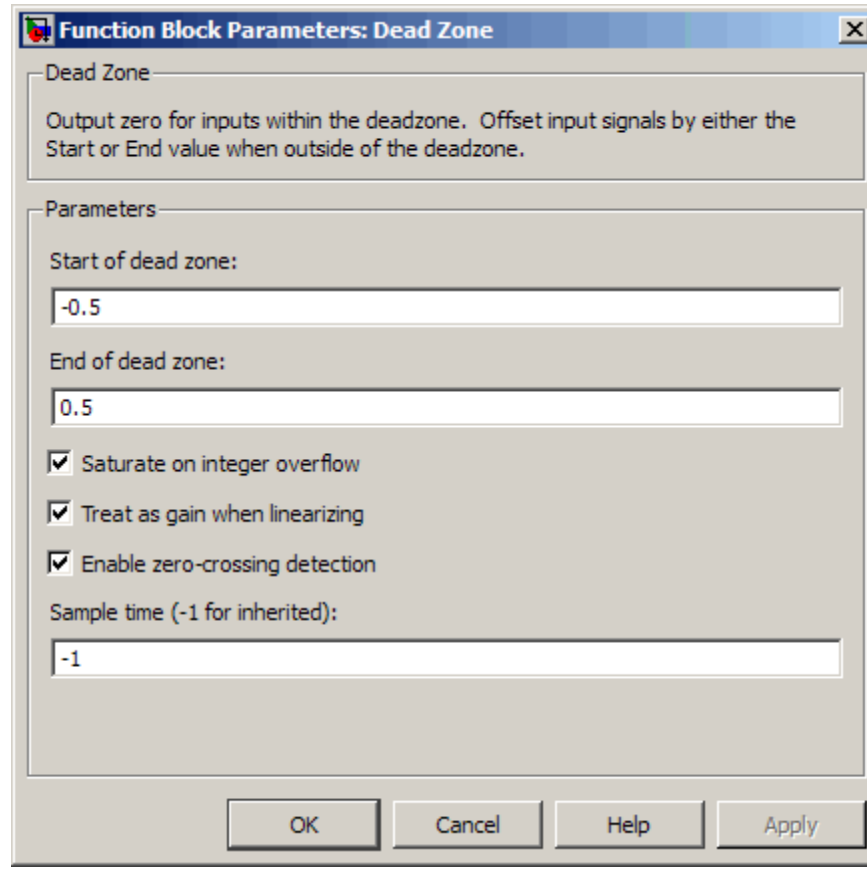


Data Type Support

The Dead Zone block accepts and outputs a real signal of any numeric data type supported by Simulink software, including fixed-point data types.

For more information, see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

Parameters and Dialog Box



Start of dead zone
Specify the lower limit of the dead zone. The default is `-0.5`.

End of dead zone
Specify the upper limit of the dead zone. The default is `0.5`.

Saturate on integer overflow
Select to have overflows saturate.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is generated.

Treat as gain when linearizing

The linearization commands in Simulink software treat this block as a gain in state space. Select this option to cause the 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 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 "How Simulink Works" chapter of the Simulink documentation.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled

See Also

Dead Zone Dynamic

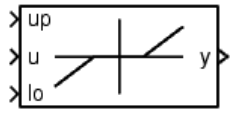
Purpose

Set inputs within bounds to zero

Library

Discontinuities

Description



The Dead Zone Dynamic block dynamically bounds the range of the input signal, providing a region of zero output. The bounds change according to the upper and lower limit input signals where

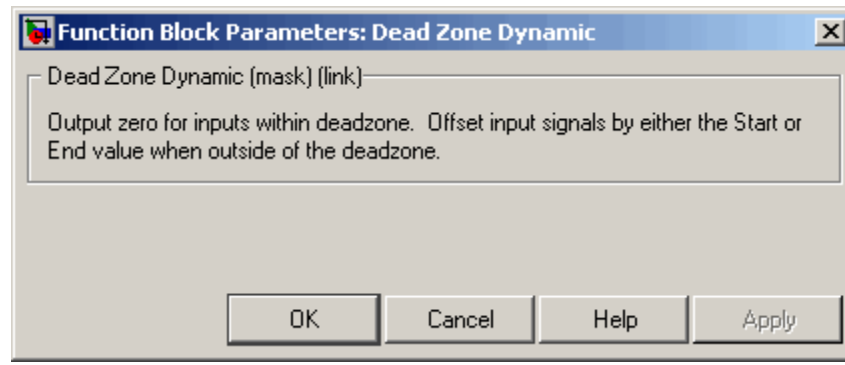
- The input within the bounds is set to zero.
- The input below the lower limit is shifted down by the lower limit.
- The input above the upper limit is shifted down by the upper limit.

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 Dead Zone Dynamic block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also Dead Zone

Purpose

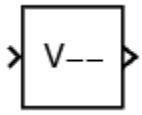
Decrease real world value of signal by one

Library

Additional Math & Discrete / Additional Math: Increment - Decrement

Description

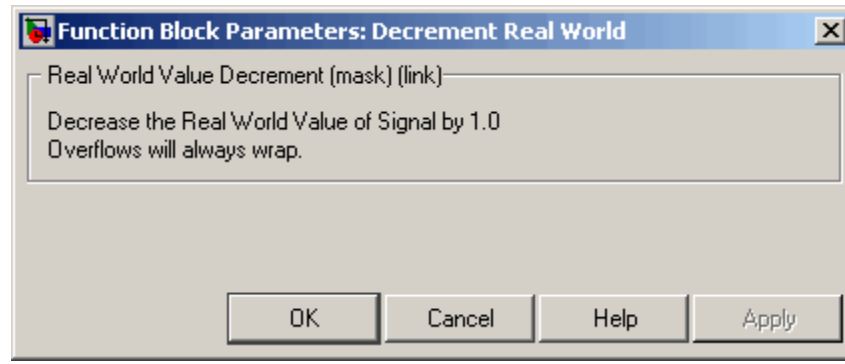
The Decrement Real World block decreases the real world value of the signal by one. Overflows always wrap.



Data Type Support

The Decrement Real World block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	No

See Also

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

Decrement Stored Integer

Purpose

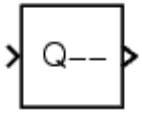
Decrease stored integer value of signal by one

Library

Additional Math & Discrete / Additional Math: Increment - Decrement

Description

The Decrement Stored Integer block decreases the stored integer value of a signal by one.

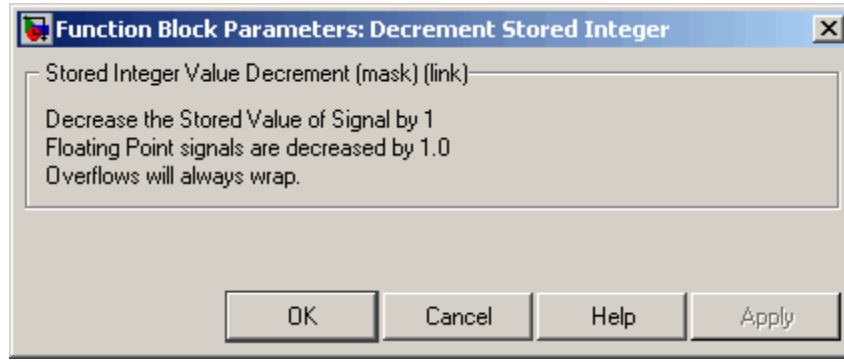


Floating-point signals are also decreased by one, and overflows always wrap.

Data Type Support

The Decrement Stored Integer block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	No

See Also

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

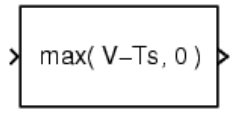
Purpose

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

Library

Additional Math & Discrete / Additional Math: Increment - Decrement

Description

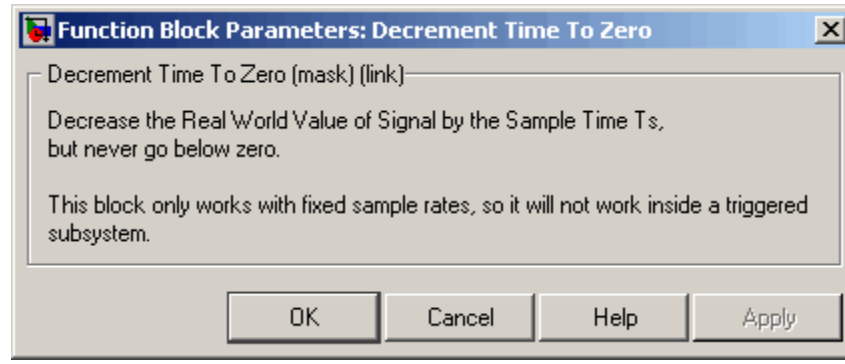


The Decrement Time To Zero block decreases the real-world value of the signal by the sample time, Ts. The output will never go below zero. This block only works with fixed sample rates.

Data Type Support

The Decrement Time To Zero block accepts signals of any numeric data type supported by Simulink software, 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

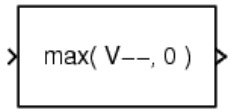
Decrease real-world value of signal by one, but only to zero

Library

Additional Math & Discrete / Additional Math: Increment - Decrement

Description

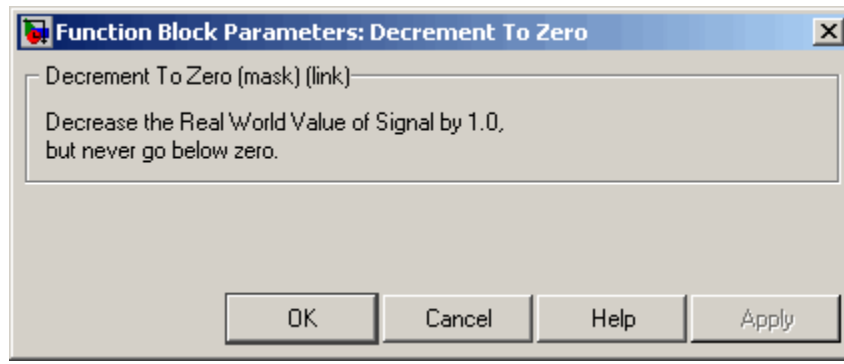
The Decrement To Zero block decreases the real-world value of the signal by one. The output will never go below zero.



Data Type Support

The Decrement To Zero block accepts signals of any numeric data type supported by Simulink software, 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 elements of vector signal

Library

Signal Routing

Description

The Demux block extracts the components of an input signal and outputs the components as separate signals. The output signals are ordered from top to bottom output port. See "How to Rotate a Block" for a description of the port order for various block orientations. To avoid adding clutter to a model, Simulink software hides the name of a Demux block when you copy it from the Simulink library to a model. See "Mux Signals" for information about creating and decomposing vectors.

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 mode 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 a bus signal.

Note The MathWorks discourages enabling **Bus selection mode** and using a Demux block to extract elements of a bus signal. Muxes and buses should not be intermixed in new models. See "Avoiding Mux/Bus Mixtures" for more information.

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

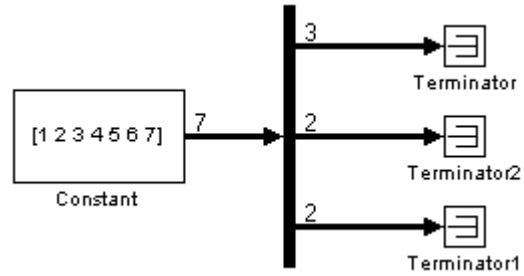
the block's output ports. The block determines the size of its outputs from the size of the input signal and the value of the **Number of outputs** parameter.

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 $p_1, p_2, \ldots p_m$	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.

Parameter Value	Block outputs...	Comments
[p_1 p_2 ... p_m] $p_1+p_2+...+p_m=n$ 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 ... p_m] $p_1+p_2+...+p_m$!=n p_i = > 0	Error	

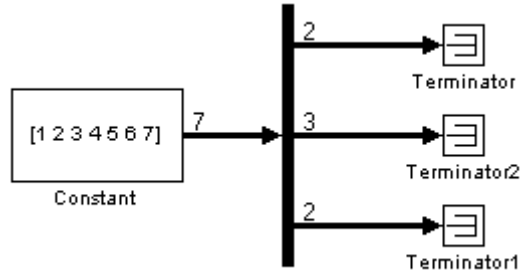
Note that you can specify the number of outputs as fewer than the number of input elements, in which case the block distributes the elements as evenly as possible over the outputs as illustrated in the following example.



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.

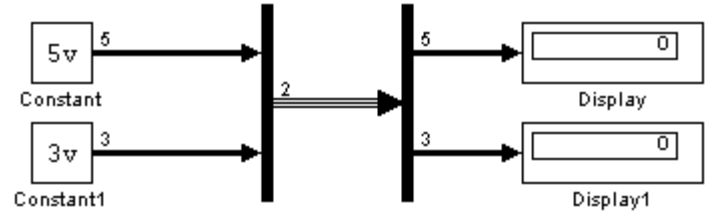


Specifying the Number of Outputs in Bus Selection Mode

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`.

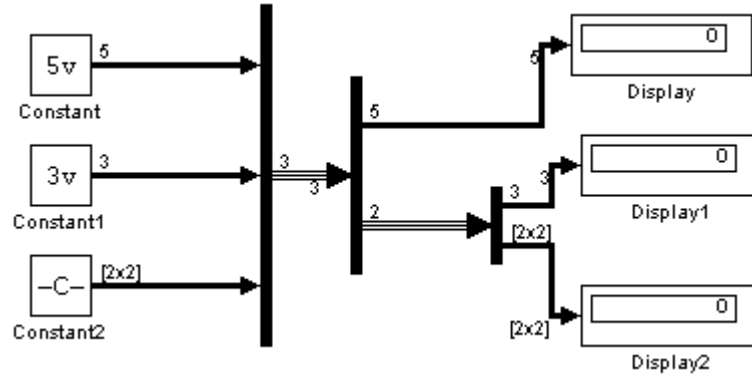


- Vector each of whose elements specifies the number of signals to output on the corresponding port

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.

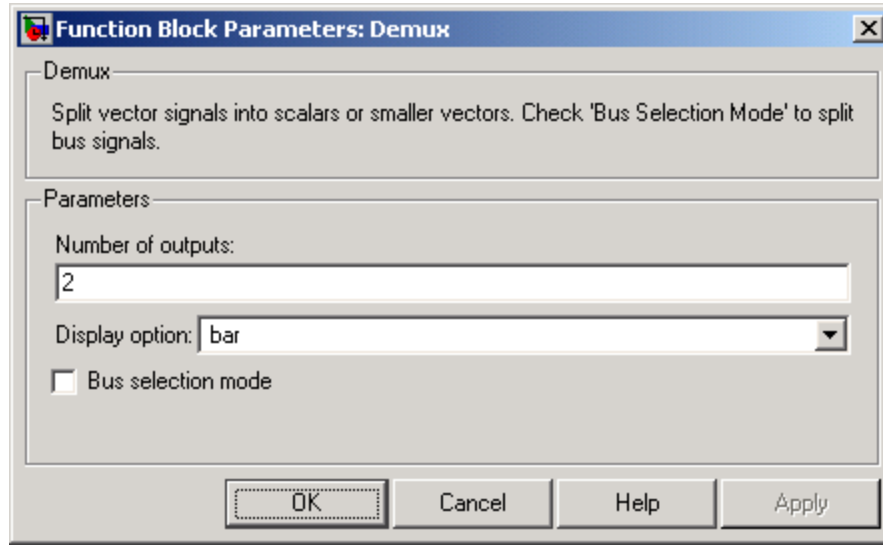
Note The MathWorks discourages enabling **Bus selection mode** and using a Demux block to extract elements of a bus signal. Muxes and buses should not be intermixed in new models. See "Avoiding Mux/Bus Mixtures" for more information.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Number of outputs

Specify the number and dimensions of outputs.

Settings

Default: 2

This block interprets this parameter depending on the **Bus selection mode** parameter. See the block description for more information.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

See the Demux block reference page for more information.

Display option

Select options to display the Demux block. The options are

Settings

Default: `bar`

`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.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See the Demux block reference page for more information.

Bus selection mode

Enable bus selection mode.

Settings

Default: Off

 On

Enable bus selection mode.

 Off

Disable bus selection mode.

Tips

The MathWorks discourages enabling **Bus selection mode** and using a Demux block to extract elements of a bus signal. Muxes and buses should not be intermixed in new models. See "Avoiding Mux/Bus Mixtures" for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Virtual	Yes For more information, see "Virtual Blocks" in the Simulink documentation.

See Also

Mux

Derivative

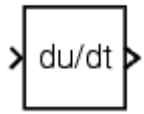
Purpose

Output time derivative of 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.

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

See "Circuit Model" in the *Simulink User's Guide* for an example of choosing the best-form mathematical model to avoid using Derivative blocks in your models.

Improved Linearization with Switched Derivative Blocks

Using `linmod` to linearize a model that contains a Derivative block can be troublesome. Before linearizing your model, replace each Derivative block with a block that avoids the problems. In the Library Browser, go to the Simulink Extras library and open the Linearization

sublibrary. For each instance of the Derivative block in your model, use the Switched derivative for linearization block.

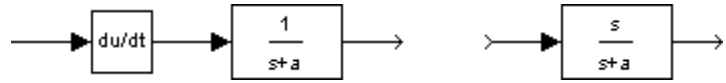
For more information, see the `linmod` reference page and "Linearizing Models" in the Simulink documentation.

Improved Linearization with Transfer Fcn Blocks

To improve linearization, you can also try to incorporate the derivative term in other blocks. For example, if you have a Derivative block in series with a Transfer Fcn block, try using a single Transfer Fcn block of the form

$$\frac{s}{s+a}$$

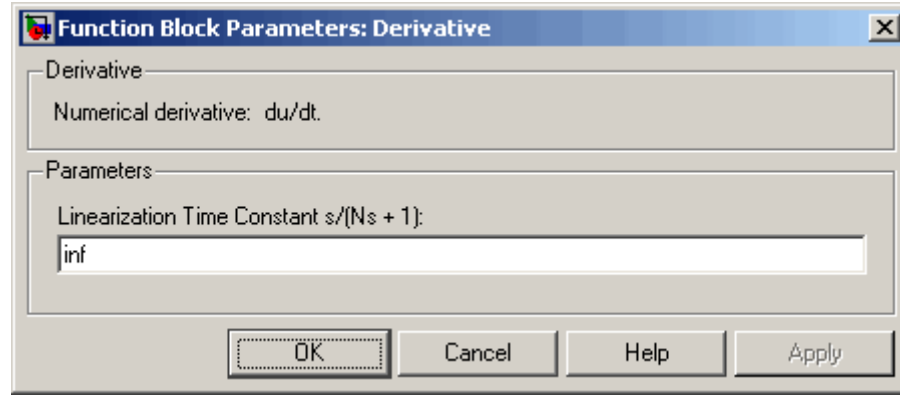
For example, you can replace the blocks on the left of this figure by using the block on the right.



Data Type Support

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

Parameters and Dialog Box



Linearization Time Constant s/(Ns + 1)

Specify the time constant N to approximate the linearization of your system.

Settings

Default: `inf`

- The default value `inf` corresponds to a linearization of 0.
- The exact linearization of the Derivative block is difficult, because the dynamic equation for the block is $y = \dot{u}$, which you cannot represent as a state-space system. However, you can approximate the linearization by adding a pole to the Derivative to create a transfer function $\frac{s}{Ns+1}$. The addition of a pole filters the signal before differentiating it, which removes the effect of noise.

Tip

A best practice is to change the value to $\frac{1}{f_b}$, where f_b is the break frequency for the filter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

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

See Also

Discrete Derivative

Purpose

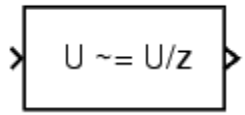
Detect change in signal's value

Library

Logic and Bit Operations

Description

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

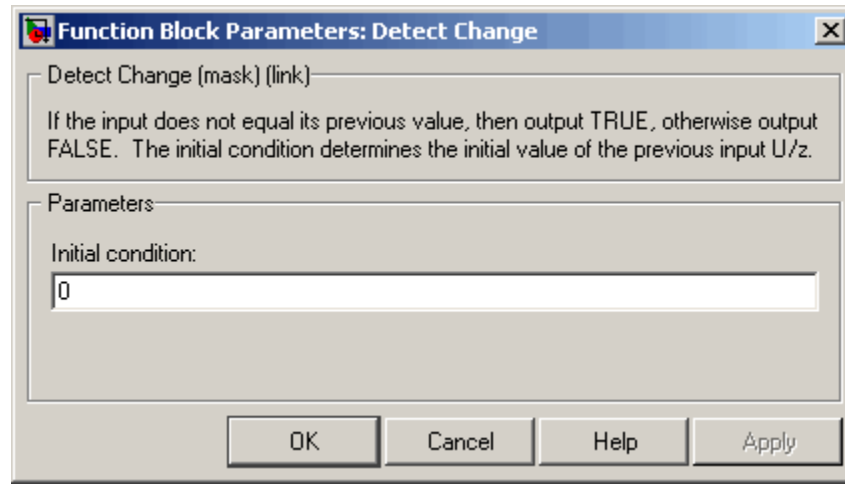


- The output is true (equal to `1`), when the input signal does not equal its previous value.
- The output is false (equal to `0`), when the input signal equals its previous value.

Data Type Support

The Detect Change block accepts signals of any data type supported by Simulink software, including fixed-point and enumerated data types. The block output is `uint8`.

Parameters and Dialog Box



Initial condition

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

Detect Change

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

Purpose

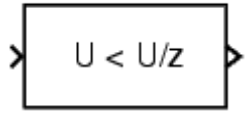
Detect decrease in signal's value

Library

Logic and Bit Operations

Description

The Detect Decrease block determines if an input is strictly less than its previous value where

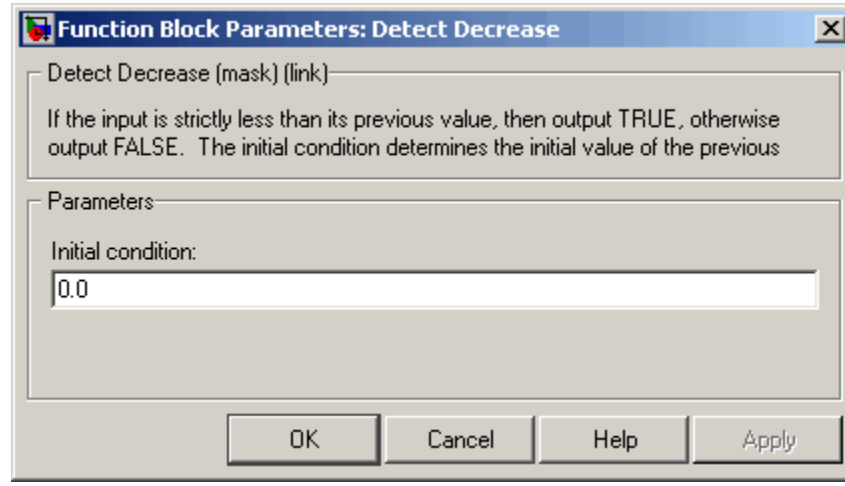


- The output is true (equal to `1`), when the input signal is less than its previous value.
- The output is false (equal to `0`), when the input signal is greater than or equal to its previous value.

Data Type Support

The Detect Decrease block accepts signals of any data type supported by Simulink software, including fixed-point and enumerated data types. The block output is `uint8`.

Parameters and Dialog Box



Initial condition

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

Detect Decrease

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

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

Purpose

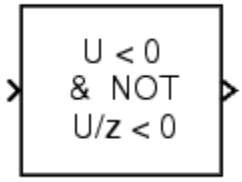
Detect falling edge when signal's value decreases to strictly negative value, and its previous value was nonnegative

Library

Logic and Bit Operations

Description

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



- The output is true (equal to 1), when the input signal is less than zero, and its previous value was greater than or equal to zero.
- The output is false (equal to 0), when the input signal is greater than or equal to zero, or if the input signal is nonnegative, its previous value was positive or zero.

Data Type Support

The Detect Fall Negative block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types. The block output is `uint8`.

Parameters and Dialog Box

Initial condition

Set the initial condition of the Boolean expression U/z < 0.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

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

Detect Fall Nonpositive

Purpose

Detect falling edge when signal's value decreases to nonpositive value, and its previous value was strictly positive

Library

Logic and Bit Operations

Description

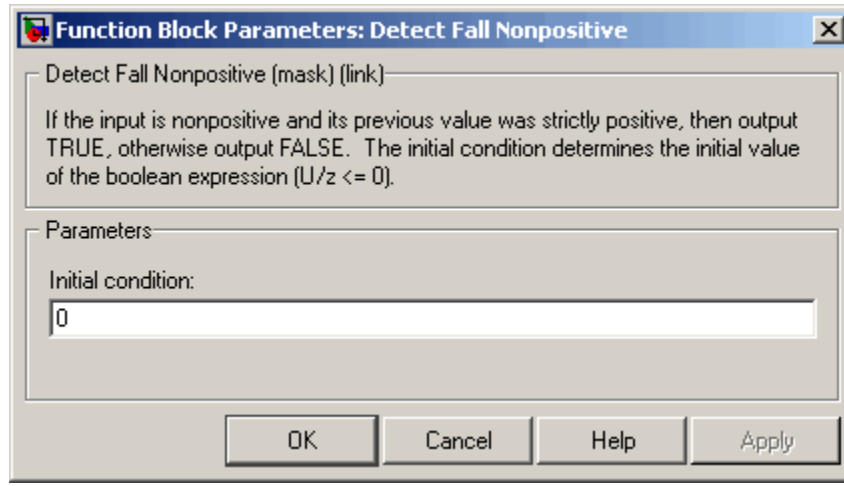
The Detect Fall Nonpositive block determines if the input is less than or equal to zero, and its previous value was positive where:

- The output is true (equal to 1), when the input signal is less than or equal to zero, and its previous value was greater than zero.
- The output is false (equal to 0), when the input signal is greater than zero, or if it is nonpositive, its previous value was nonpositive.

Data Type Support

The Detect Fall Nonpositive block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types. The block output is `uint8`.

Parameters and Dialog Box



Initial condition

Set the initial condition of the Boolean expression U/z <= 0.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

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

Detect Increase

Purpose

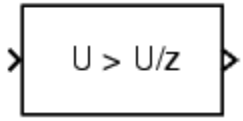
Detect increase in signal's value

Library

Logic and Bit Operations

Description

The Detect Increase block determines if an input is strictly greater than its previous value where

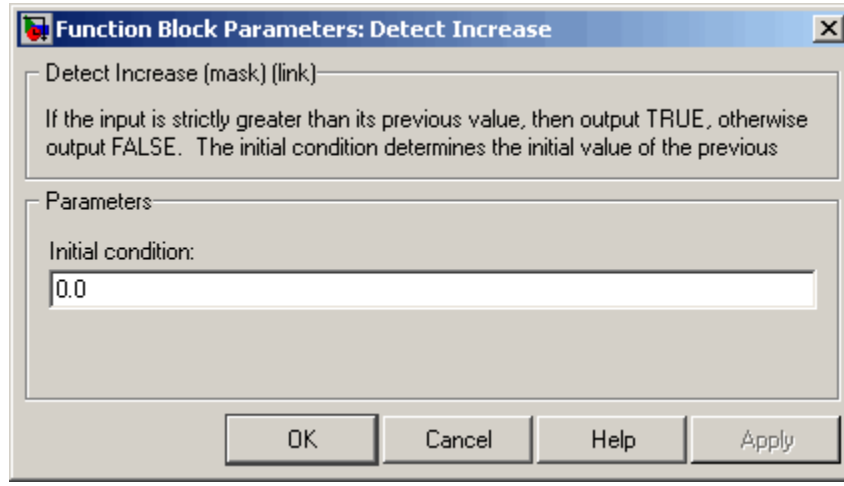


- The output is true (equal to `1`), when the input signal is greater than its previous value.
- The output is false (equal to `0`), when the input signal is less than or equal to its previous value.

Data Type Support

The Detect Increase block accepts signals of any data type supported by Simulink software, including fixed-point and enumerated data types. The block output is `uint8`.

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

Detect Rise Nonnegative

Purpose

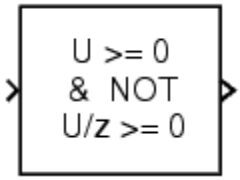
Detect rising edge when signal's value increases to nonnegative value, and its previous value was strictly negative

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

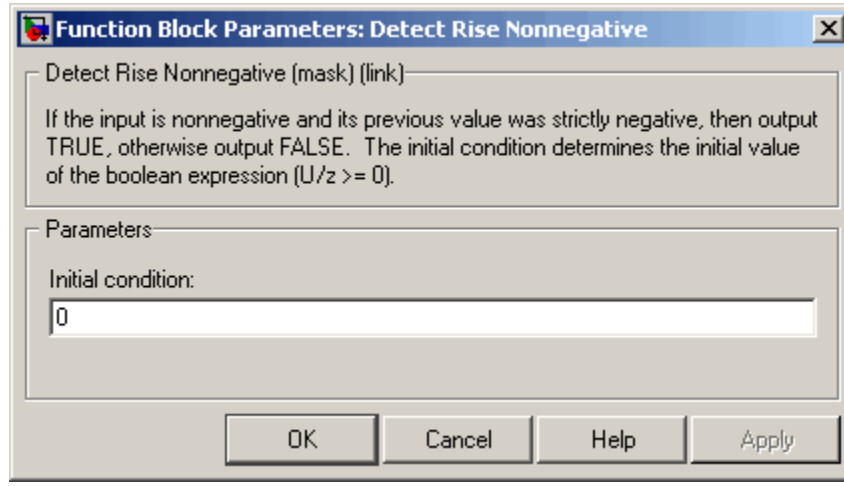


- The output is true (equal to 1), when the input signal is greater than or equal to zero, and its previous value was less than zero.
- The output is false (equal to 0), when the input signal is less than zero, or if nonnegative, its previous value was greater than or equal to zero.

Data Type Support

The Detect Rise Nonnegative block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types. The block output is `uint8`.

Parameters and Dialog Box



Initial condition

Set the initial condition of the Boolean expression U/z >= 0.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

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

Detect Rise Positive

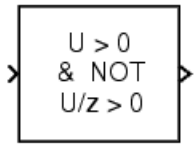
Purpose

Detect rising edge when signal's value increases to strictly positive value, and its previous value was nonpositive

Library

Logic and Bit Operations

Description



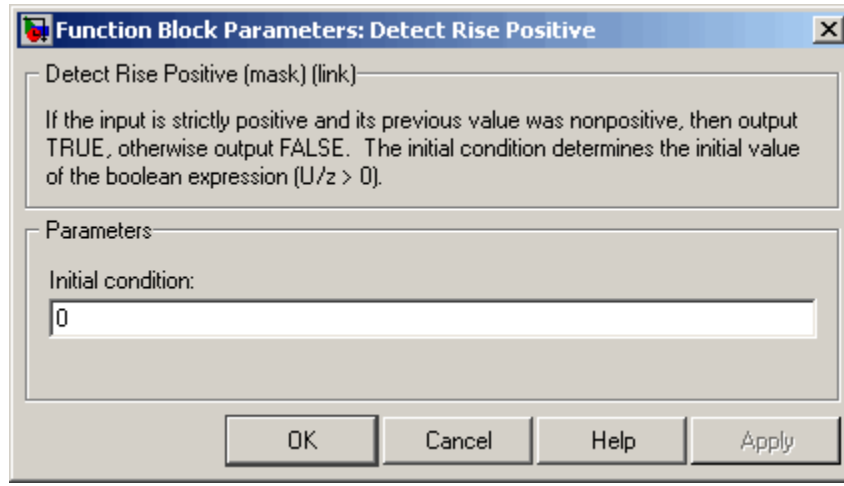
The Detect Rise Positive block determines if the input is strictly positive, and its previous value was nonpositive where

- The output is true (equal to 1), when the input signal is greater than zero, and its previous value was less than zero.
- The output is false (equal to 0), when the input is negative or zero, or if the input is positive, its previous value was also positive.

Data Type Support

The Detect Rise Positive block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types. The block output is `uint8`.

Parameters and Dialog Box



Initial condition

Set the initial condition of the Boolean expression U/z > 0.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

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

Difference

Purpose

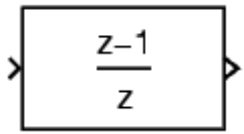
Calculate change in signal over one time step

Library

Discrete

Description

The Difference block outputs the current input value minus the previous input value.

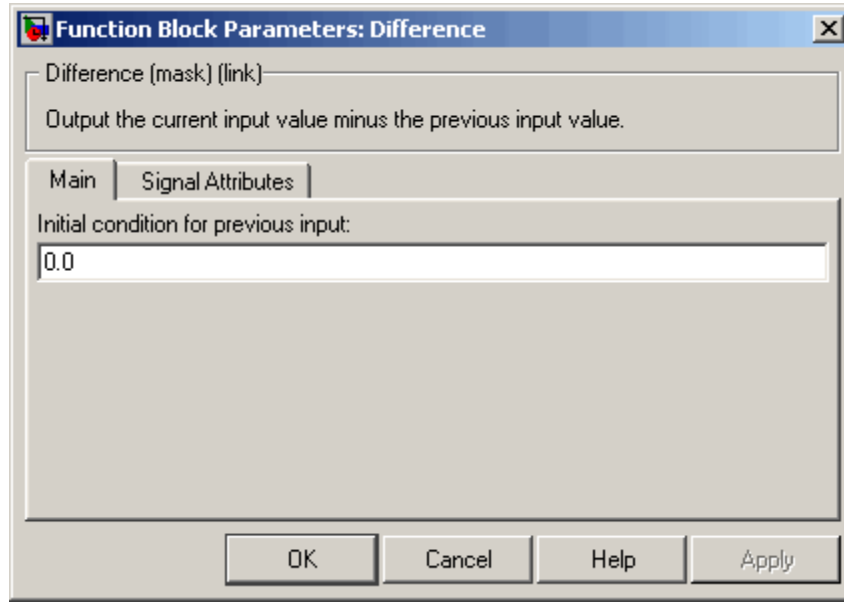


Data Type Support

The Difference block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

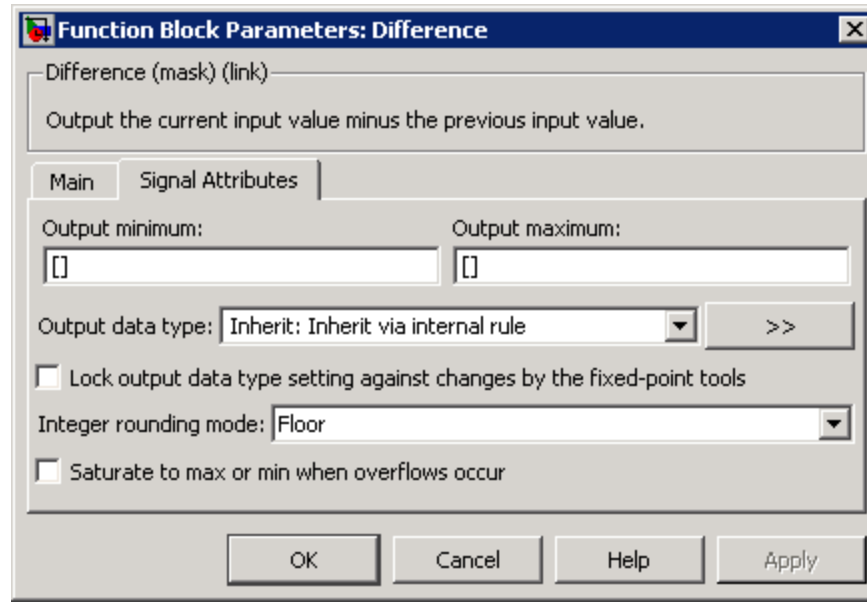
Parameters and Dialog Box

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



Initial condition for previous output
Set the initial condition for the previous output.

The **Signal Attributes** pane of the Difference block dialog box appears as follows:



Output minimum
Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum
Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in the *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see “Rounding” in the *Simulink Fixed Point User's Guide*.

Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate. Otherwise, they wrap.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of inputs and gain

Digital Clock

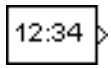
Purpose

Output simulation time at specified sampling interval

Library

Sources

Description



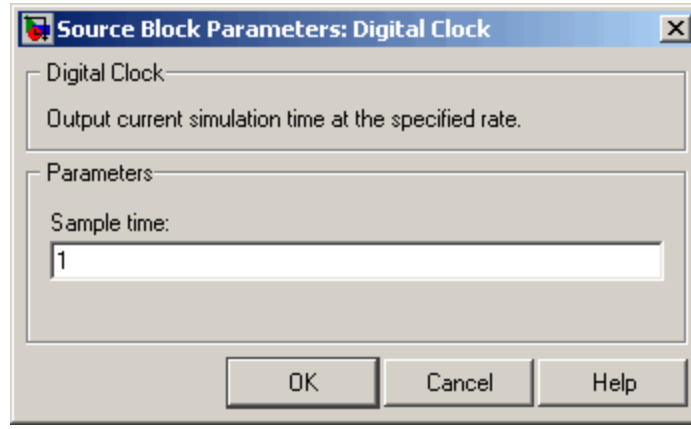
The Digital Clock block outputs the simulation time only at the specified sampling interval. At other times, the output is held at the previous value.

Use this block rather than the Clock block (which outputs continuous time) when you need the current time within a discrete system.

Data Type Support

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

Parameters and Dialog Box



Sample time

The sampling interval. The default value is `1` second. See Specifying Sample Time in the "How Simulink Works" chapter of the Simulink documentation.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	No
Zero Crossing	No

Direct Lookup Table (n-D)

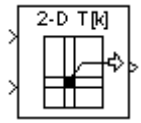
Purpose

Index into N-dimensional table to retrieve element, column, or 2-D matrix

Library

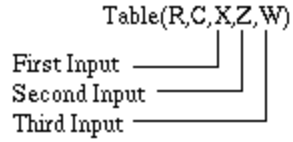
Lookup Tables

Description



The Direct Lookup Table (n-D) block uses its inputs as zero-based indices into an *n*-dimensional table. The number of inputs varies with the shape of the output, which can be an element, a column, or a 2-D matrix. The lookup table uses zero-based indexing, so integer data types can fully address their range. For example, a table dimension using the `uint8` data type can address all 256 elements.

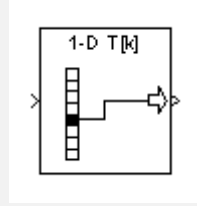
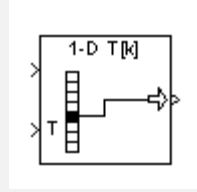
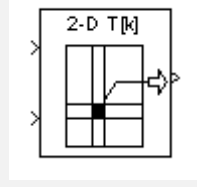
You define a set of output values as the **Table data** parameter. You specify what object the inputs select from the table: an element, a column, or a 2-D matrix. The first input specifies the zero-based index to the first dimension higher than the number of dimensions in the output, the next input specifies the index to the next table dimension, and so on:

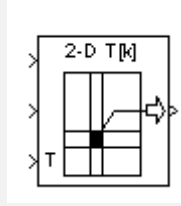
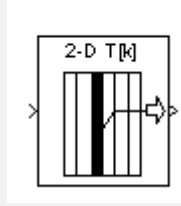
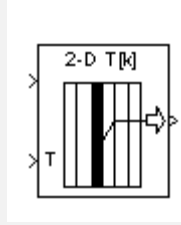


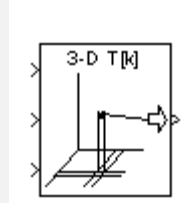
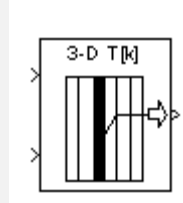
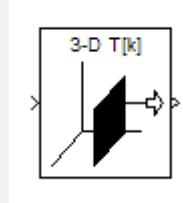
The figure shows a 5-D table, and the output is a 2-D matrix with R rows and C columns. (See “Changing a Block’s Orientation” in the Simulink documentation for a description of the port order for various block orientations.)

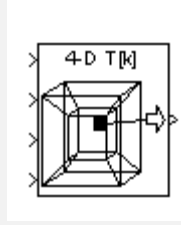
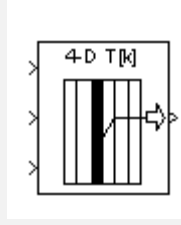
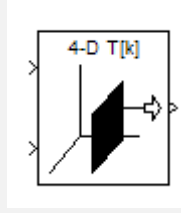
Changes in Block Icon Appearance

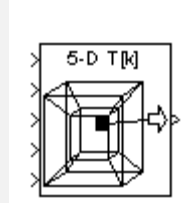
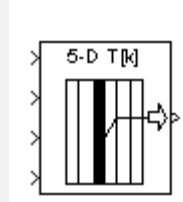
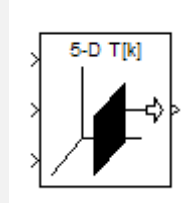
Depending on parameters you set in the block dialog box, the block icon changes appearance. For table dimensions higher than 4, the icon image matches the 4-D version but shows the exact number of dimensions in the top text.

Block Icon	Block Parameter Settings		
	Number of table dimensions	**Inputs select this object from table**	**Make table an input**
	1	Element	Not selected
	1	Element	Selected
	2	Element	Not selected

Block Icon	Block Parameter Settings		
	Number of table dimensions	**Inputs select this object from table**	**Make table an input**
	2	Element	Selected
	2	Column	Not selected
	2	Column	Selected

Block Icon	Block Parameter Settings		
	Number of table dimensions	**Inputs select this object from table**	**Make table an input**
	3	Element	Not selected When you select this check box, you can use only 1-D or 2-D table data, because this block does not support multidimensional inputs.
	3	Column	
	3	2-D Matrix	

Block Icon	Block Parameter Settings		
	Number of table dimensions	**Inputs select this object from table**	**Make table an input**
	4	Element	Not selected When you select this check box, you can use only 1-D or 2-D table data, because this block does not support multidimensional inputs.
	4	Column	
	4	2-D Matrix	

Block Icon	Block Parameter Settings		
	Number of table dimensions	**Inputs select this object from table**	**Make table an input**
	5	Element	Not selected When you select this check box, you can use only 1-D or 2-D table data, because this block does not support multidimensional inputs.
	5	Column	
	5	2-D Matrix	

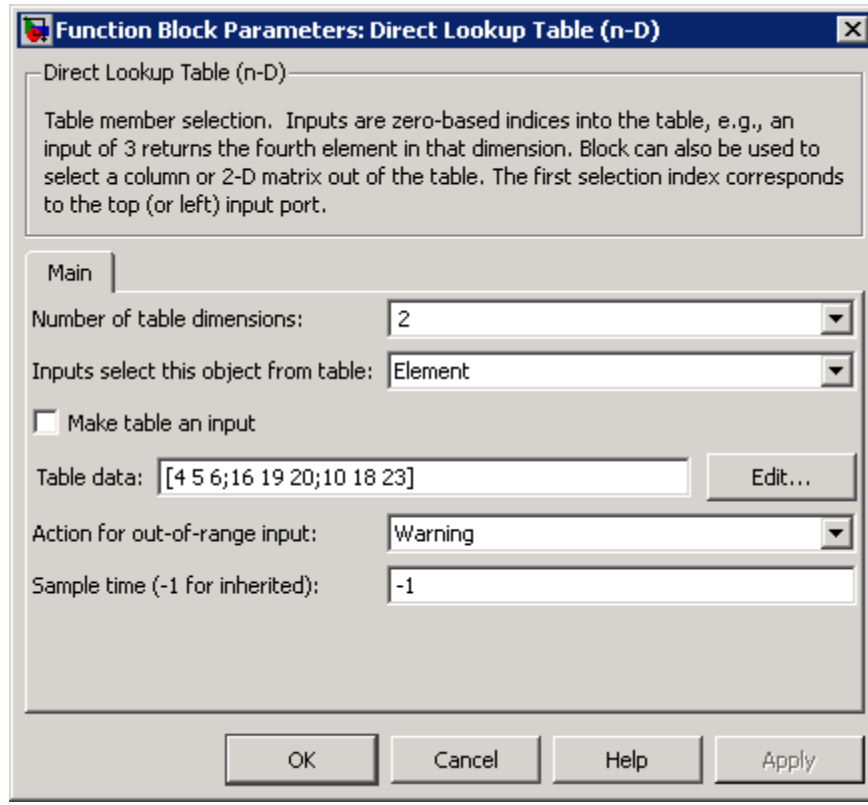
Data Type Support

The Direct Lookup Table (n-D) block accepts signals of a numeric data type supported by Simulink software, but not fixed-point data types. For a discussion on the data types supported by Simulink software,

see “Data Types Supported by Simulink” in the “Working with Data” chapter of the Simulink documentation.

The output type can differ from the input type and can be any of the types listed for input. When the table data is not an input, the block inherits the output type from the data type of the **Table data** parameter. When the table data is an input, the block inherits the output type from the table input port. Inputs for indexing must be real, but table data can be complex.

Parameters and Dialog Box



Number of table dimensions

The number of dimensions that the **Table data** parameter must have. This value determines the number of independent variables for the table and the number of inputs to the block. Options include: `1`, `2` (default), `3`, or `4`. To specify a higher number of dimensions, enter a positive integer directly in the field.

Inputs select this object from table

Specify whether the output data is a single element, a column, or a 2-D matrix. The number of input ports for indexing depends on your selection.

Selection	Number of Input Ports for Indexing
`Element`	Number of table dimensions
`Column`	Number of table dimensions - 1
`2-D Matrix`	Number of table dimensions - 2

This numbering matches MATLAB indexing. For example, if you have a 4-D table of data, follow these guidelines:

To access...	Specify...	As in...
An element	Four indices	`array(1,2,3,4)`
A column	Three indices	`array(:,2,3,4)`
A 2-D matrix	Two indices	`array(:,:,3,4)`

Make table an input

Selecting this check box forces the Direct Lookup Table (n-D) block to ignore the **Table data** parameter. Instead, a new input port appears with `T` next to it. Use this port to input table data.

Tip If you select this check box, use only 1-D or 2-D table data, because this block does not support multidimensional inputs.

Table data

The table of output values. The matrix size must match the dimensions of the **Number of table dimensions** parameter. During block diagram editing, you can leave the **Table data** field empty. But for simulation, you must match the number of dimensions in **Table data** to the **Number of table dimensions**. For details on how to construct multidimensional MATLAB arrays, see "Multidimensional Arrays" in the MATLAB documentation.

This parameter is available only if you clear the **Make table an input** check box.

Click **Edit** to open the Lookup Table Editor. For more information, see "Lookup Table Editor" in the Simulink documentation.

Action for out-of-range input

Specifies whether to produce a warning or error message when the input is out of range. Options include:

- `None` — the default, which means 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

Sample time (-1 for inherited)

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

Examples

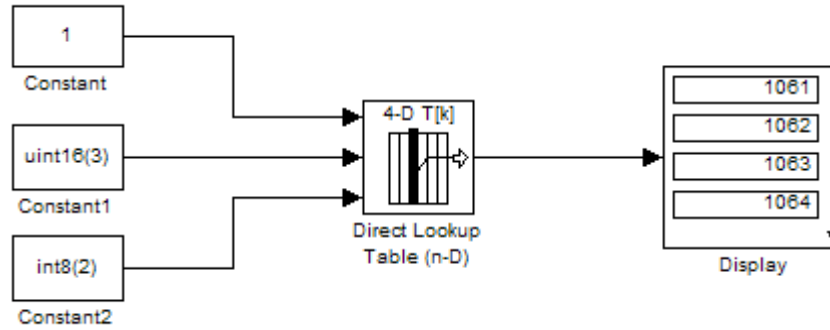
In this example, the block parameters are:

```
Number of table dimensions:  4
Inputs select this object from table:  "Column"
Table data:  int16(a)
```

where a is a 4-D array of linearly increasing numbers calculated with MATLAB functions.

```
a = ones(20,4,5,7);
L = numel(a);
a(1:L) = (1:L)';
```

The block output is the vector of 20 values in the second column, of the fourth element of the third dimension, from the third element of the fourth dimension.



The output has the same data type as the table, for example, `int16`. The block also uses zero-based indexing.

To calculate the output values manually, use the following MATLAB command (which uses one-based indexing):

```
a(:,1+1,1+3,1+2)

ans =

        1061
        1062
        1063
        1064
        1065
```

1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	For scalar lookups only (not when returning a column or a 2-D matrix from the table)
Dimensionalized	For scalar lookups only (not when returning a column or a 2-D matrix from the table)
Multidimensionalized	No
Zero-Crossing Detection	No

See Also

Lookup Table (n-D)

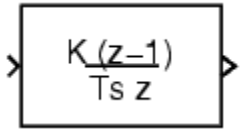
Purpose

Compute discrete time derivative

Library

Discrete

Description



The Discrete Derivative block computes an optionally scaled discrete time derivative as follows

$$y(t_n) = \frac{Ku(t_n)}{T_s} - \frac{Ku(t_{n-1})}{T_s}$$

where $u(t_n)$ and $y(t_n)$ are the block's input and output at the current time step, respectively, $u(t_{n-1})$ is the block's input at the previous time step, K is a scaling factor, and T_s is the simulation's discrete step size, which must be fixed.

Guidelines for Usage in Triggered Subsystems

When you use the Discrete Derivative block in triggered subsystems, follow these guidelines:

- When the **Sample time type** parameter of the trigger port is `triggered`, verify that your model does not trigger the subsystem at the first time step.

  If your model triggers the subsystem at the first time step, the Discrete Derivative block generates a divide-by-zero warning message. This behavior occurs because the time step at t = 0 is zero.

- When the **Sample time type** parameter of the trigger port is `periodic`, your model can trigger the subsystem at any time step.

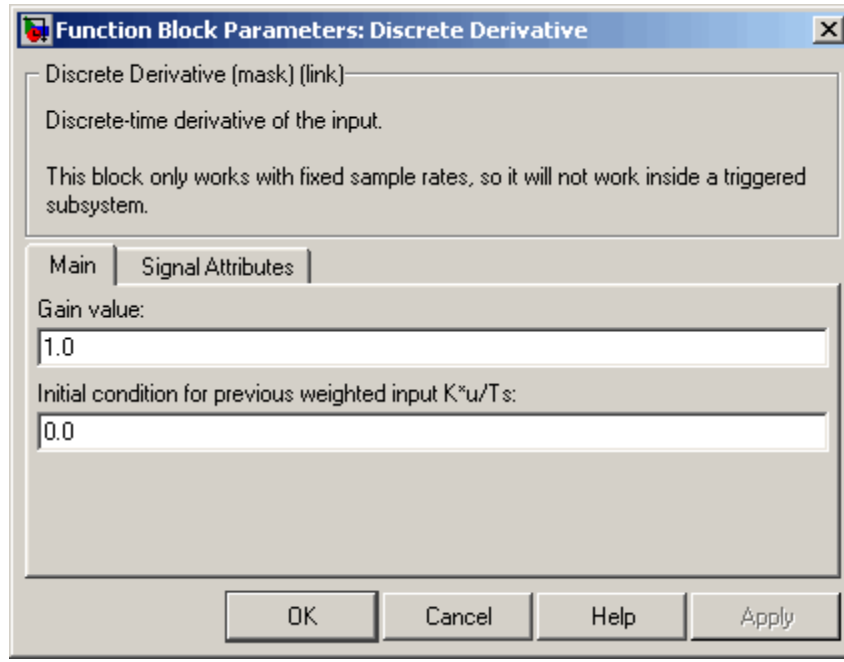
Data Type Support

The Discrete Derivative block supports all numeric Simulink data types, including fixed-point data types.

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box

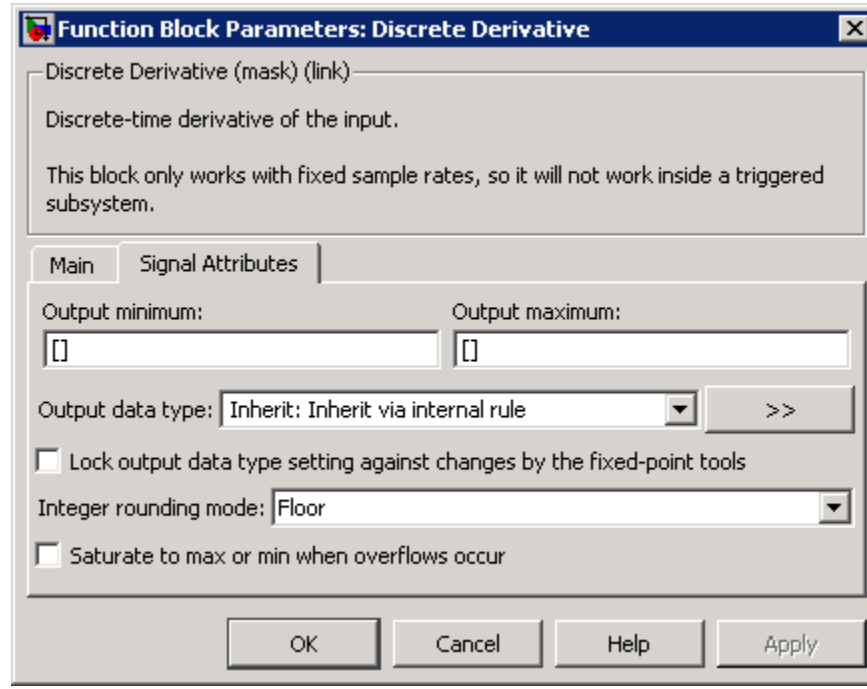
The **Main** pane of the Discrete Derivative block dialog box appears as follows:



Gain value
: Scaling factor used to weight the block's input at the current time step.

Initial condition for previous weighted input K*u/Ts
: Set the initial condition for the previous scaled input.

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



Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate. Otherwise, they wrap.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of inputs and gain

See Also Derivative

Discrete Filter

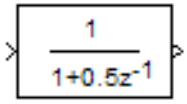
Purpose

Model Infinite Impulse Response (IIR) direct form II filters

Library

Discrete

Description



The Discrete Filter block independently filters each channel of the input signal with the specified digital IIR direct form II filter. The block implements static filters with fixed coefficients. You can tune the coefficients of a static filter.

This block filters each channel of the input signal independently over time, treating each element of the input as an individual channel. The output dimensions always equal the dimensions of the filtered input signal, except in single-input/multi-output mode.

The output of this block numerically matches the output of the Signal Processing Toolbox™ `dfilt.df2` function.

Use the **Numerator coefficients** parameter to specify the coefficients of the discrete filter numerator polynomial. Use the **Denominator coefficients** parameter to specify the coefficients of the denominator polynomial of the function. The **Denominator coefficients** parameter must be a vector of coefficients.

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

Specifying Initial States

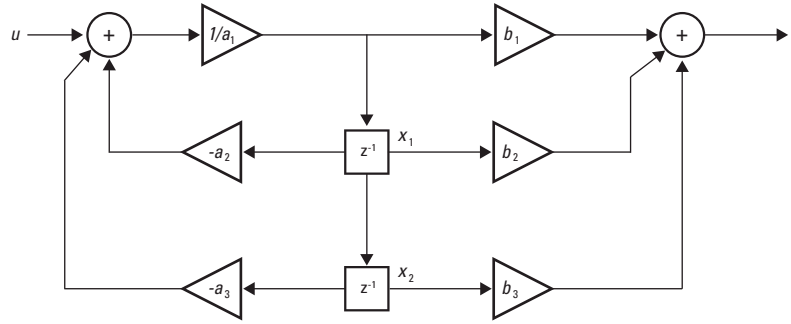
Use the **Initial states** parameter to specify initial filter states. To determine the number of initial states you must specify and how to specify them, see the following table of valid initial state dimensions. The **Initial states** parameter can take one of the forms described in this table.

Valid Initial States

Initial States	Description
Scalar	The block initializes all filter states to the same scalar value. Enter `0` to initialize all states to zero.
Vector or matrix	Each vector or matrix element specifies a unique initial state for a corresponding delay element in a corresponding channel: • The vector length must equal the number of delay elements in the filter, `max(number of zeros, number of poles)`. • The matrix must have the same number of rows as the number of delay elements in the filter, `max(number of zeros, number of poles)`. The matrix must also have one column for each channel of the input signal.

The following example shows the relationship between the initial filter output and the initial input and state. Given an initial input u_1, the first output y_1 is related to the initial state $[x_1, x_2]$ and initial input by:

$$y_1 = b_1\left[\frac{(u_1 - a_2x_1 - a_3x_2)}{a_1}\right] + b_2x_1 + b_3x_2$$



Initial States: 1

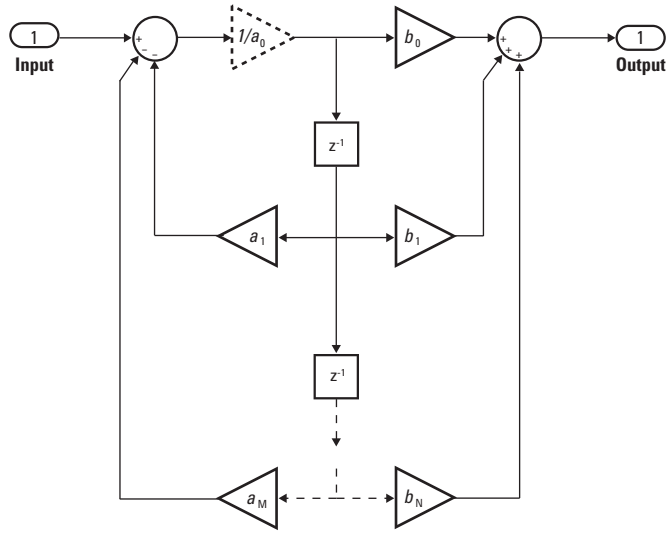
Data Type Support

The Discrete Filter block accepts and outputs real and complex signals of any signed numeric data type that Simulink supports. The block supports the same types for the numerator and denominator coefficients.

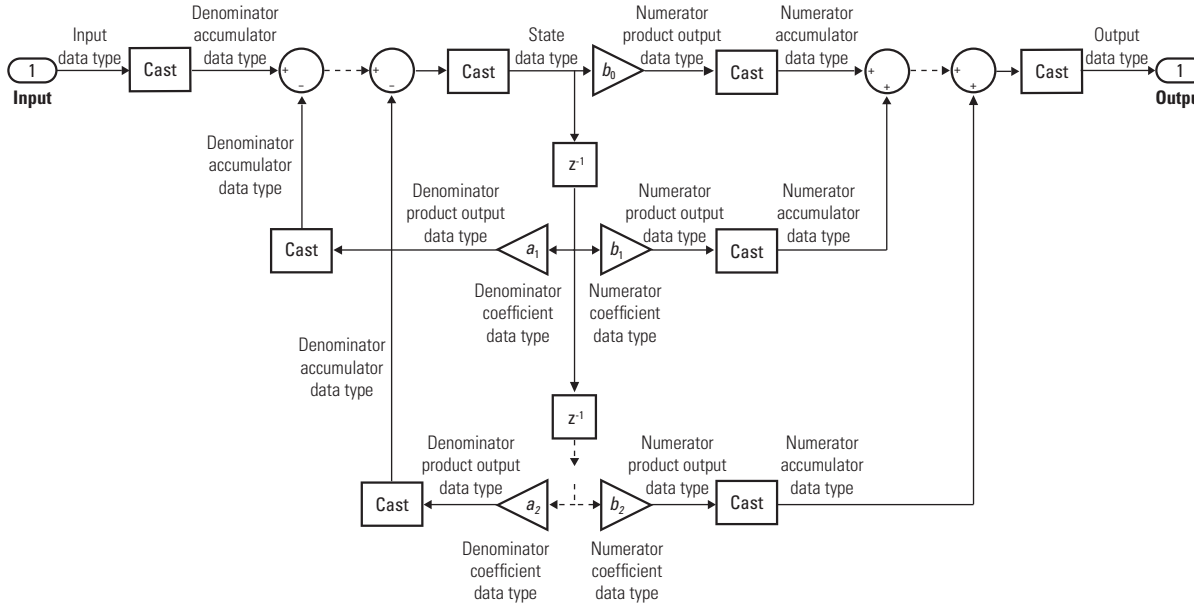
Numerator and denominator coefficients must have the same complexity. They can have different word lengths and fraction lengths.

States are complex when either the input or the coefficients are complex.

The following diagrams show the filter structure and the data types used within the Discrete Filter block for fixed-point signals.

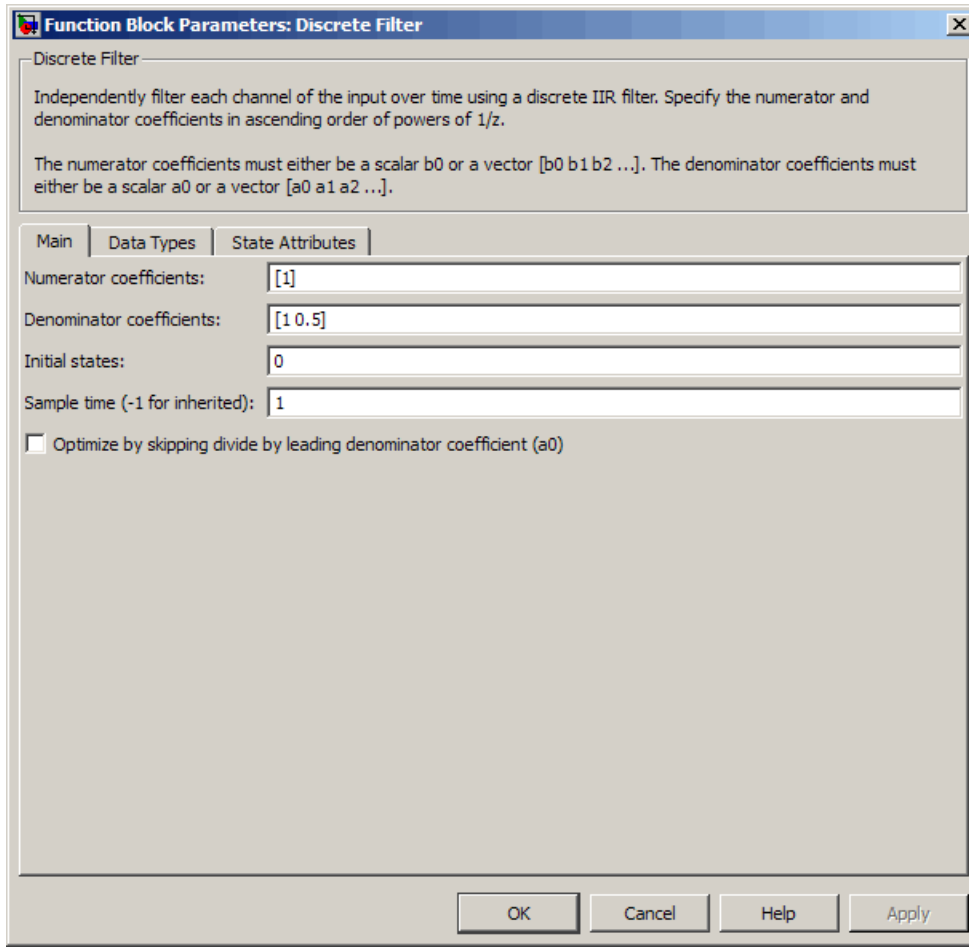


The block omits the dashed divide when you select the **Optimize by skipping divide by leading denominator coefficient (a0)** parameter.



Parameters and Dialog Box

The **Main** pane of the Discrete Filter block dialog box appears as follows.



Numerator coefficients

Specify the coefficients of the discrete filter numerator polynomial or polynomials in ascending powers of z^{-1}. Use a row vector to specify the coefficients for a single numerator polynomial.

Denominator coefficients

Specify the coefficients of the discrete filter denominator polynomial as a row vector in ascending powers of z^{-1}.

Initial states

Specify the initial states of the filter. To learn how to specify initial states, see "Specifying Initial States" on page 2-280.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in "How Simulink Works" in the *Simulink User's Guide*.

Optimize by skipping divide by leading denominator coefficient (a0)

Select when the leading denominator coefficient, a_0, equals one. This parameter generates optimized code.

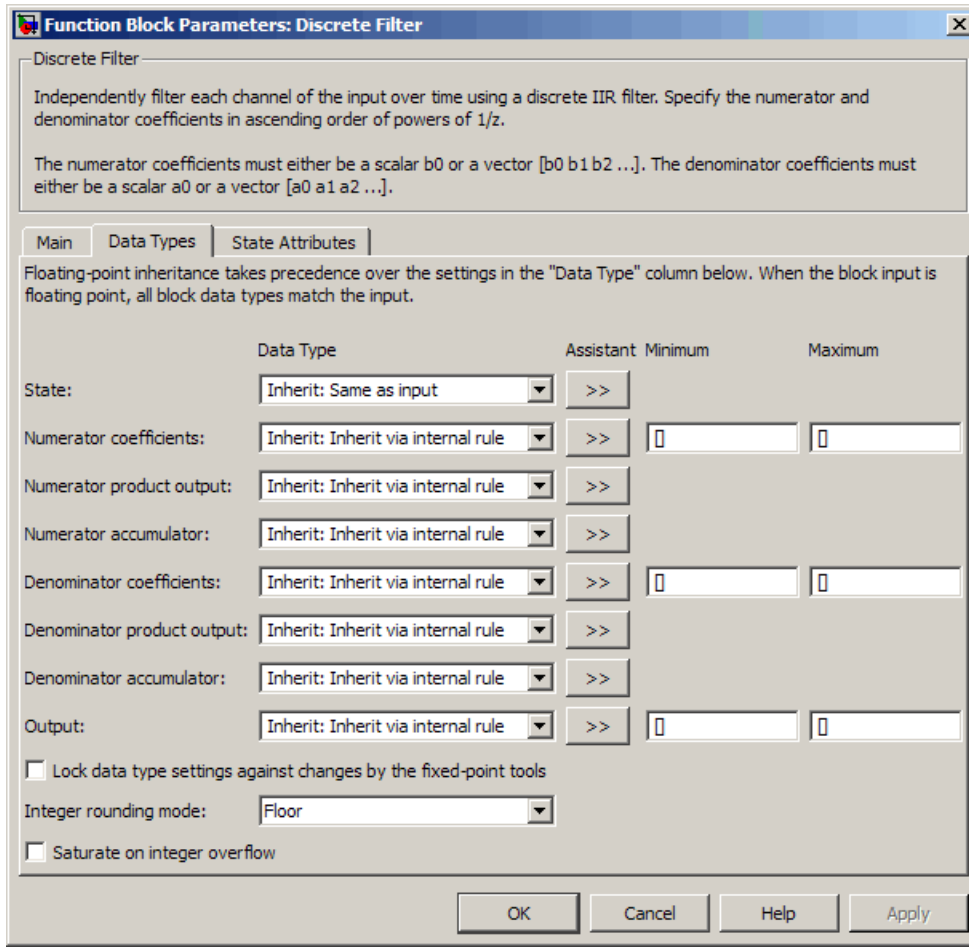
When you select this check box, the block:

- Does not perform a divide-by-a_0 either in simulation or in the generated code
- Errors out at model edit time if the a_0 value you provide in the dialog is not one
- Errors out if you tune a_0 to any nonunity value

When you clear this check box, the block:

- Has denominator coefficients that are fully tunable during simulation
- Performs a divide-by-a_0 in both simulation and code generation

The **Data Types** pane of the Discrete Filter block dialog box appears as follows.



State

Specify the state data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- A built-in integer, for example, `int8`

- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **State** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Numerator coefficients

Specify the numerator coefficient data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in integer, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Numerator coefficients** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Numerator coefficient minimum

Specify the minimum value that a numerator coefficient can have. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")

- Automatic scaling of fixed-point data types

Numerator coefficient maximum

Specify the maximum value that a numerator coefficient can have. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Automatic scaling of fixed-point data types

Numerator product output

Specify the product output data type for the numerator coefficients. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Numerator product output** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Numerator accumulator

Specify the accumulator data type for the numerator coefficients. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`

- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Numerator accumulator** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Denominator coefficients

Specify the denominator coefficient data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in integer, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Denominator coefficients** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Denominator coefficient minimum

Specify the minimum value that a denominator coefficient can have. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")

- Automatic scaling of fixed-point data types

Denominator coefficient maximum

Specify the maximum value that a denominator coefficient can have. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Automatic scaling of fixed-point data types

Denominator product output

Specify the product output data type for the denominator coefficients. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Denominator product output** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Denominator accumulator

Specify the accumulator data type for the denominator coefficients. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`

- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Denominator accumulator** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Output

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output** parameter.

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Output minimum

Specify the minimum value that the block can output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")

- Automatic scaling of fixed-point data types

Output maximum
Specify the maximum value that the block can output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

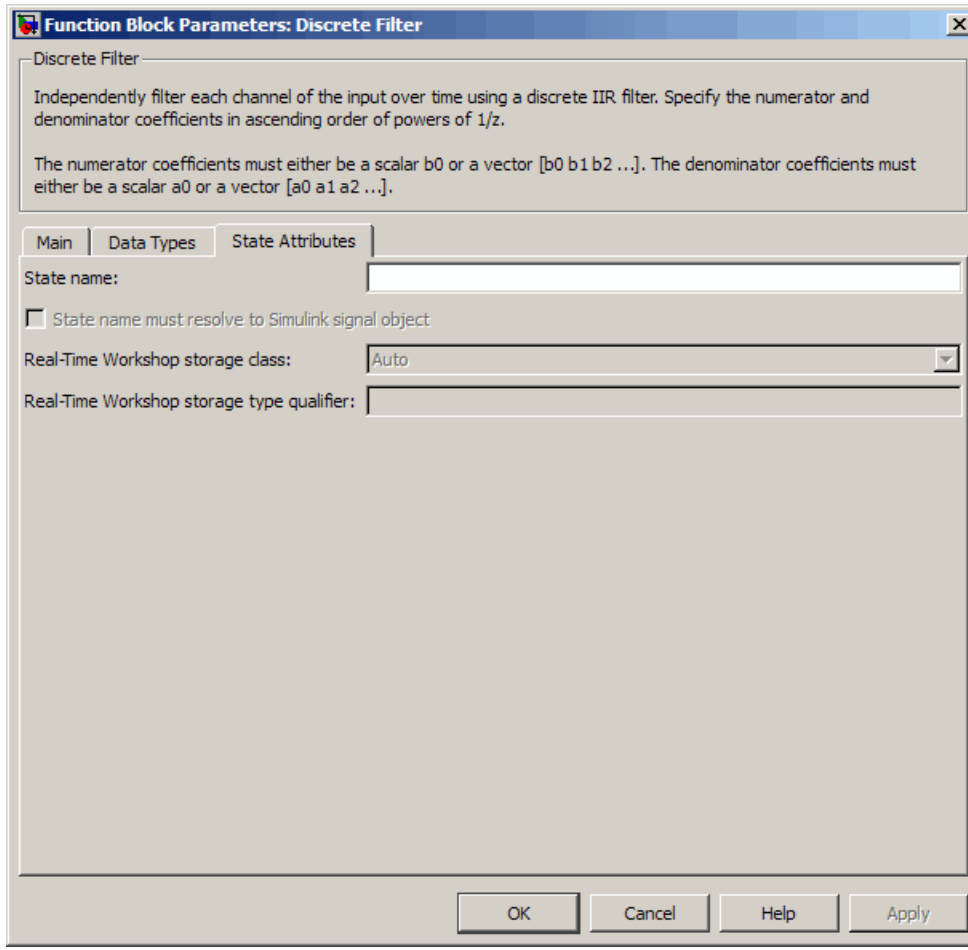
Lock data type settings against changes by the fixed-point tools
Select to lock all data type settings of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate on integer overflow
Select this check box to have overflows saturate. Otherwise, they wrap.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

The **State Attributes** pane of the Discrete Filter block dialog box appears as follows.



State name

Use this parameter to assign a unique name to each state. The default is `' '`. When this field is blank, no name is assigned. Consider the following when using this parameter:

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'`.

- The state names apply only to the selected block.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, `{'a', 'b', 'c'}`. Each name must be unique.
- The number of states must be an integer multiple of the number of state names. You can have fewer names than states, but you cannot have more names than states. For example, you can specify two names in a system with four states. Simulink software assigns the first name to the first two states and the second name to the last two.
- To assign state names with a variable that you have defined in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell, or structure.

This parameter enables **State name must resolve to Simulink signal object** when you click **Apply**.

State name must resolve to Simulink signal object

Select this check box to require that the state name resolves to a Simulink signal object. This check box is cleared by default.

Specifying the **State name** parameter enables this parameter.

Selecting this check box enables **Real-Time Workshop storage class**.

Real-Time Workshop storage class

From the list, select the state storage class.

`Auto`

If you do not need states to interface to external code, select `Auto` as the storage class.

`ExportedGlobal`

The class stores the state in a global variable.

`ImportedExtern`
model`_private.h` declares the state as an extern variable.

`ImportedExternPointer`
model`_private.h` declares the state as an extern pointer.

Specifying the **State name** parameter enables this parameter.

Setting this parameter to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables **Real-Time Workshop storage type qualifier**.

During simulation, the block uses the following values:

- The initial value of the signal object to which the state name resolves
- Minimum and maximum values of the signal object

See "Block State Storage and Interfacing Considerations" in the *Real-Time Workshop User's Guide* for more information.

Characteristics

Direct Feedthrough	Only when the leading numerator coefficient is not equal to zero
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of initial states
States	See "Specifying Initial States" on page 2-280
Dimensionalized	Yes
Zero-Crossing Detection	No

Discrete FIR Filter

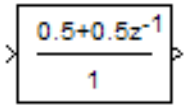
Purpose

Model FIR filters

Library

Discrete

Description



The Discrete FIR Filter block independently filters each channel of the input signal with the specified digital FIR filter. The block can implement static filters with fixed coefficients, as well as time-varying filters with coefficients that change over time. You can tune the coefficients of a static filter during simulation.

This block filters each channel of the input signal independently over time, treating each element of the input as an individual channel. The output dimensions equal those of the input, except in single-input/multi-output mode.

The outputs of this block numerically match the outputs of the Signal Processing Blockset Digital Filter Design block and of the Signal Processing Toolbox `dfilt` function.

This block supports the Simulink state logging feature. See “States” in the *Simulink User’s Guide* for more information.

Filter Structure Support

You can change the filter structure implemented with the Discrete FIR Filter block by selecting one of the following from the **Filter structure** parameter:

- `Direct form`
- `Direct form symmetric`
- `Direct form antisymmetric`
- `Direct form transposed`
- `Lattice MA`

You must have an available Signal Processing Blockset software license to run a model with any of these filter structures other than direct form.

Specifying Initial States

The Discrete FIR Filter block initializes the internal filter states to zero by default, which has the same effect as assuming that past inputs and outputs are zero. You can optionally use the **Initial states** parameter to specify nonzero initial conditions for the filter delays.

To determine the number of initial states you must specify and how to specify them, see the table on valid initial states. The **Initial states** parameter can take one of the forms described in the next table.

Valid Initial States

Initial Condition	Description
Scalar	The block initializes all delay elements in the filter to the scalar value.
Vector or matrix (for applying different delay elements to each channel)	Each vector or matrix element specifies a unique initial condition for a corresponding delay element in a corresponding channel: • The vector length equal the product of the number of input channels and the number of delay elements in the filter, `#_of_filter_coeffs-1` (or `#_of_reflection_coeffs` for `Lattice MA`). • The matrix must have the same number of rows as the number of delay elements in the filter, `#_of_filter_coeffs-1` (`#_of_reflection_coeffs` for `Lattice MA`), and must have one column for each channel of the input signal.

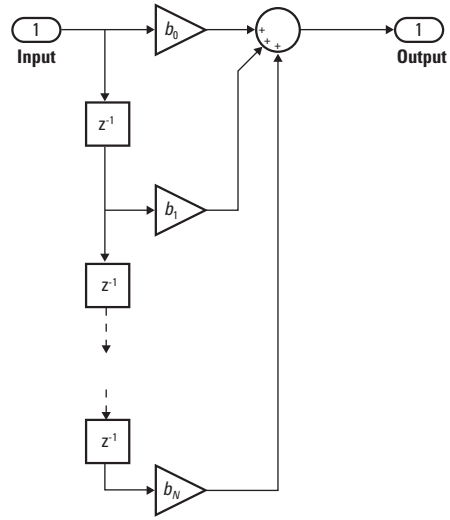
Data Type Support

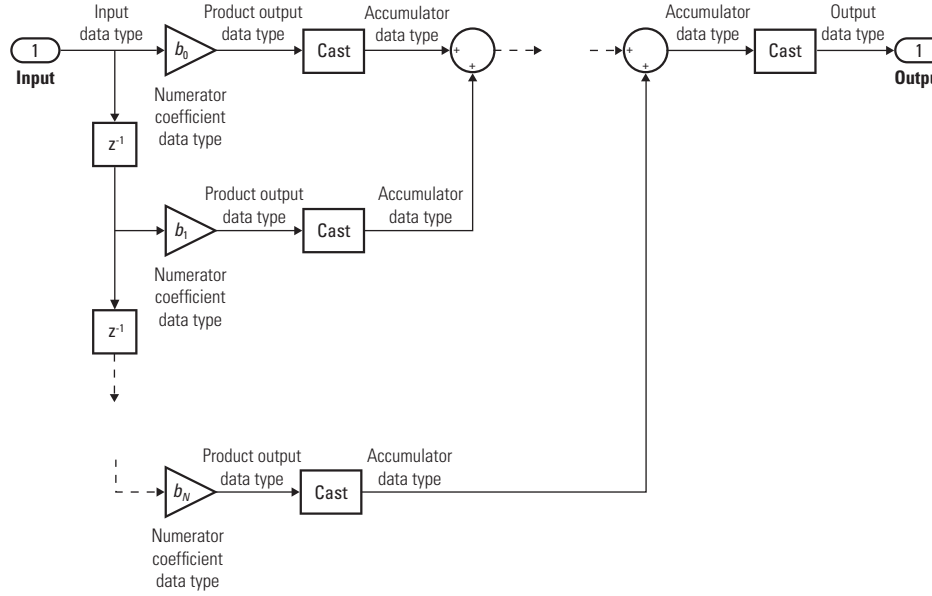
The Discrete FIR Filter block accepts and outputs real and complex signals of any numeric data type supported by Simulink. The bock supports the same types for the numerator coefficients.

The following diagrams show the filter structure and the data types used within the Discrete FIR Filter block for fixed-point signals.

Direct Form

You cannot specify the state data type on the block mask for this structure because the input states have the same data types as the input.

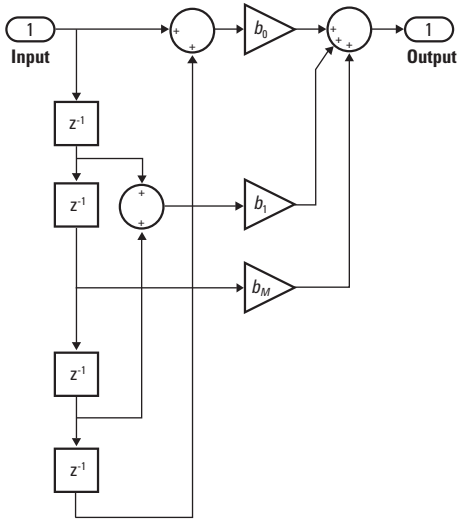



Input
data type
Product output
data type
Accumulator
data type
Accumulator
data type
Output
data type
1
Input
b_0
Cast
Cast
1
Output
Numerator
coefficient
data type
z^{-1}
Product output
data type
Accumulator
data type
b_1
Cast
Numerator
coefficient
data type
z^{-1}
Product output
data type
Accumulator
data type
b_N
Cast
Numerator
coefficient
data type

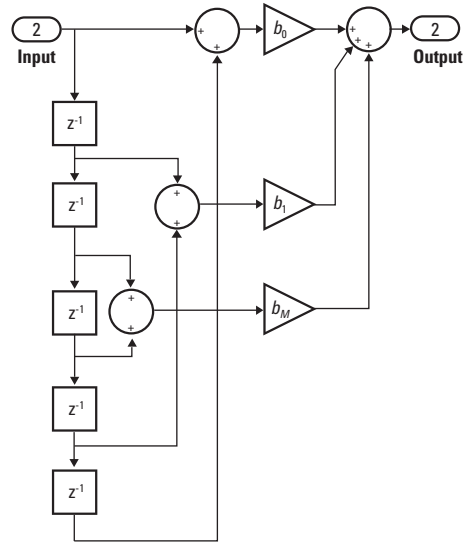
Direct Form Symmetric

You cannot specify the state data type on the block mask for this structure because the input states have the same data types as the input.

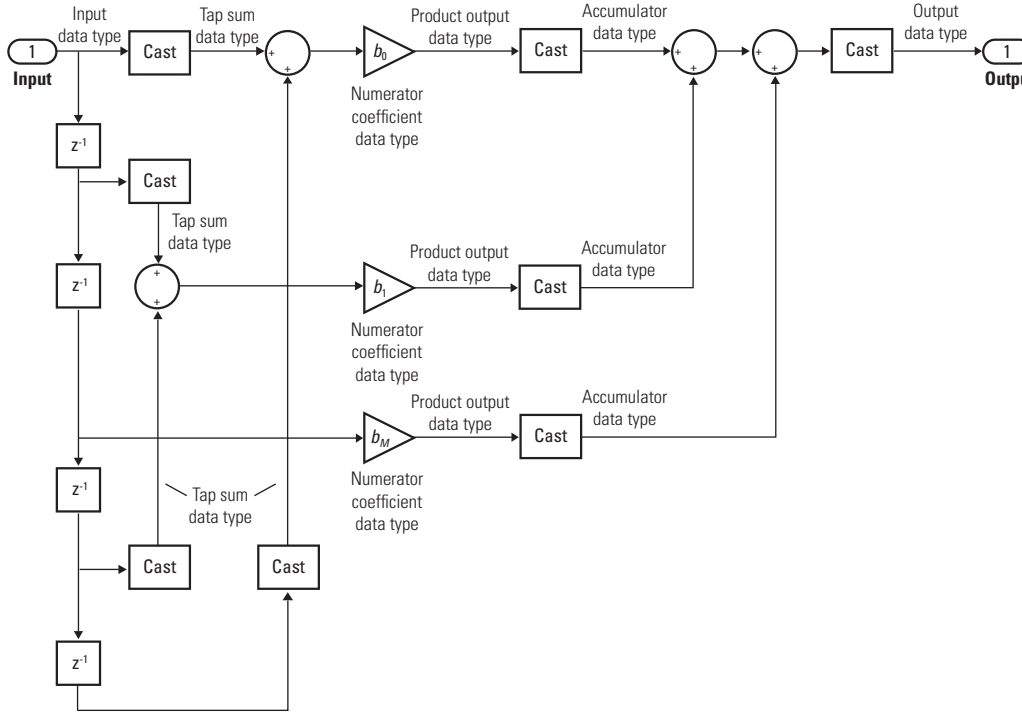
It is assumed that the filter coefficients are symmetric. The block only uses the first half of the coefficients for filtering.



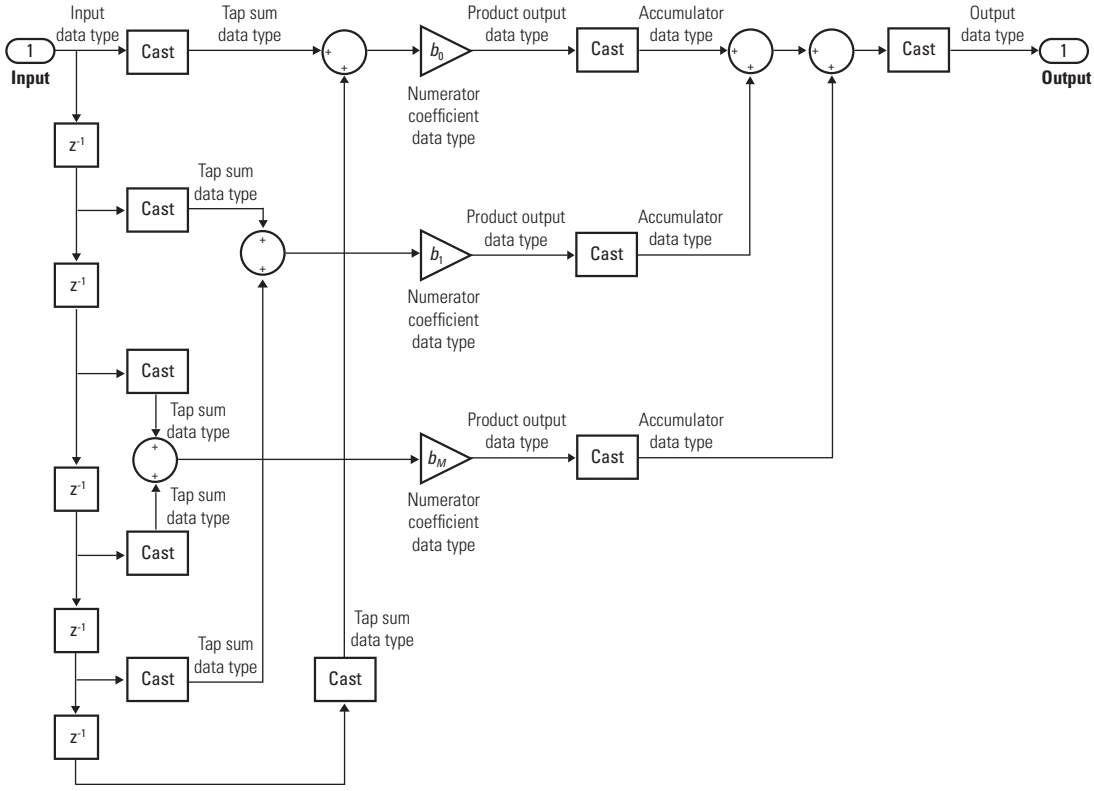
Even Order - Type I



Odd Order - Type II



Even Order - Type I

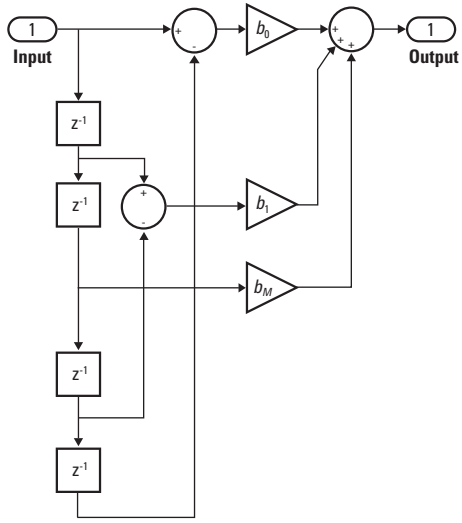


Odd Order - Type II

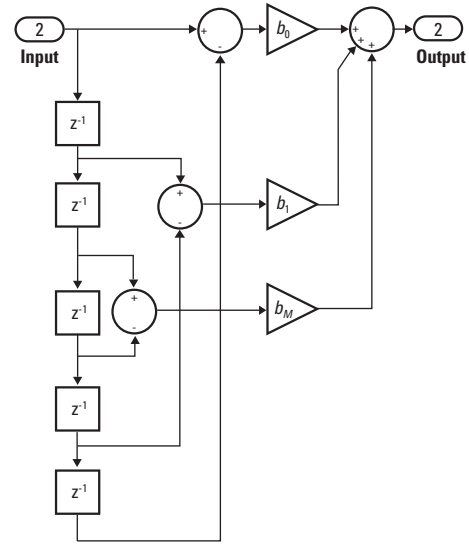
Direct Form Antisymmetric

You cannot specify the state data type on the block mask for this structure because the input states have the same data types as the input.

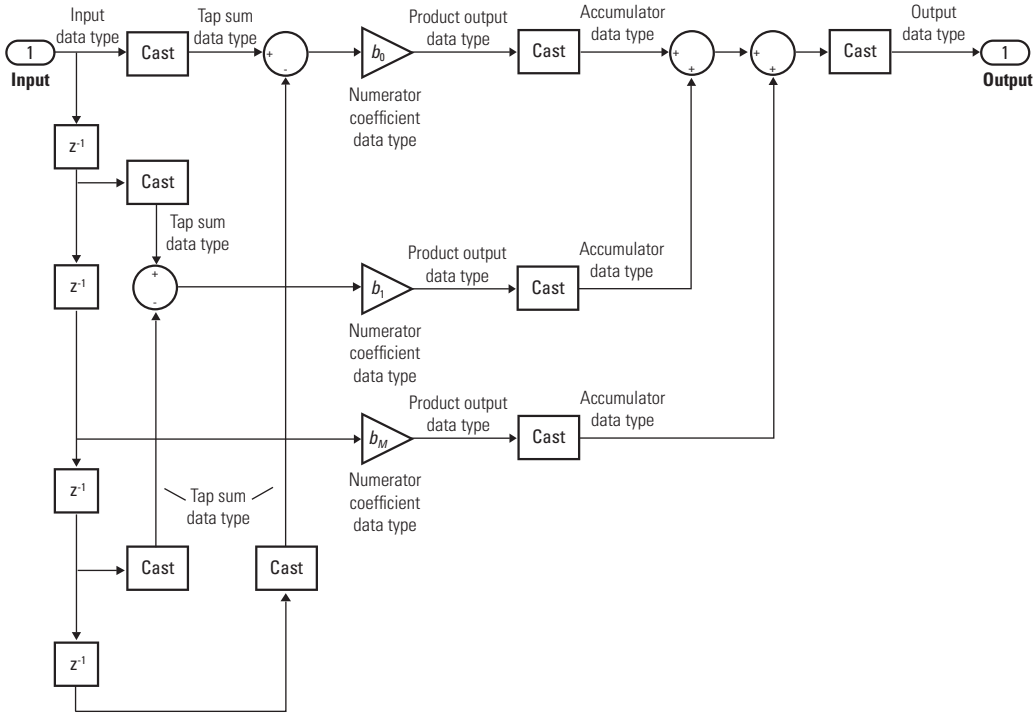
It is assumed that the filter coefficients are antisymmetric. The block only uses the first half of the coefficients for filtering.



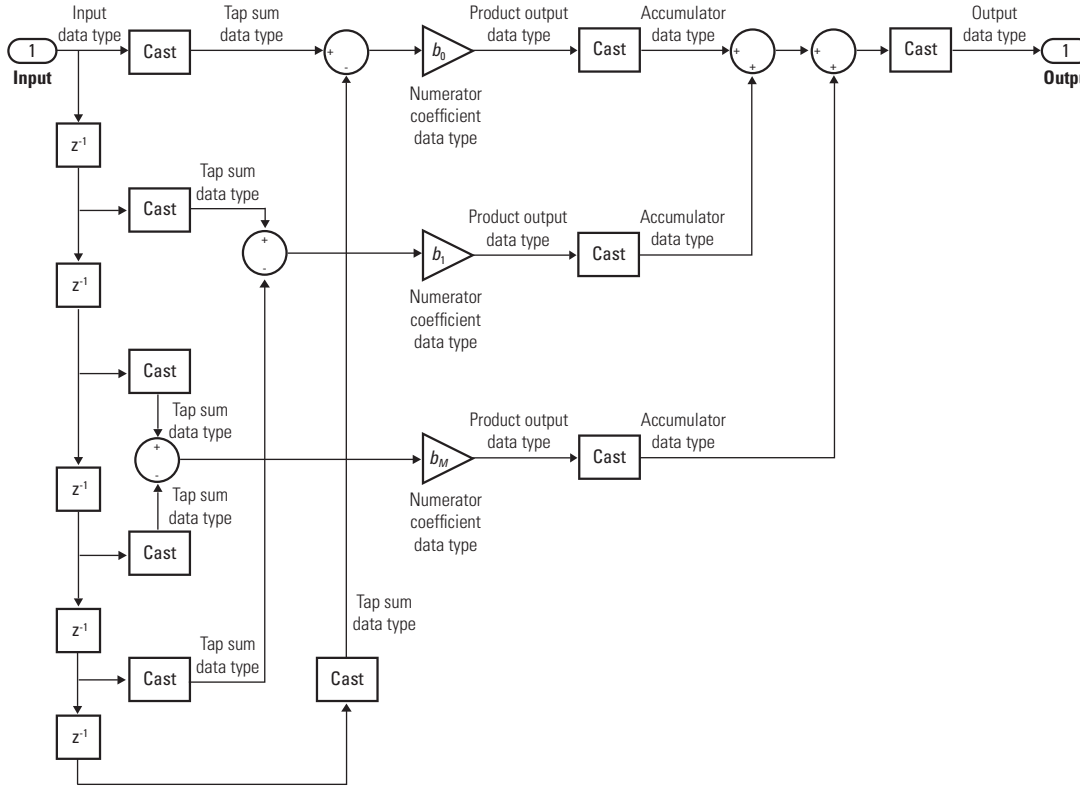
Even Order - Type III



Odd Order - Type IV



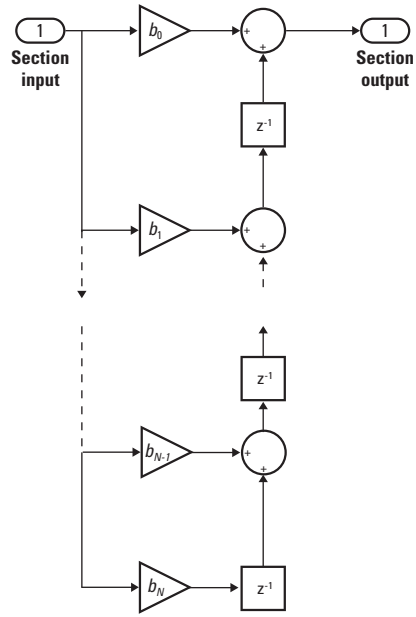
Even Order - Type III

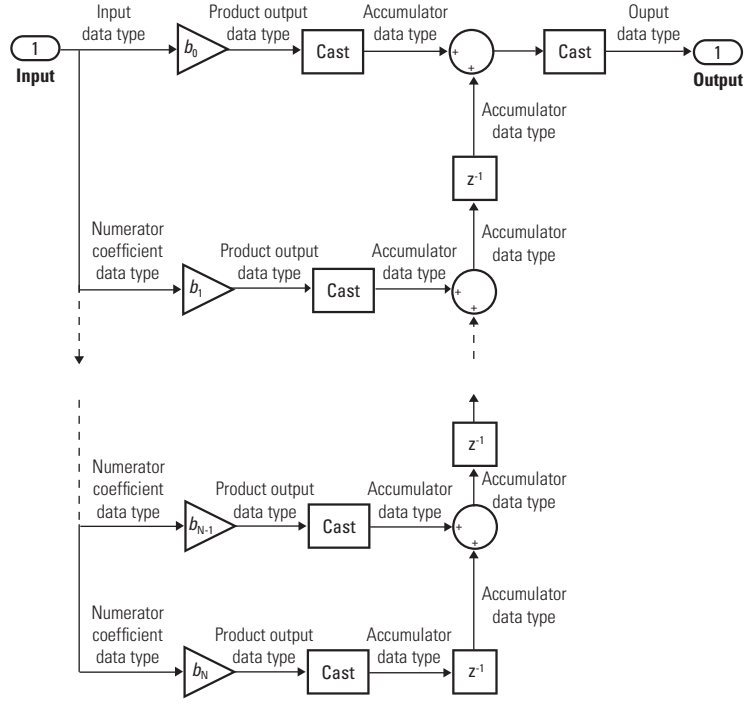


Odd Order - Type IV

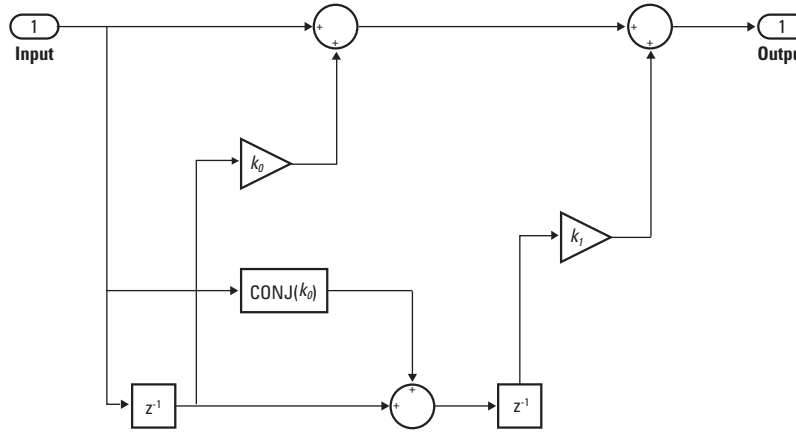
Direct Form Transposed

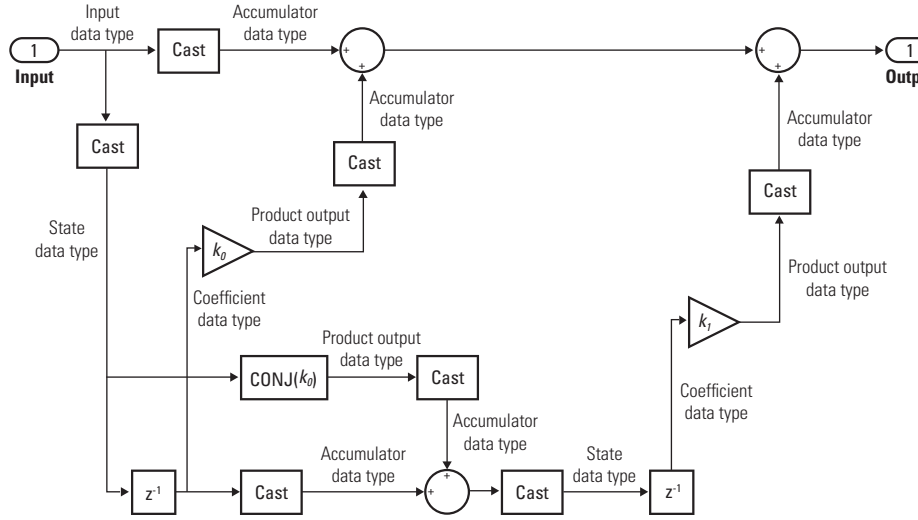
States are complex when either the inputs or the coefficients are complex.




1
Input
Input data type
b_0
Product output data type
Cast
Accumulator data type
Cast
Ouput data type
1
Output
Accumulator data type
z^{-1}
Numerator coefficient data type
b_1
Product output data type
Cast
Accumulator data type
Accumulator data type
z^{-1}
Numerator coefficient data type
b_{N-1}
Product output data type
Cast
Accumulator data type
Accumulator data type
Numerator coefficient data type
b_N
Product output data type
Cast
Accumulator data type
z^{-1}

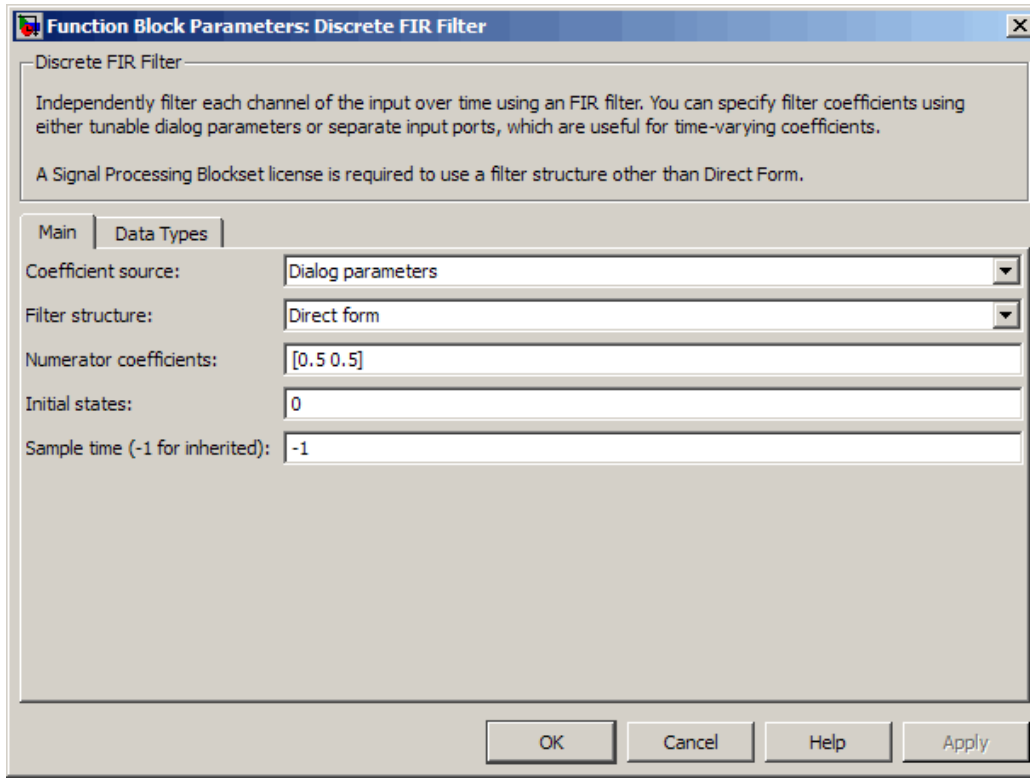
Lattice MA


1
Input
1
Output
k_0
k_1
CONJ(k_0)
z^{-1}
z^{-1}


1
Input
Input data type
Cast
Accumulator data type
1
Output
Cast
Accumulator data type
Cast
Accumulator data type
Cast
State data type
Product output data type
k_0
Coefficient data type
Product output data type
k_1
CONJ(k_0)
Product output data type
Cast
Coefficient data type
Accumulator data type
Accumulator data type
State data type
z^{-1}
Cast
Cast
z^{-1}

Parameters and Dialog Box

The **Main** pane of the Discrete FIR Filter block dialog box appears as follows.



Coefficient source

Specify whether you want to input the filter coefficients on the block mask or inherit them from an input port.

Filter structure

Select the filter structure you want the block to implement. You must have an available Signal Processing Blockset software

license to run a model with a Discrete FIR Filter block that implements any filter structure other than direct form.

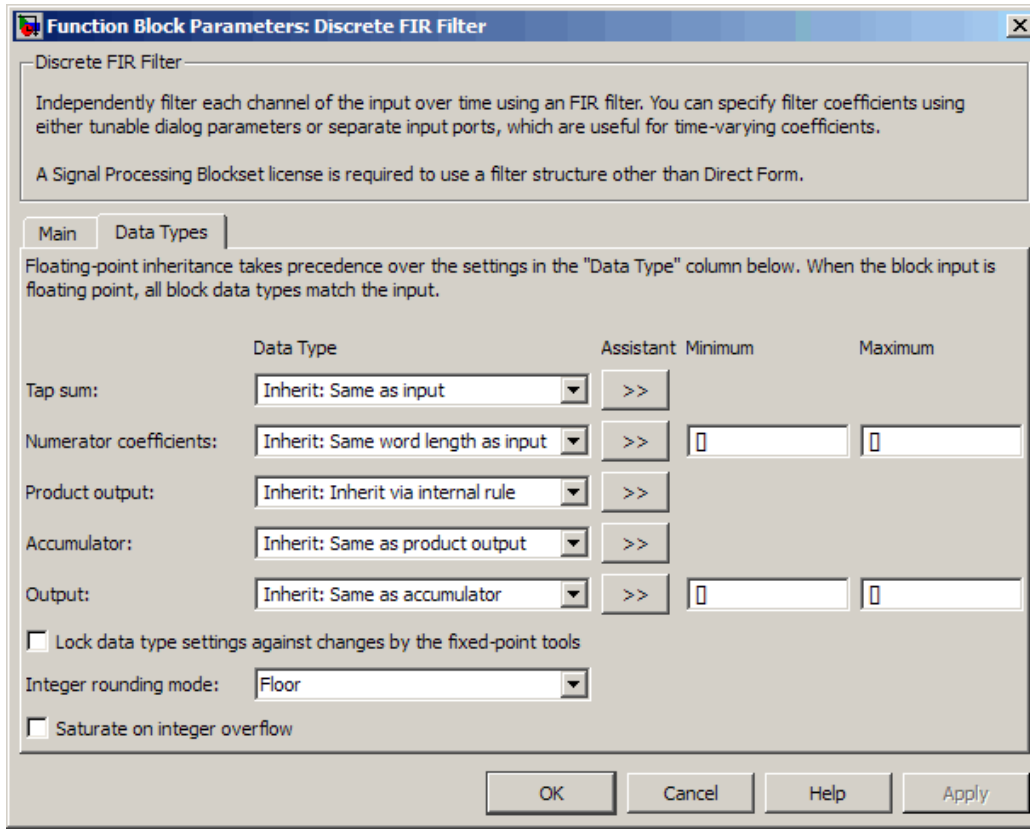
Numerator coefficients
Specify the vector of numerator coefficients of the filter's transfer function.

This parameter is only visible when `Dialog parameters` is selected for the **Coefficient source** parameter.

Initial states
Specify the initial conditions of the filter states. To learn how to specify initial states, see "Specifying Initial States" on page 2-297.

Sample time (-1 for inherited)
Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in "How Simulink Works" in the *Simulink User's Guide*.

The **Data Type Attributes** pane of the Discrete FIR Filter block dialog box appears as follows.



Tap sum

Specify the tap sum data type of a direct form symmetric or direct form antisymmetric filter, which is the data type the filter uses when it sums the inputs prior to multiplication by the coefficients. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- A built-in integer, for example, `int8`

- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

This parameter is only visible when the selected filter structure is either `Direct form symmetric` or `Direct form antisymmetric`.

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Tap sum** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Coefficient

Specify the coefficient data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same word length as input`
- A built-in integer, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Coefficient** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Coefficient minimum

Specify the minimum value that a filter coefficient should have. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Automatic scaling of fixed-point data types

Coefficient maximum

Specify the maximum value that a filter coefficient should have. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Automatic scaling of fixed-point data types

Product output

Specify the product output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Product output** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Accumulator

Specify the accumulator data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as product output`
- A built-in data type, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Accumulator** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

State

Specify the state data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- A built-in integer, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

This parameter is only visible when the selected filter structure is `Lattice MA`.

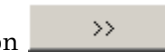
Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **State** parameter.

See "Using the Data Type Assistant" in *Simulink User's Guide* for more information.

Output

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as accumulator`
- A built-in data type, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output** parameter.

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Lock data type settings against changes by the fixed-point tools
Select to lock all data type settings of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate on integer overflow
Select to have overflows saturate. Otherwise, they wrap.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of initial states
States	See "Specifying Initial States" on page 2-297
Dimensionalized	Yes
Zero-Crossing Detection	No

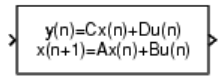
Purpose

Implement discrete state-space system

Library

Discrete

Description

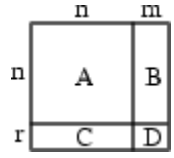


The Discrete State-Space block implements the system described by

$$x(n+1) = Ax(n) + Bu(n)$$
$$y(n) = Cx(n) + Du(n)$$

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.

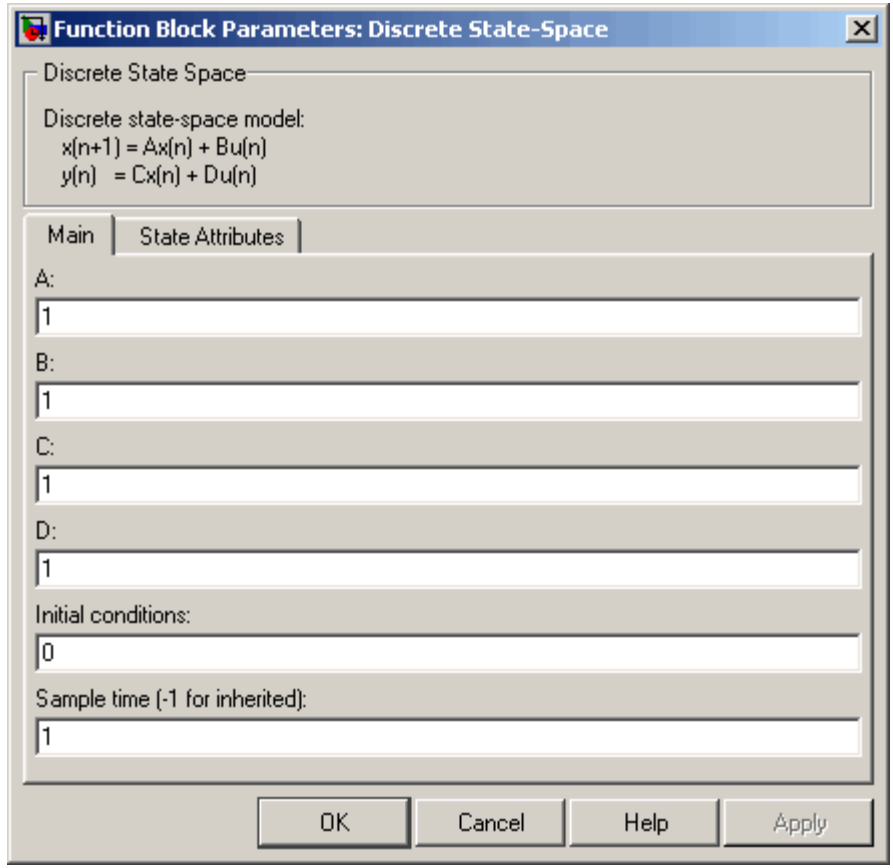
Simulink software converts a matrix containing zeros to a sparse matrix for efficient multiplication.

Data Type Support

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

Parameters and Dialog Box

The **Main** tab of the Discrete State-Space block dialog box appears as follows:



A, B, C, D

The matrix coefficients, as defined in the preceding equations.

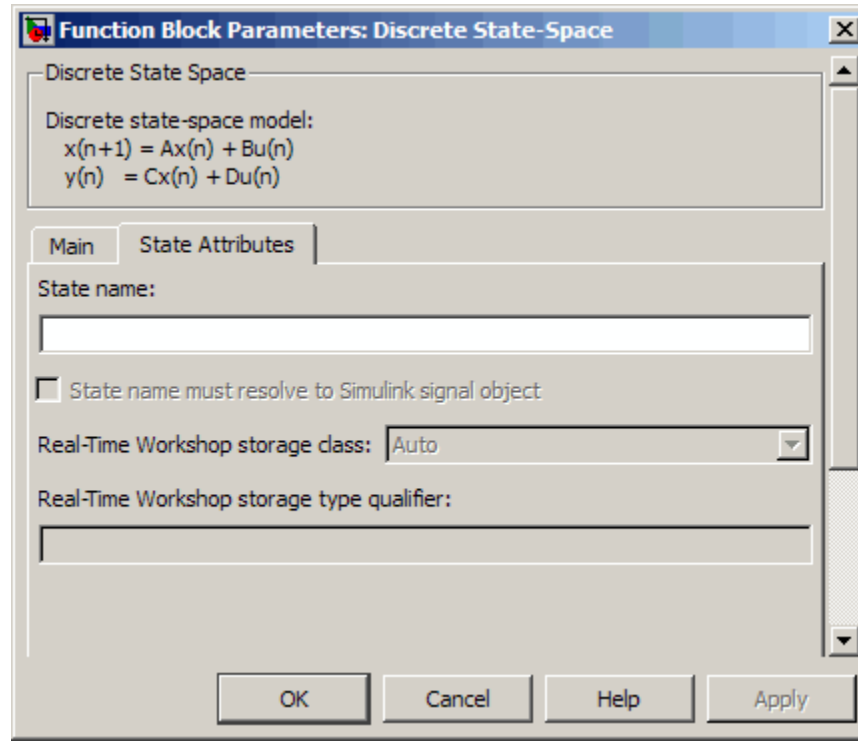
Initial conditions

The initial state vector. The default is `0`. Simulink software 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 the "How Simulink Works" chapter of the Simulink documentation.

The **State Attributes** tab of the Discrete State-Space block dialog box appears as follows:



State name

Use this parameter to assign a unique name to each state. The default is `' '`. If left blank, no name is assigned. Consider the following when using this parameter:

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'` .
- The state names apply only to the selected block.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces. For example, `{'a', 'b', 'c'}` . Each name must be unique.
- The number of states must be evenly divided by the number of state names. You can have fewer names than states, but you cannot have more names than states.

  For example, you can specify two names in a system with four states. Simulink assigns the first name to the first two states and the second name to the last two.
- To assign state names with a variable that has been defined in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell, or structure.

This parameter enables **State name must resolve to Simulink signal object** when you click the **Apply** button.

State name must resolve to Simulink signal object

Select this check box to require that the state name resolve to the Simulink signal object. This check box is cleared by default.

This parameter is enabled by **State name**.

Selecting this check box enables **Real-Time Workshop storage class**.

Real-Time Workshop storage class

From the list, select state storage class.

`Auto`

Auto is the appropriate storage class for states that you do not need to interface to external code.

`ExportedGlobal`
State is stored in a global variable

`ImportedExtern`
model`_private.h` declares the state as an extern variable.

`ImportedExternPointer`
model`_private.h` declares the state as an extern pointer.

This parameter is enabled by **State name**.

Setting this parameter to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables **Real-Time Workshop storage type qualifier**.

During simulation, the block uses the following values:

- The initial value of the signal object to which the state name is resolved
- Min and Max values of the signal object

See "Block State Storage and Interfacing Considerations" 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

Discrete-Time Integrator

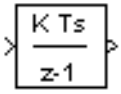
Purpose

Perform discrete-time integration or accumulation of signal

Library

Discrete

Description



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

For the Discrete-Time Integrator block, you can:

- 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 software 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:   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:   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)
  ```

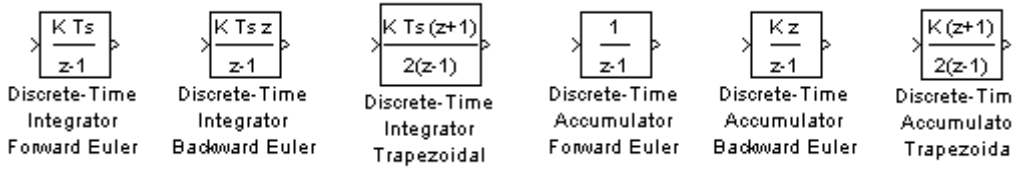
  or, depending on **Use initial condition as initial and reset value for** parameter:

  ```
  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.

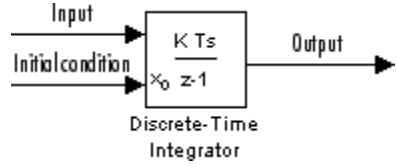
The block reflects the selected integration or accumulation method.



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.



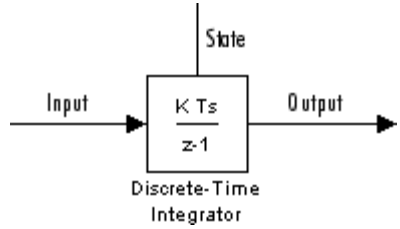
Using the State Port

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 `sldemo_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 `sldemo_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 software 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.

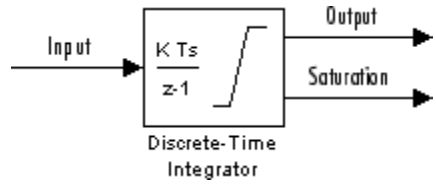


Limiting the Integral

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.

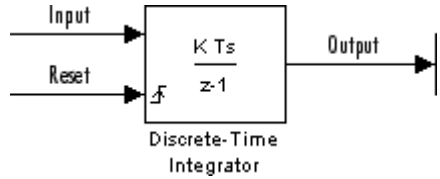


The signal has one of three values:

- 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.



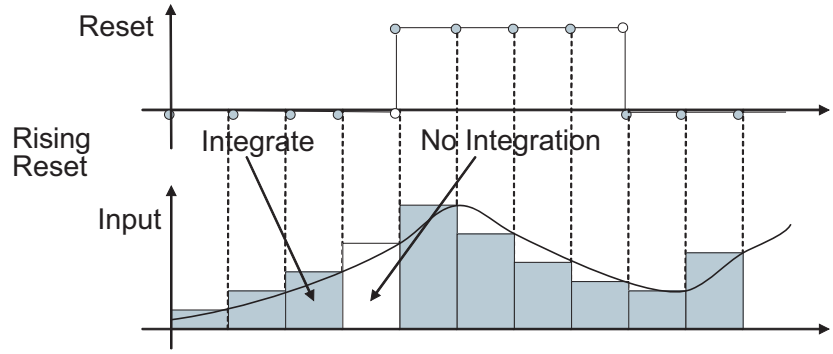
The reset port has direct feedthrough. If the block output is fed back into this port, either directly or through a series of blocks with direct feedthrough, an algebraic loop results. To resolve this loop, feed the output of the block's state port into the reset port instead. To access the block's state,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

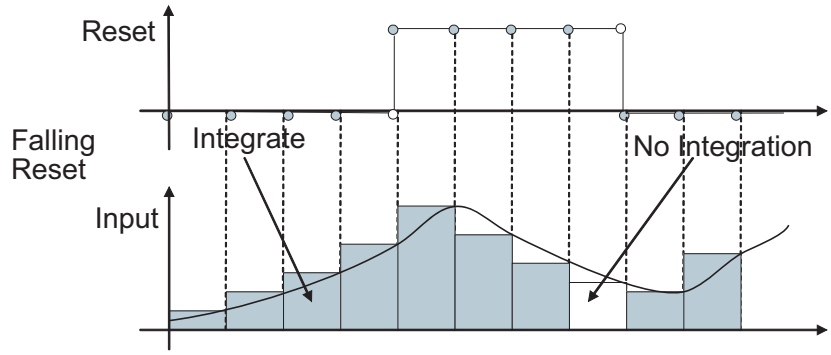
- `rising`

  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.



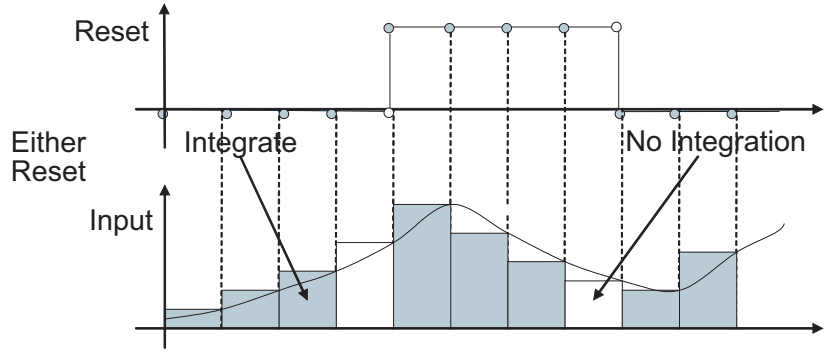
- `falling`

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



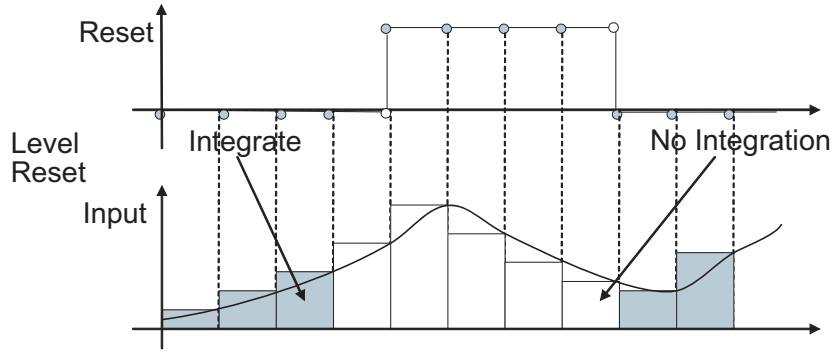
- `either`

  Resets the state when the reset signal rises or falls. For example, the following figure shows the effect that an either reset trigger has on backward Euler integration.



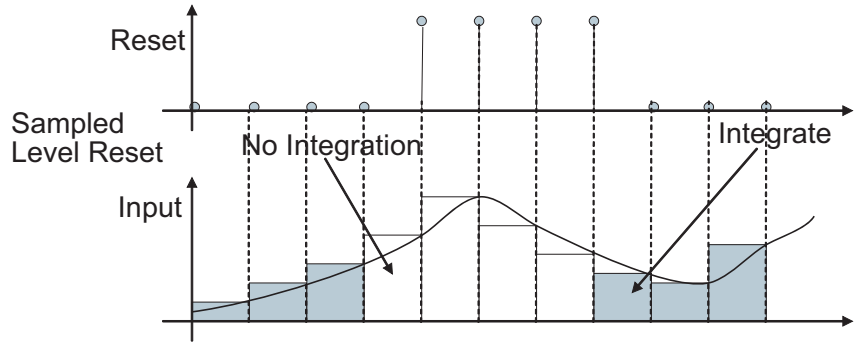
- `level`

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`

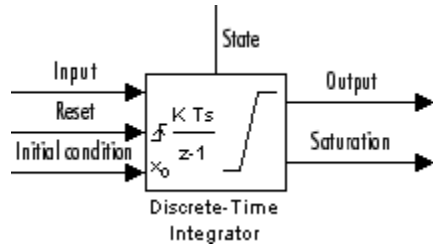
  Resets the output to the initial condition when the reset signal is nonzero. For example, the following figure shows the effect that a sampled level reset trigger has on backward Euler integration.



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

Selecting All Options

When you select all options, the icon looks like this.



Using Simplified Initialization Mode

If you use simplified initialization mode, the behavior of the Discrete-Time Integrator block differs from its behavior in classic initialization mode. The new initialization behavior is more robust, and provides more consistent behavior in these cases:

- In algebraic loops
- On enable and disable
- When comparing results using triggered sample time against results using explicit sample time, where the block is triggered at the same rate as the explicit sample time

In addition, the simplified initialization behavior makes it easier to convert Continuous-Time Integrator blocks to Discrete-Time Integrator blocks, since the initial conditions have the same meaning for both blocks.

For more information on classic and simplified initialization modes, see "Underspecified initialization detection".

Initial Conditions in Simplified Initialization Mode

When you use simplified initialization mode, the **Initial conditions** parameter is applied only to the integrator output.

In addition, the **Use initial condition as initial and reset value for** parameter is disabled. The initial condition is always used as the initial and reset value for the output.

Input-Output Equations in Simplified Initialization Mode

When you use simplified initialization mode, the block starts from first time step `n = 0` with initial output `y(0) = IC` (clipped if necessary).

For a given step `n > 0` with simulation time `t(n)`, Simulink updates output `y(n)` as follows:

- Forward Euler Method:

```
y(n) = y(n-1) + K*[t(n)-t(n-1)]*u(n-1)
```

- Backward Euler Method:

```
y(n) = y(n-1) + K*[t(n)-t(n-1)]*u(n)
```

- Trapezoidal Method:

```
y(n) = y(n-1) + K*[t(n)-t(n-1)]*[u(n)+u(n-1)]/2
```

Simulink automatically selects a state-space realization of these input-output equations depending on the block sample time, which can be explicit or triggered. When using explicit sample time, `t(n)-t(n-1)` reduces to the sample time `T` for all `n > 0`.

Enable and Disable Behavior in Simplified Initialization Mode

When you use simplified initialization mode, the enable and disable behavior of the block is simplified as follows:

At disable time t_d:

$$y(t_d) = y(t_d\text{-}1)$$

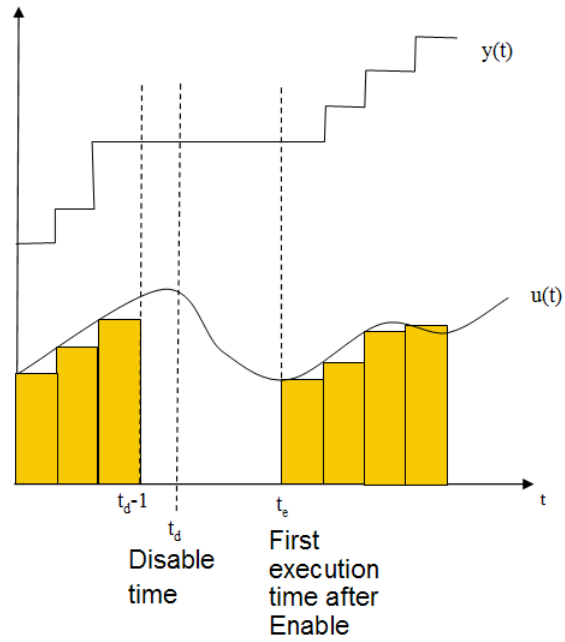
At enable time t_e:

- If parent subsystem resets on enable:

  $y(t_e) = IC$

- In all other cases (see below):

  $y(t_e) = y(t_d)$



Iterator Subsystems

When using simplified initialization mode, you cannot place the Discrete-Time Integrator block within an Iterator Subsystem.

In simplified initialization mode, Iterator subsystems do not maintain elapsed time, so Simulink reports an error if any block needing elapsed

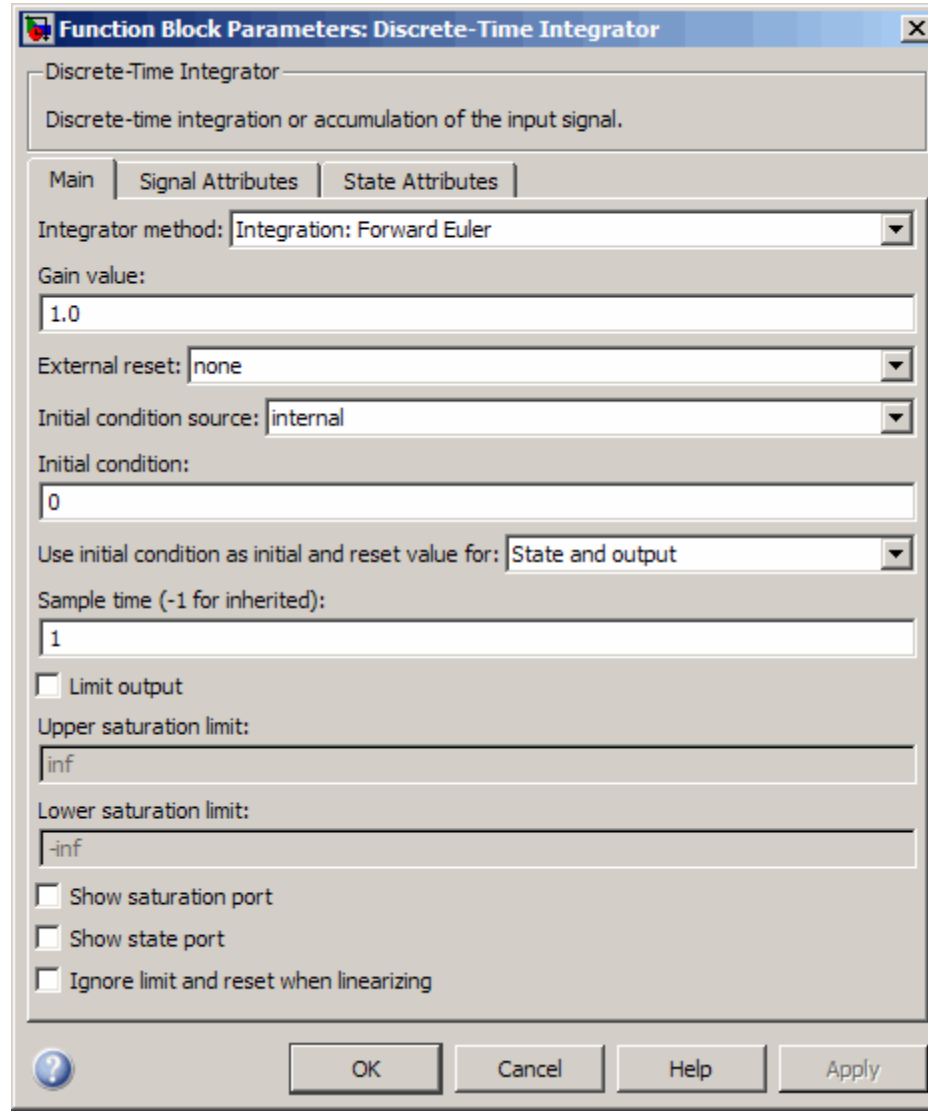
time, such as the Discrete-Time Integrator, is placed inside an Iterator Subsystem.

Data Type Support

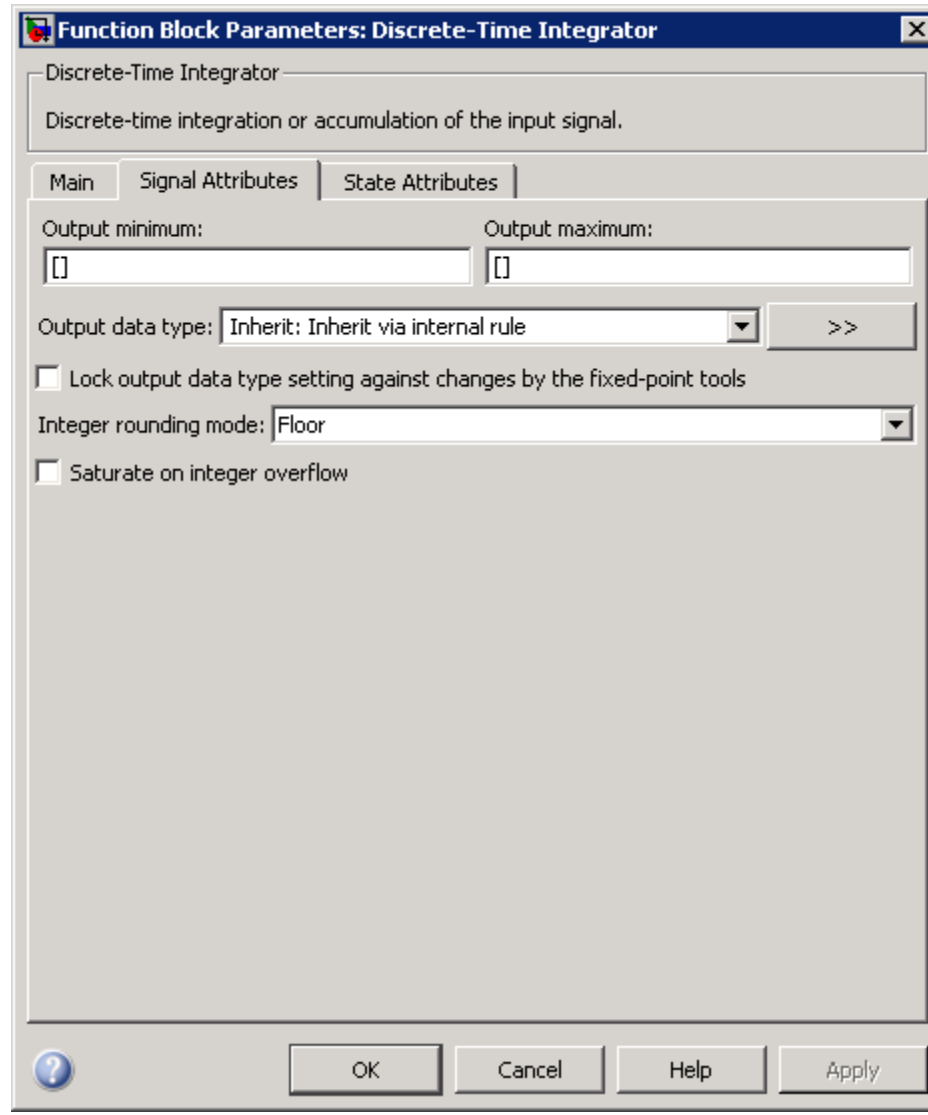
The Discrete-Time Integrator block accepts real signals of any numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box

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



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



During simulation, the block uses the following values:

- The initial value of the signal object to which the state name is resolved
- Min and Max values of the signal object

See “Block State Storage and Interfacing Considerations” in the Real-Time Workshop User’s Guide for more information.

Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Integrator method

Specify the integration or accumulation method.

Settings

Default: `Integration:  Forward Euler`

`Integration:  Forward Euler`
Integrator method is Forward Euler.

`Integration:  Backward Euler`
Integrator method is Backward Euler.

`Integration:  Trapezoidal`
Integrator method is Trapezoidal.

`Accumulation:  Forward Euler`
Accumulation method is Forward Euler.

`Accumulation:  Backward Euler`
Accumulation method is Backward Euler.

`Accumulation:  Trapezoidal`
Accumulation method is Trapezoidal.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

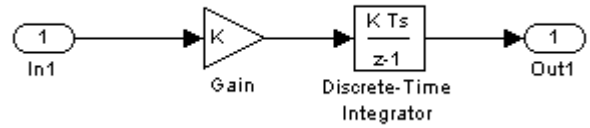
Gain value

Specify a scalar, vector, or matrix by which to multiply the integrator input. Each element of the gain must be a positive real number.

Settings

Default: `1.0`

- Specifying a value other than 1.0 (the default) is semantically equivalent to connecting a signal to the input of the integrator using a Gain block.



- Valid entries include:
  - `double(1.0)`
  - `single(1.0)`
  - `[1.1 2.2 3.3 4.4]`
  - `[1.1 2.2; 3.3 4.4]`
- 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 software 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.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

External reset

Reset the states to their initial conditions when a trigger event occurs in the reset signal.

Settings

Default: `none`

`none`
: Do not reset the state to initial conditions.

`rising`
: Reset the state when the reset signal has a rising edge.

`falling`
: Reset the state when the reset signal has a falling edge.

`either`
: Reset the state when the reset signal rises or falls.

`level`
: Reset and holds the output to the initial condition while the reset signal is nonzero.

`sampled level`
: Reset the output to the initial condition when the reset signal is nonzero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Initial condition source

Get the initial conditions of the states.

Settings

Default: `internal`

`internal`
: Get the initial conditions of the states from the **Initial condition** parameter.

`external`
: Get the initial conditions of the states from an external block.

Tips

Simulink software does not allow the initial condition of this block to be `inf` or `NaN`.

Dependencies

Selecting `internal` enables the **Initial condition** parameter.

Selecting `external` disables the **Initial condition** parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Initial condition

Specify the states' initial conditions.

Settings

Default: `0`

Minimum: value of **Output minimum** parameter

Maximum: value of **Output maximum** parameter

Tips

Simulink software does not allow the initial condition of this block to be `inf` or `NaN`.

Dependencies

Setting **Initial condition source** to `internal` enables this parameter.

Setting **Initial condition source** to `external` disables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Use initial condition as initial and reset value for

Specify whether to apply the initial condition as the initial and reset value for the state and output, or the state only.

Note If you are using simplified initialization mode, this parameter is disabled. The initial condition is always used as the initial and reset value for the output. For more information, see "Underspecified initialization detection".

Settings

Default: `State and output`

`State and output`
: Set the following for initial

  `y(0) = IC`

  `x(0) = IC`

  or at reset

  `y(n) = IC`

  `x(n) = IC`

`State only (most efficient)`
: Set the following for initial

  `x(0) = IC`

  or at reset

  `x(n) = IC`

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits.

Settings

Default: 1

By default, the block uses a discrete sample time of 1. To set a different sample time, enter another discrete value, such as 0.1.

See also "How to Specify the Sample Time" in the online documentation for more information.

Tips

- Do not specify a sample time of 0. This value specifies a continuous sample time, which the Discrete-Time Integrator block does not support.
- Do not specify a sample time of `inf` or `NaN` because these values are not discrete.
- If you specify -1 to inherit the sample time from an upstream block, verify that the upstream block uses a discrete sample time. For example, the Discrete-Time Integrator block cannot inherit a sample time of 0.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Limit output

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

Settings

Default: Off

 On

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

 Off

Do not limit the block's output to a value between the **Lower saturation limit** and **Upper saturation limit** parameters.

Dependencies

This parameter enables **Upper saturation limit**.

This parameter enables **Lower saturation limit**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Upper saturation limit

Specify the upper limit for the integral.

Settings

Default: `inf`

Minimum: value of **Output minimum** parameter

Maximum: value of **Output maximum** parameter

Dependencies

Limit output enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Lower saturation limit

Specify the lower limit for the integral.

Settings

Default: `-inf`

Minimum: value of **Output minimum** parameter

Maximum: value of **Output maximum** parameter

Dependencies

Limit output enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Show saturation port

Add a saturation output port to the block.

Settings

Default: Off

 On

Add a saturation output port to the block.

 Off

Do not add a saturation output port to the block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Show state port

Add an output port to the block for the block's state.

Settings

Default: Off

 On

Add an output port to the block for the block's state.

 Off

Do not add an output port to the block for the block's state.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Ignore limit and reset when linearizing

Cause Simulink linearization commands to treat this block as not resettable and as having no limits on its output, regardless of the settings of the block reset and output limitation options.

Settings

Default: Off

 On

Cause Simulink linearization commands to treat this block as not resettable and as having no limits on its output, regardless of the settings of the block reset and output limitation options.

 Off

Do not cause Simulink linearization commands to treat this block as not resettable and as having no limits on its output, regardless of the settings of the block reset and output limitation options.

Tips

Ignoring the limit and resetting allows you to linearize a model around an operating point. This point may cause the integrator to reset or saturate.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Floor`

`Ceiling`
: Round both positive and negative numbers toward positive infinity.

`Convergent`
: Round number to the nearest representable value. If a tie occurs, round to the nearest even stored value.

`Floor`
: Round both positive and negative numbers toward negative infinity.

`Nearest`
: Round number to the nearest representable value. If a tie occurs, round toward positive infinity.

`Round`
: Round number to the nearest representable value. If a tie occurs, round positive numbers toward positive infinity and round negative numbers toward negative infinity.

`Simplest`
: This option provides for an optimization of the rounding code for several blocks.

`Zero`
: Round number toward zero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Rounding” in the *Simulink Fixed Point User’s Guide*.

Saturate on integer overflow

Specify whether overflows saturate.

Settings

Default: Off

 On

Overflows saturate.

 Off

Overflows do not saturate.

Tips

- When you select this check box, saturation applies to every internal operation on the block, not just the output or result.
- In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

State name

Use this parameter to assign a unique name to each state.

Settings

Default: `' '`

- If left blank, no name is assigned.

Tips

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'` .
- The state names apply only to the selected block.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces. For example, `{'a', 'b', 'c'}` . Each name must be unique.
- The number of states must be evenly divided by the number of state names. There can be fewer names than states, but there cannot be more names than states.
- For example, you can specify two names in a system with four states. Simulink software will assign the first name to the first two states and the second name to the last two.
- To assign state names with a variable that has been defined in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell, or structure.

Dependencies

This parameter enables **State name must resolve to Simulink signal object** when you click the **Apply** button.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

State name must resolve to Simulink signal object

Require that state name resolve to Simulink signal object.

Settings

Default: Off

 On

Require that state name resolve to Simulink signal object.

 Off

Do not require that state name resolve to Simulink signal object.

Dependencies

State name enables this parameter.

This parameter enables **Real-Time Workshop storage class**.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Real-Time Workshop storage class

Select state storage class.

Settings

Default: `Auto`

`Auto`
: Auto is the appropriate storage class for states that you do not need to interface to external code.

`ExportedGlobal`
: State is stored in a global variable

`ImportedExtern`
: *`model`*`_private.h` declares the state as an extern variable.

`ImportedExternPointer`
: *`model`*`_private.h` declares the state as an extern pointer.

Dependencies

State name enables this parameter.

Setting this parameter to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables **Real-Time Workshop storage type qualifier**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Block State Storage Classes" in the *Real-Time Workshop User's Guide*.

Real-Time Workshop storage type qualifier

Specify Real-Time storage type qualifier.

Settings

Default: `' '`

If left blank, no qualifier is assigned.

Dependencies

Setting **Real-Time Workshop storage class** to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `Inherit: Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block), `Inherit: Inherit from 'Constant value'` (Constant block), `Inherit: Inherit via back propagation` (Data Type Conversion block), `Inherit: Same as input` (Saturation block)

`Inherit: Inherit via internal rule`
: This option appears for some blocks. Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the calculated 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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 software displays an error message in the Simulation Diagnostics Viewer.

`Inherit: Inherit from 'Constant value'`
: Use data type of **Constant value** parameter. This option appears for some blocks.

`Inherit: Inherit via back propagation`
: Use data type of the driving block.

`Inherit: Same as input`
: Use data type of sole input signal. This option appears for some blocks.

`Inherit: Same as first input`
: Use data type of first input signal. This option appears for some blocks.

`Inherit: Same as accumulator`
: Output data type is the same as accumulator data type. This option appears for some blocks.

`double`
: Output data type is `double`.

`single`
: Output data type is `single`.

`int8`
: Output data type is `int8`.

`uint8`
: Output data type is `uint8`.

`int16`
: Output data type is `int16`.

`uint16`
: Output data type is `uint16`.

`int32`
: Output data type is `int32`.

`uint32`
: Output data type is `unt32`.

`fixdt(1,16,0)`
: Output data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Output data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
: Use an enumerated data type, for example, `Enum: BasicColors`. This option appears for some blocks.

`<data type expression>`
: Use a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<class name>`

`Expression`
: Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink 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 Detection	No

See Also

Integrator

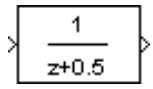
Purpose

Implement discrete transfer function

Library

Discrete

Description



The Discrete Transfer Fcn block implements the *z*-transform transfer function:

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

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 you specify both as parameters on the block dialog box. The order of the denominator must be greater than or equal to the order of the numerator.

Specify the coefficients of the numerator and denominator polynomials in descending powers of *z*. The Discrete Filter block lets you use polynomials in z^{-1} (the delay operator) to represent a discrete system, a method that signal processing engineers typically use. Conversely, the Discrete Transfer Fcn block lets you use polynomials in *z* to represent a discrete system, the method that control engineers typically use. The two methods are identical when the numerator and denominator polynomials have the same length.

Specifying Initial States

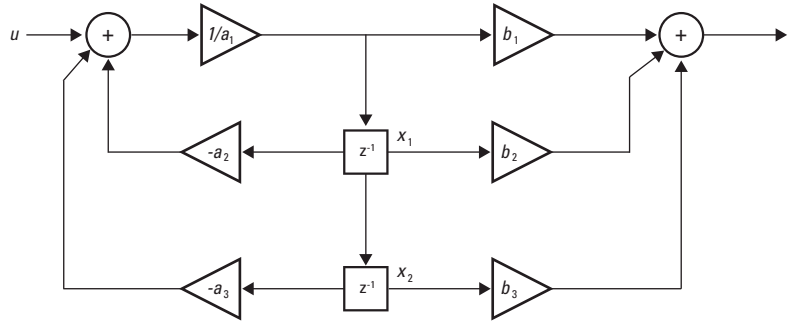
Use the **Initial states** parameter to specify initial filter states. To determine the number of initial states you must specify and how to specify them, see the following table of valid initial state dimensions. The **Initial states** parameter can take one of the forms described in this table.

Valid Initial States

Initial States	Description
Scalar	The block initializes all filter states to the same scalar value. Enter `0` to initialize all states to zero.
Vector or matrix	Each vector or matrix element specifies a unique initial state for a corresponding delay element in a corresponding channel: • The vector length must equal the number of delay elements in the filter, `max(number of zeros, number of poles)`. • The matrix must have the same number of rows as the number of delay elements in the filter, `max(number of zeros, number of poles)`. The matrix must also have one column for each channel of the input signal.

The following example shows the relationship between the initial filter output and the initial input and state. Given an initial input u_1, the first output y_1 is related to the initial state $[x_1, x_2]$ and initial input by:

$$y_1 = b_1\left[\frac{(u_1 - a_2x_1 - a_3x_2)}{a_1}\right] + b_2x_1 + b_3x_2$$



Initial States: 1

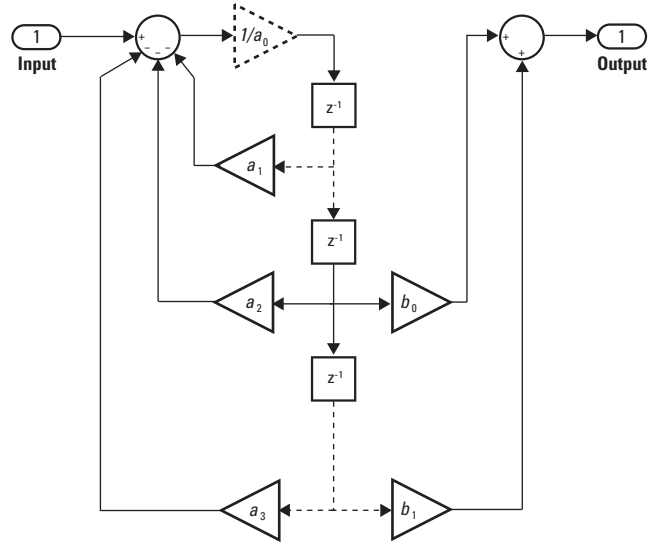
Data Type Support

The Discrete Transfer Function block accepts and outputs real and complex signals of any signed numeric data type that Simulink supports. The block supports the same types for the numerator and denominator coefficients.

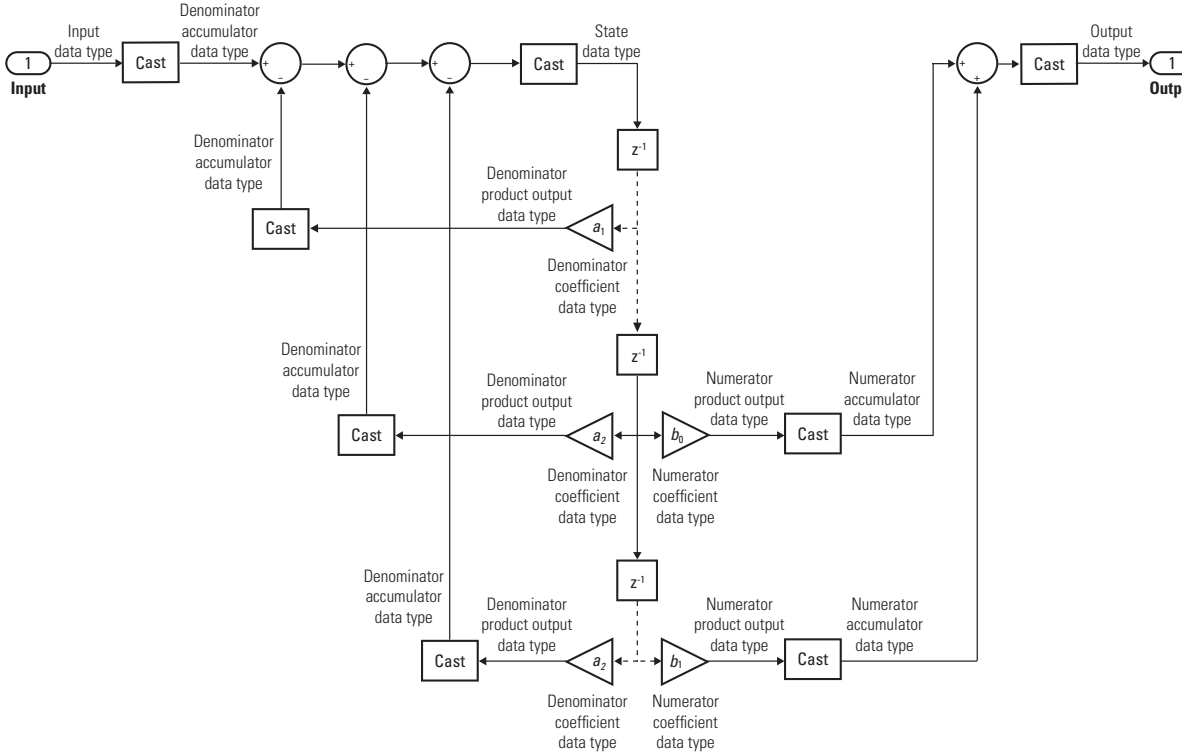
Numerator and denominator coefficients must have the same complexity. They can have different word lengths and fraction lengths.

States are complex when either the input or the coefficients are complex.

The following diagrams show the filter structure and the data types that the block uses for floating-point and fixed-point signals.

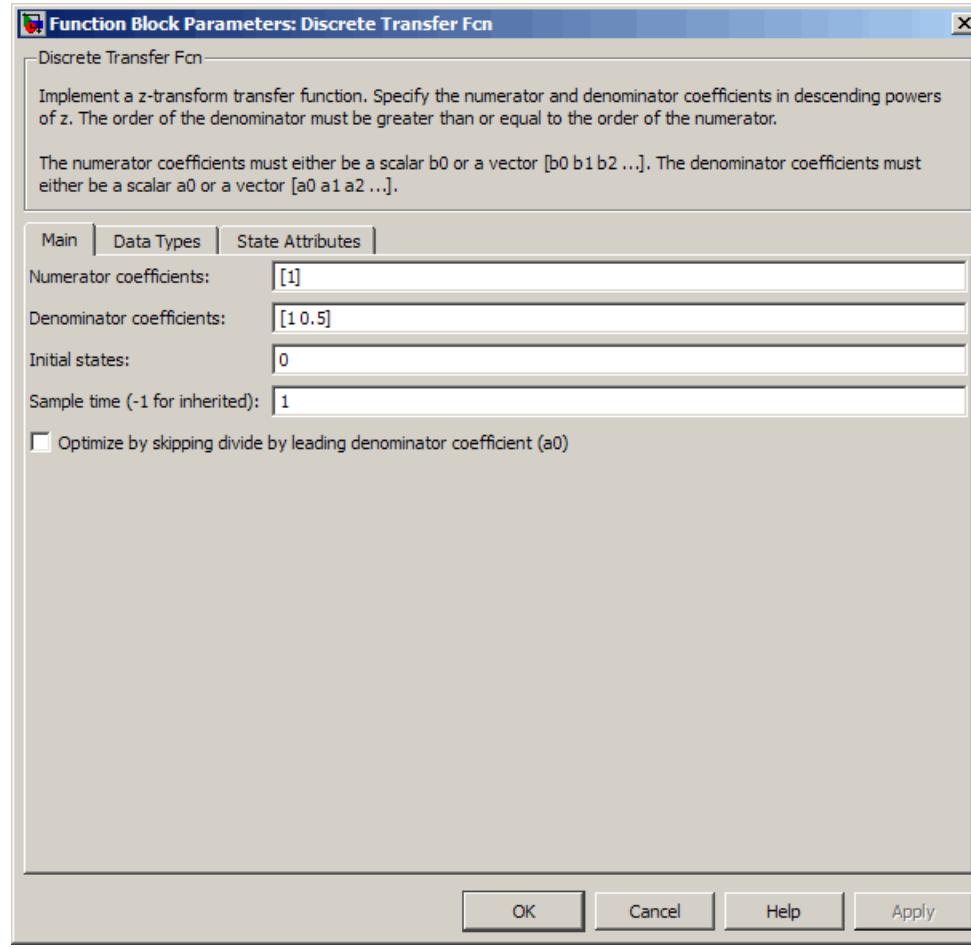


The block omits the dashed divide when you select the **Optimize by skipping divide by leading denominator coefficient (a0)** parameter.


1
Input
Input data type
Cast
Denominator accumulator data type
State data type
Cast
Output data type
Cast
1
Output
z-1
Denominator accumulator data type
Cast
Denominator product output data type
a1
Denominator coefficient data type
z-1
Denominator accumulator data type
Cast
Denominator product output data type
a2
b0
Numerator product output data type
Cast
Numerator accumulator data type
Denominator coefficient data type
Numerator coefficient data type
z-1
Denominator accumulator data type
Cast
Denominator product output data type
a2
b1
Numerator product output data type
Cast
Numerator accumulator data type
Denominator coefficient data type
Numerator coefficient data type

Parameters and Dialog Box

The **Main** pane of the Discrete Transfer Fcn block dialog box appears as follows.



Numerator coefficients

Specify the coefficients of the discrete filter numerator polynomial or polynomials in descending powers of z. Use a row vector to specify the coefficients for a single numerator polynomial.

Denominator coefficients

Specify the coefficients of the discrete filter denominator polynomial as a row vector in descending powers of z.

Initial states

Specify the initial states of the filter states. To learn how to specify initial states, see "Specifying Initial States" on page 2-375.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in "How Simulink Works" in the *Simulink User's Guide*.

Optimize by skipping divide by leading denominator coefficient (a0)

Select when the leading denominator coefficient, a_0, equals one. This parameter generates optimized code.

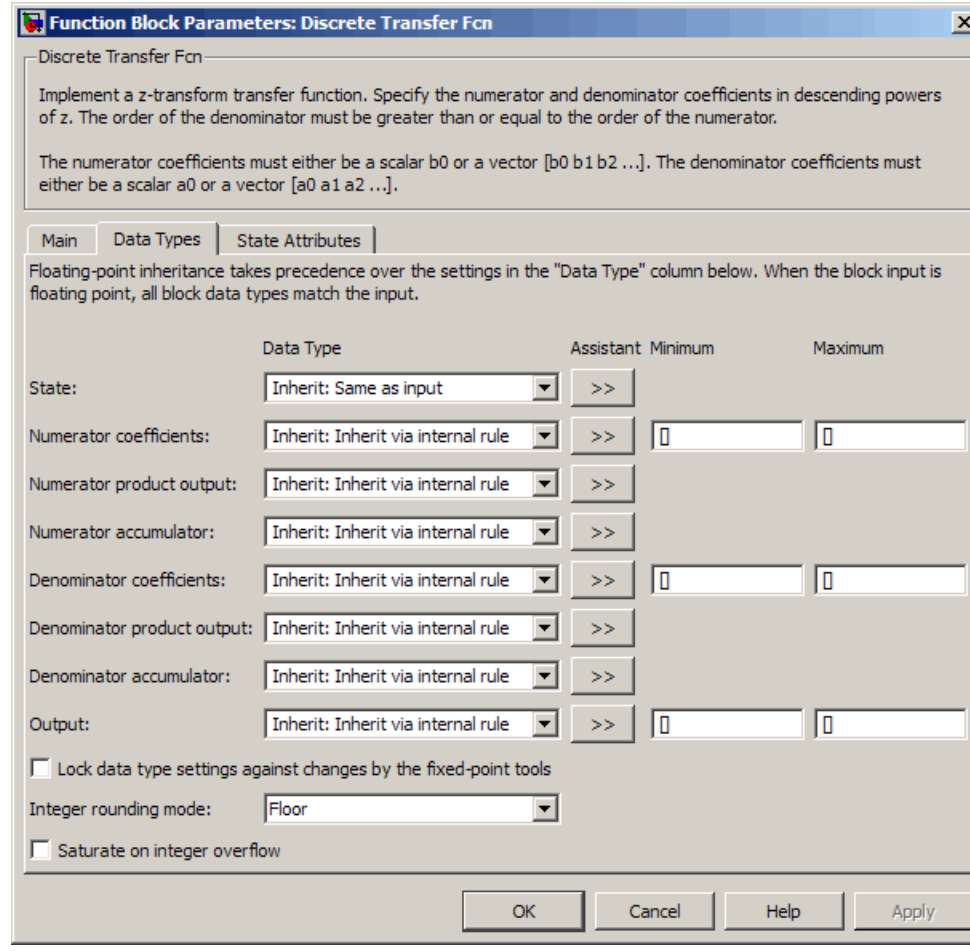
When you select this check box, the block:

- Does not perform a divide-by-a_0 either in simulation or in the generated code
- Errors out at model edit time if the a_0 value you provide in the dialog is not one
- Errors out if you tune a_0 to any nonunity value

When you clear this check box, the block:

- Is fully tunable during simulation
- Performs a divide-by-a_0 in both simulation and code generation

The **Data Types** pane of the Discrete Transfer Fcn block dialog box appears as follows.



State

Specify the state data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- A built-in integer, for example, `int8`

- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **State** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Numerator coefficients

Specify the numerator coefficient data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in integer, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Numerator coefficients** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Numerator coefficient minimum

Specify the minimum value that a numerator coefficient can have. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")

- Automatic scaling of fixed-point data types

Numerator coefficient maximum

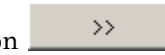
Specify the maximum value that a numerator coefficient can have. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Automatic scaling of fixed-point data types

Numerator product output

Specify the product output data type for the numerator coefficients. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Numerator product output** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Numerator accumulator

Specify the accumulator data type for the numerator coefficients. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`

- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Numerator accumulator** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Denominator coefficients

Specify the denominator coefficient data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in integer, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Denominator coefficients** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Denominator coefficient minimum

Specify the minimum value that a denominator coefficient can have. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")

- Automatic scaling of fixed-point data types

Denominator coefficient maximum

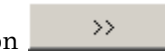
Specify the maximum value that a denominator coefficient can have. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Automatic scaling of fixed-point data types

Denominator product output

Specify the product output data type for the denominator coefficients. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Denominator product output** parameter.

See "Using the Data Type Assistant" in the*Simulink User's Guide* for more information.

Denominator accumulator

Specify the accumulator data type for the denominator coefficients. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`

- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Denominator accumulator** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Output

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- A built-in data type, for example, `int8`
- A data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output** parameter.

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Output minimum

Specify the minimum value that the block can output. The default value, `[]`, is equivalent to `-Inf`. Simulink uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")

- Automatic scaling of fixed-point data types

Output maximum
Specify the maximum value that the block can output. The default value, `[]`, is equivalent to `Inf`. Simulink uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

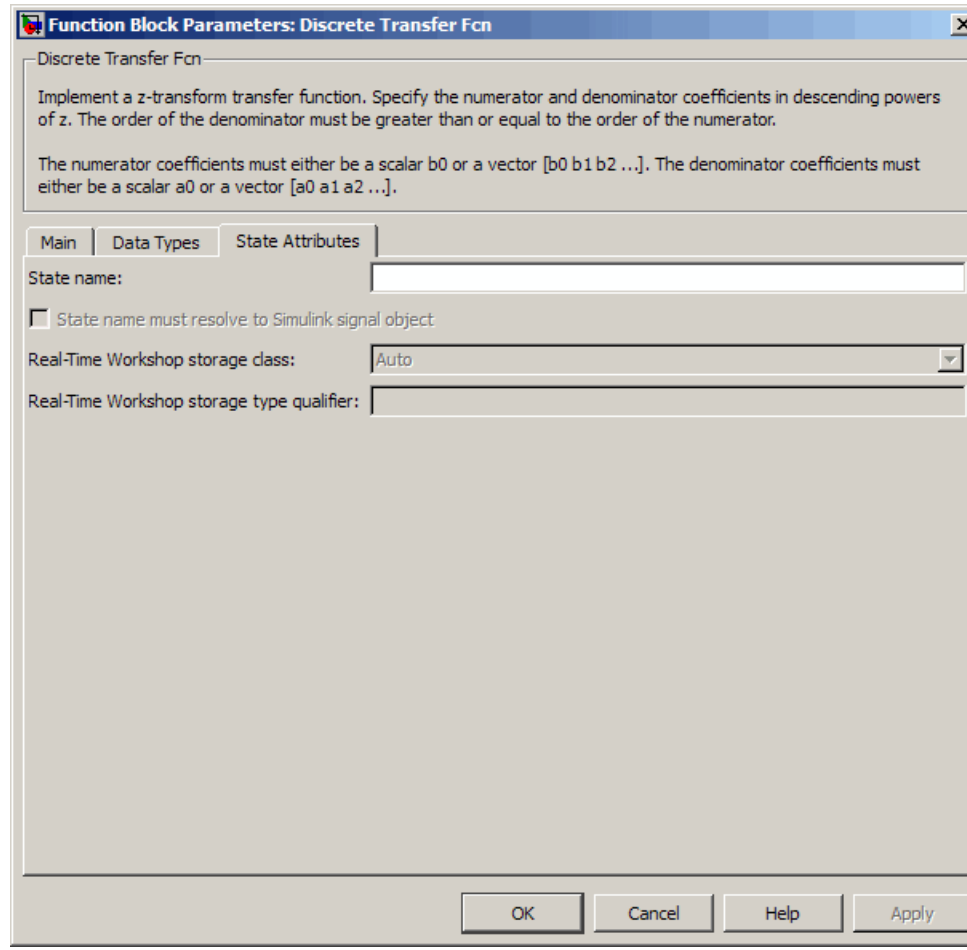
Lock data type settings against changes by the fixed-point tools
Select to lock all data type settings of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate on integer overflow
Select this check box to have overflows saturate. Otherwise, they wrap.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

The **State Attributes** pane of the Discrete Filter block dialog box appears as follows.



State name

Use this parameter to assign a unique name to each state. The default is `' '`. When this field is blank, no name is assigned. Consider the following when using this parameter:

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'`.

- The state names apply only to the selected block.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, `{'a', 'b', 'c'}`. Each name must be unique.
- The number of states must be an integer multiple of the number of state names. You can have fewer names than states, but you cannot have more names than states.
- For example, you can specify two names in a system with four states. Simulink software assigns the first name to the first two states and the second name to the last two.
- To assign state names with a variable that you have defined in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell, or structure.

This parameter enables **State name must resolve to Simulink signal object** when you click **Apply**.

State name must resolve to Simulink signal object

Select this check box to require that the state name resolves to a Simulink signal object. This check box is cleared by default.

Specifying the **State name** parameter enables this parameter.

Selecting this check box enables **Real-Time Workshop storage class**.

Real-Time Workshop storage class

From the list, select the state storage class.

`Auto`

If you do not need states to interface to external code, select `Auto` as the storage class.

`ExportedGlobal`

The class stores the state in a global variable.

`ImportedExtern`
: *model*`_private.h` declares the state as an extern variable.

`ImportedExternPointer`
: *model*`_private.h` declares the state as an extern pointer.

  Specifying the **State name** parameter enables this parameter.

  Setting this parameter to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables **Real-Time Workshop storage type qualifier**.

During simulation, the block uses the following values:

- The initial value of the signal object to which the state name resolves
- Minimum and maximum values of the signal object

See "Block State Storage and Interfacing Considerations" in the *Real-Time Workshop User's Guide* for more information.

Characteristics

Direct Feedthrough	Only when the leading numerator coefficient is not equal to zero and the numerator order equals the denominator order
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of initial states
States	See "Specifying Initial States" on page 2-375
Dimensionalized	Yes
Zero-Crossing Detection	No

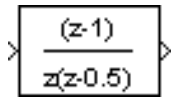
Purpose

Model system defined by zeros and poles of discrete transfer function

Library

Discrete

Description



The Discrete Zero-Pole block models a discrete system defined by the zeros, poles, and gain of a *z*-domain transfer function. This block assumes that the transfer function has the following form

$$H(z) = K\frac{Z(z)}{P(z)} = K\frac{(z-Z_1)(z-Z_2)\ldots(z-Z_m)}{(z-P_1)(z-P_2)\ldots(z-P_n)}$$

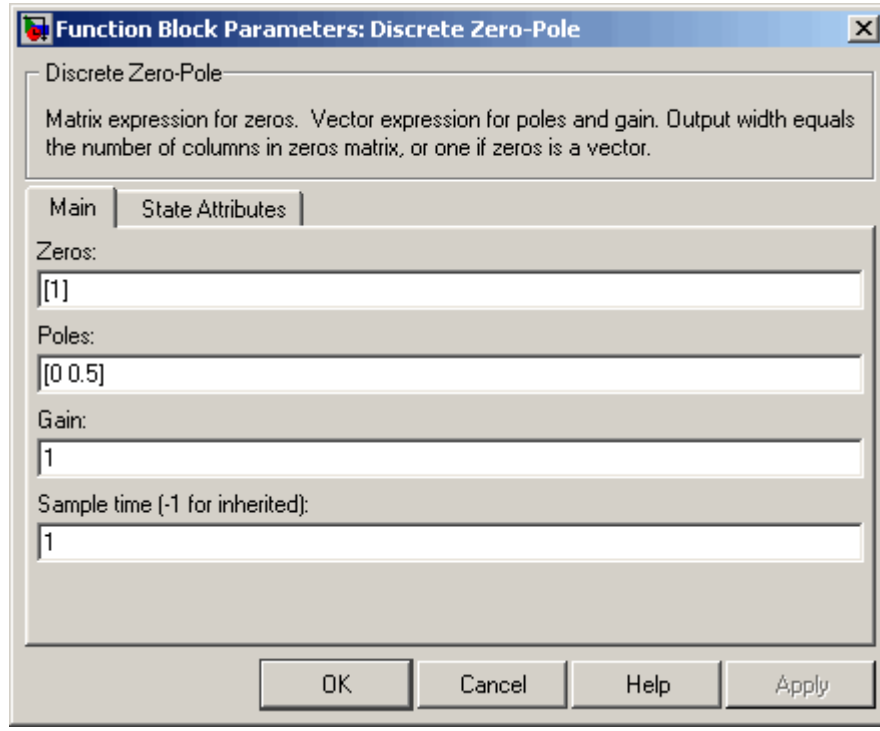
where Z represents the zeros vector, P the poles vector, and K the gain. The number of poles must be greater than or equal to the number of zeros ($n \geq m$). If the poles and zeros are complex, they must be complex conjugate pairs.

The block displays the transfer function depending on how the parameters are specified. See Zero-Pole for more information.

Data Type Support

The Discrete Zero-Pole block accepts and outputs real signals of type `double`.

Parameters and Dialog Box



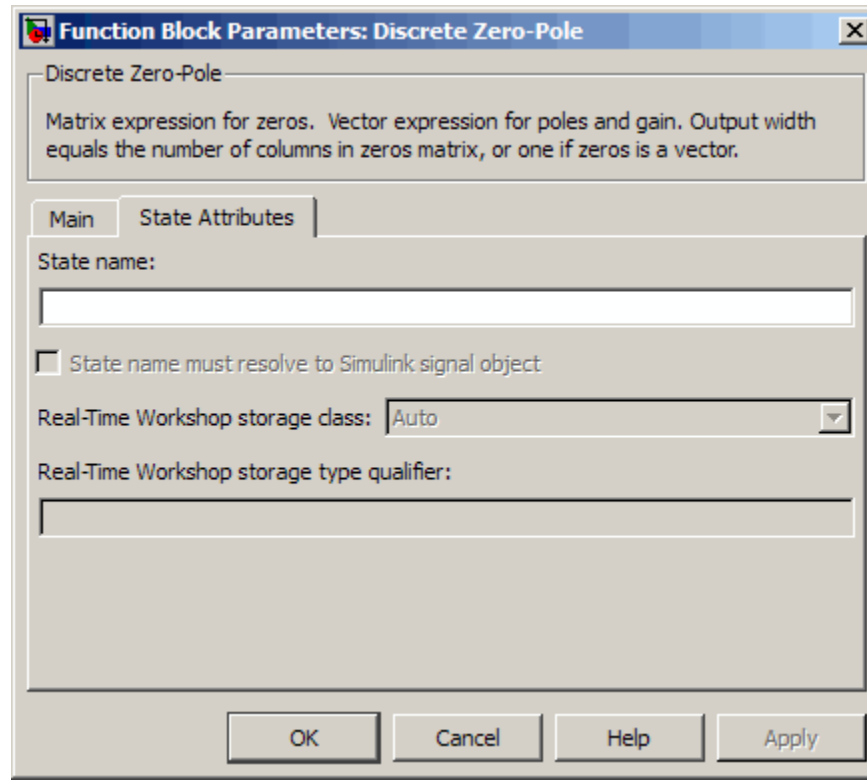
Zeros
The matrix of zeros. The default is `[1]`.

Poles
The vector of poles. The default is `[0 0.5]`.

Gain
The gain. The default is `1`.

Sample time
The time interval between samples. See Specifying Sample Time in the "How Simulink Works" chapter of the Simulink documentation.

The **State Attributes** pane of the Discrete Zero-Pole block dialog box appears as follows:



State name

Use this parameter to assign a unique name to each state. The default is `' '`. If left blank, no name is assigned. Consider the following when using this parameter:

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'` .
- The state names apply only to the selected block.

- To assign names to multiple states, enter a comma-delimited list surrounded by braces. For example, `{'a', 'b', 'c'}`. Each name must be unique.
- The number of states must be evenly divided by the number of state names. There can be fewer names than states, but there cannot be more names than states.
- For example, you can specify two names in a system with four states. Simulink software will assign the first name to the first two states and the second name to the last two.
- To assign state names with a variable that has been defined in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell, or structure.

This parameter enables **State name must resolve to Simulink signal object** when you click the **Apply** button.

State name must resolve to Simulink signal object

Select this checkbox to require that state name resolve to Simulink signal object. This check box is cleared by default.

This parameter is enabled by **State name**.

Selecting this check box enables **Real-Time Workshop storage class**.

Real-Time Workshop storage class

From the list, select state storage class.

`Auto`
: Auto is the appropriate storage class for states that you do not need to interface to external code.

`ExportedGlobal`
: State is stored in a global variable

`ImportedExtern`
: *`model`*`_private.h` declares the state as an extern variable.

`ImportedExternPointer`
model`_private.h` declares the state as an extern pointer.

This parameter is enabled by **State name**.

Setting this parameter to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables **Real-Time Workshop storage type qualifier**.

During simulation, the block uses the following values:

- The initial value of the signal object to which the state name is resolved
- Min and Max values of the signal object

See "Block State Storage and Interfacing Considerations" in the Real-Time Workshop User's Guide for more information.

Characteristics

Direct Feedthrough	Yes, if the number of zeros and poles are equal
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
States	Length of **Poles** vector
Dimensionalized	No
Zero Crossing	No

Display

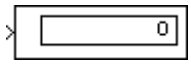
Purpose

Show value of input

Library

Sinks

Description



The Display block shows the value of its input on its icon.

You control the display format using the **Format** parameter:

- `short` — displays a 5-digit scaled value with fixed decimal point
- `long` — displays a 15-digit scaled value with fixed decimal point
- `short_e` — displays a 5-digit value with a floating decimal point
- `long_e` — displays a 16-digit value with a floating decimal point
- `bank` — displays a value in fixed dollars and cents format (but with no $ or commas)
- `hex (Stored Integer)` — displays the stored integer value of a fixed-point input in hexadecimal format
- `binary (Stored Integer)` — displays the stored integer value of a fixed-point input in binary format
- `decimal (Stored Integer)` — displays the stored integer value of a fixed-point input in decimal format
- `octal (Stored Integer)` — displays the stored integer value of a fixed-point input in octal format

If the signal input to a Display block has an enumerated data type (see "Using Enumerated Data"):

- The block displays enumerated values, not the values' underlying integers.
- Setting the **Format** to any of the `Stored Integer` settings causes an error.

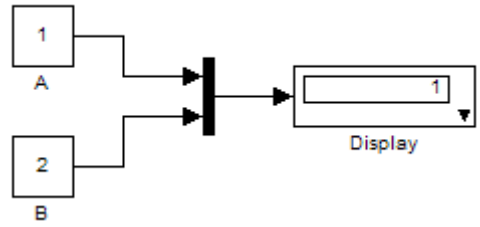
The amount of data displayed and the time steps at which the data is displayed are determined by the **Decimation** block parameter and the `SampleTime` property:

- The **Decimation** parameter enables you to display data at every `n`th sample, where `n` is the decimation factor. The default decimation, `1`, displays data at every time step.

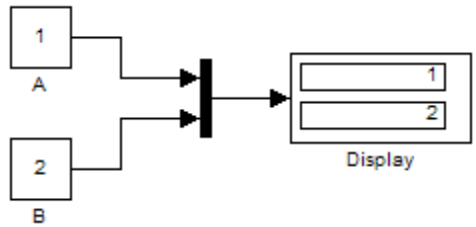
  Note The Display block updates its display at the initial time, even when the **Decimation** value is greater than one.

- The `SampleTime` property, settable with `set_param`, enables you to specify a sampling interval at which to display points. This property 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 ignore the sampling interval when determining the points to display.

If the block input is an array, you can resize the block to show more than just the first element. You can resize the block vertically or horizontally; the block adds display fields in the appropriate direction. A black triangle indicates that the block is not displaying all input array elements. For example, the following figure shows a model that passes a vector (1-D array) to a Display block. The black triangle on the Display block indicates more data to be displayed.



The following figure shows the resized block displaying both input elements.



Note The Display block shows only the first 200 elements of a one-dimensional (vector) signal and only the first 20 rows and 10 columns of a two-dimensional (matrix) signal.

Display Abbreviations

The following abbreviations appear on the Display block to help you identify the format of the number being displayed.

Symbol	Description
`(SI)`	This alerts you to the fact that the number being displayed is the stored integer value. This symbol does not appear when the signal is of an integer data type.
`hex`	The number being displayed is in hexadecimal format.
`bin`	The number being displayed is in binary format.
`oct`	The number being displayed is in octal format.

Floating Display

To use the block as a floating display, select the **Floating display** check box. The block's input port disappears and the block displays

the value of the signal on a selected line. If you select the **Floating display** option, you must turn off the signal storage reuse feature in your Simulink software. See "Signal storage reuse" in the "Running Simulations" chapter of the Simulink documentation.

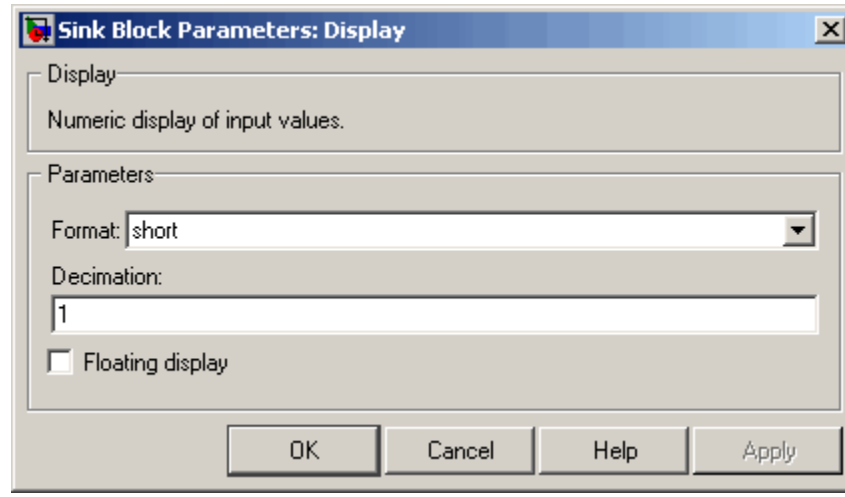
Note The floating display does not support multidimensional signals. If you connect a multidimensional signal to a floating display, the display generates an error.

Data Type Support

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

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

Parameters and Dialog Box



Format
Specify the format of the data displayed, as discussed in Description. The default is `short`.

Decimation
Specify how often to display data. The default value, `1`, displays every input point.

Floating display
If selected, the block's input port disappears, which enables the block to be used as a floating Display block.

Characteristics

SampleTime	Use `set_param` to specify the `SampleTime` property
Dimensionalized	Yes

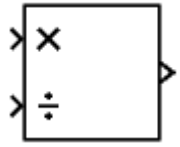
Purpose

Divide one input by another

Library

Math Operations

Description



The Divide block outputs the result of dividing its first input by its second. The inputs can be scalars, a scalar and a nonscalar, or two nonscalars that have the same dimensions. The Divide block is functionally a Product block that has two block parameter values preset:

- **Multiplication:** `Element-wise(.*)`
- **Number of Inputs:** `*/`

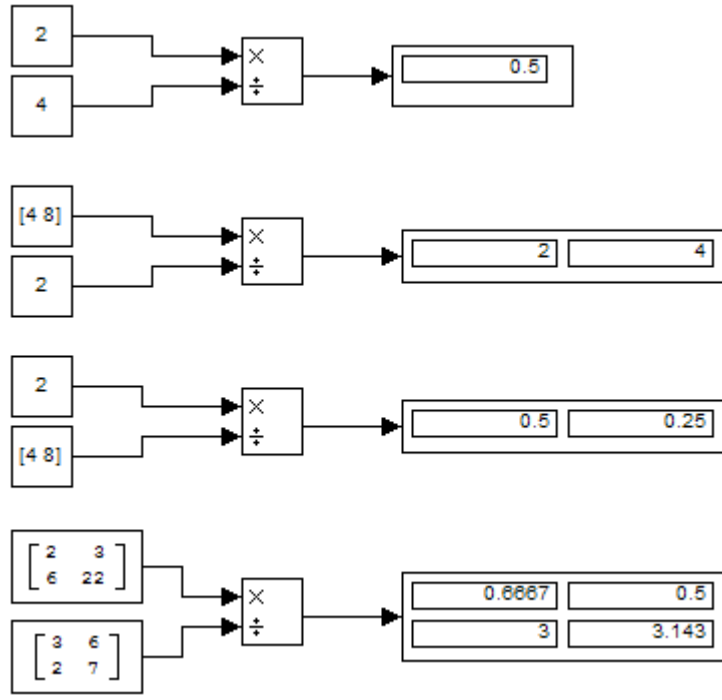
Setting non-default values for either of those parameters can change a Divide block to be functionally equivalent to a Product block or a Product of Elements block. See the documentation of those two blocks for more information.

Parameters and Dialog Box

The Divide block has the same parameters and dialog box as the Product block. If all you need is to divide two inputs to create an output, you can use the Divide block with default parameter values. If you need additional capabilities, see the Product block documentation, which also describes the capabilities of the Divide block's "Signal Attributes Pane" on page 2-995.

Examples

These examples show the output of the Divide block for some typical inputs using default block parameter values.


2
4
0.5
[4 8]
2
2
4
2
[4 8]
0.5
0.25
2 3
6 22
3 6
2 7
0.6667
0.5
3
3.143

Purpose

Create text that documents model and save text with model

Library

Model-Wide Utilities

Description

The DocBlock allows you to create and edit text that documents a model, and save that text with the model. Double-clicking an instance of the block creates a temporary file containing the text associated with this block and opens the file in an editor. Use the editor to modify the text and save the file. Simulink software stores the contents of the saved file in the model file.

The DocBlock supports HTML, Rich Text Format (RTF), and ASCII text document types. The default editors for these different document types are

- HTML — Microsoft® Word (if available). Otherwise, the DocBlock opens HTML documents using the editor specified on the **Editor/Debugger Preferences** pane of the Preferences dialog box.
- RTF — Microsoft Word (if available). Otherwise, the DocBlock opens RTF documents using the editor specified on the **Editor/Debugger Preferences** pane of the Preferences dialog box.
- Text — The DocBlock opens text documents using the editor specified on the **Editor/Debugger Preferences** pane of the Preferences dialog box.

Use the `docblock` command to change the default editors.

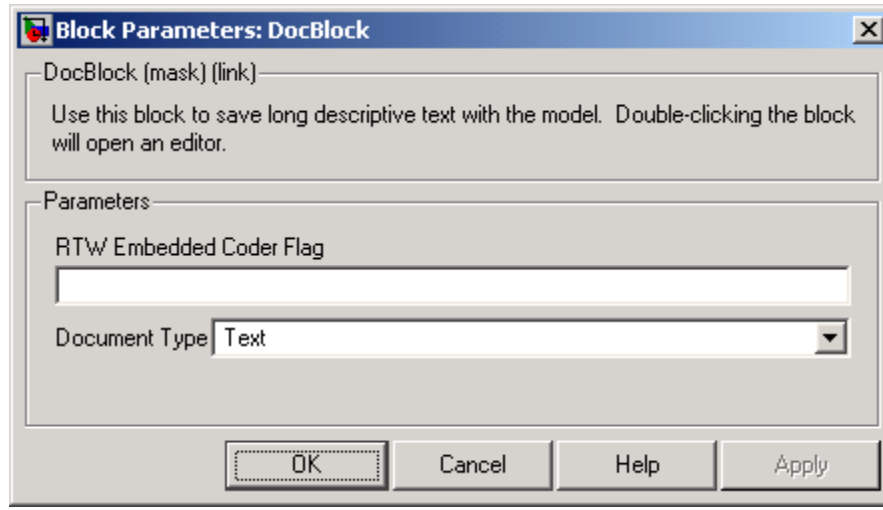
Note Simulink software embeds DocBlock documents in the model file (see Chapter 9, "Model File Format"). This can greatly increase the size of a model file, for example, if the RTF document contains bitmapped images, and can require more time to open and save the model.

Data Type Support

Not applicable.

Parameters and Dialog Box

Double-clicking an instance of the DocBlock opens an editor. To access the DocBlock parameter dialog box, select the block in the Model Editor and then select **Mask Parameters** from either the **Edit** menu or the block's context menu.



RTW Embedded Coder Flag (Real-Time Workshop® Embedded Coder™ license required)

Enter a template symbol name in this field. Real-Time Workshop Embedded Coder software uses this symbol to add comments to the code generated from the model. See in the documentation for more information.

Document Type

Specifies the type of document associated with the DocBlock. The options are

- `Text` (the default)
- `RTF`
- `HTML`

Characteristics Not applicable

Dot Product

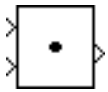
Purpose

Generate dot product of two vectors

Library

Math Operations

Description



The Dot Product block generates the dot product of the vectors at its inputs. The scalar output, `y`, is equal to the MATLAB operation

```
y = sum(conj(u1) .* u2 )
```

where `u1` and `u2` represent the vectors at the block's top and bottom inputs, respectively. (See "How to Rotate a Block" in the Simulink User's Guide for a description of the port order for various block orientations.) The inputs can be vectors, column vectors (single-column matrices), or scalars. If both inputs are vectors or column vectors, they must be the same length. If `u1` and `u2` are both column vectors, the block outputs the equivalent of the MATLAB expression `u1'*u2`.

The elements of the input vectors can be real- or complex-valued signals. The signal type (complex or real) of the output depends on the signal types of the inputs.

Input 1	Input 2	Output
real	real	real
real	complex	complex
complex	real	complex
complex	complex	complex

To perform element-by-element multiplication without summing, use the Product block.

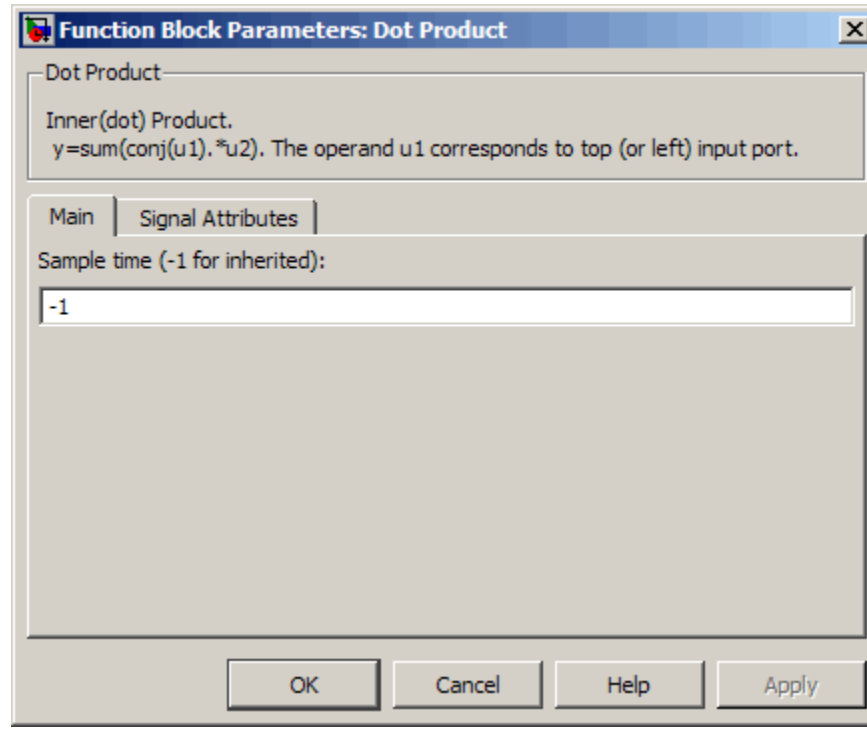
Data Type Support

The Dot Product block accepts and outputs signals of any numeric data type supported by Simulink software, including fixed-point data types.

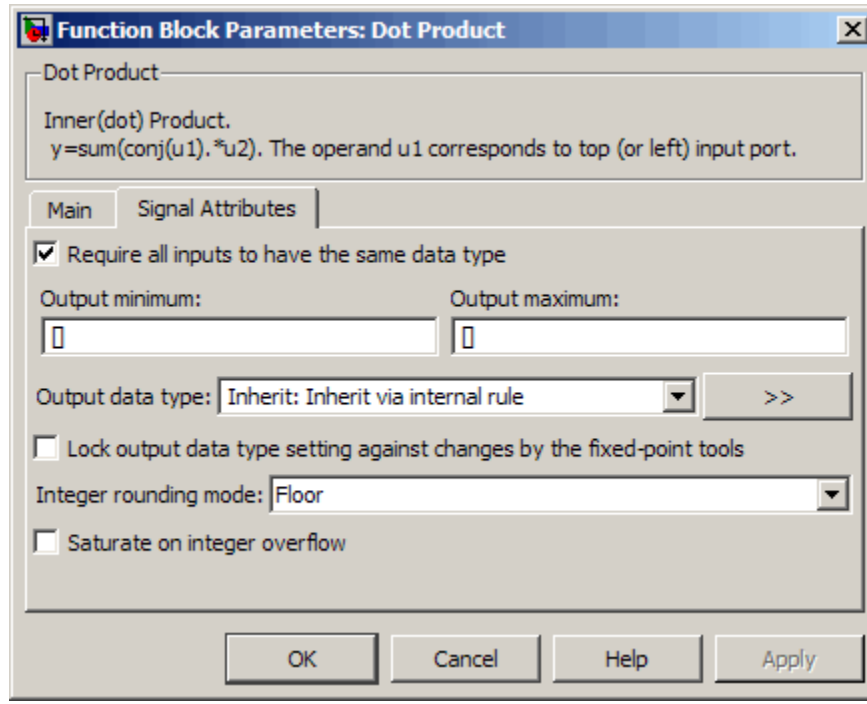
For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink".

Parameters and Dialog Box

The **Main** pane of the Dot Product block dialog box appears as follows:



The **Signal Attributes** pane of the Dot Product block dialog box appears as follows:



Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to -1. See "How to Specify the Sample Time" in the Simulink User's Guide for more information.

Require all inputs to have same data type

Select to require all inputs to have the same data type.

Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate on integer overflow
Select to have overflows saturate.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
States	0
Dimensionalized	Yes
Zero-Crossing Detection	No

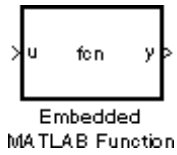
Purpose

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 within a Simulink model like the following example:

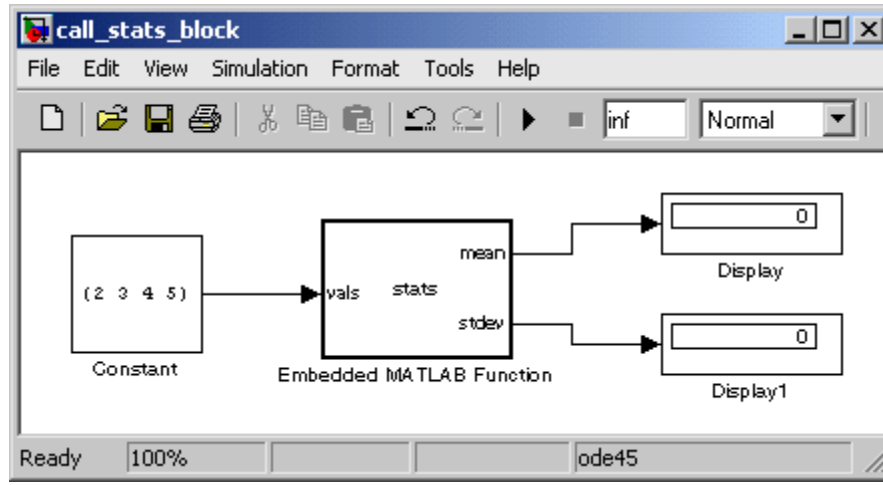
```
1   function [mean, stdev] = stats(vals)
2   %#eml
3
4   % calculates a statistical mean and standard deviation
5   % for the values in vals.
6
7   eml.extrinsic('plot');
8
9   len = length(vals);
10  mean = avg(vals, len);
11  stdev = sqrt(sum(((vals-avg(vals,len)).^2))/len);
12  plot(vals,'-+');
13
14  function mean = avg(array,size)
15  mean = sum(array)/size;
16
```

The MATLAB function you create executes for simulation and generates code for a Real-Time Workshop target. If you are new to the Simulink and MATLAB products, see “Using the Embedded MATLAB Function Block” in the 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 “Embedded MATLAB Function 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 the Simulink model.



The Embedded MATLAB Function block supports a subset of the language for which it can generate efficient embeddable code. For details about the Embedded MATLAB subset, see "Working with the Embedded MATLAB Subset" in the Embedded MATLAB documentation.

To generate embeddable code, the Embedded MATLAB Function block relies on an analysis that determines the size and class of each variable. This analysis imposes the following additional restrictions on the way in which the above features may be used.

1. The first definition of a variable must define both its class and size. The class and size of a variable cannot be changed once it has been set.

2. Whether data is complex or real is determined by the first definition. Subsequent definitions may assign real numbers into complex storage but may not assign complex numbers into real storage.

The preceding limitations require you to code in a certain style. Some common idioms to avoid are listed in "Using Matrix Indexing Operations" and "Working with Complex Numbers" in the Embedded MATLAB documentation.

In addition to language restrictions, Embedded MATLAB Function blocks support only a subset of the functions available in MATLAB. A list of supported functions is given in the "Embedded MATLAB Function Library Reference" in the Embedded MATLAB documentation. These functions include functions in common categories like

- Arithmetic functions like `plus`, `minus`, and `power`
- Matrix operations like `size`, and `length`
- Advanced matrix operations like `lu`, `inv`, `svd`, and `chol`
- Trigonometric functions like `sin`, `cos`, `sinh`, and `cosh`

to name just a few. See "Embedded MATLAB Function Library — Categorical List" in the Embedded MATLAB documentation for a complete list of function categories.

Note Although Embedded MATLAB software attempts to produce exactly the same results as MATLAB software, there will be occasions when they will differ due to rounding errors. These numerical differences, which may be a few `eps` initially, might be magnified after repeated operations. Reliance on the behavior of `nan` is not recommended. Different C compilers may yield different results for the same computation.

To support visualization of data, Embedded MATLAB Function blocks support calls to MATLAB functions for simulation only. See "Calling MATLAB Functions" in the Embedded MATLAB documentation to understand some of the limitations of this capability, and how it is integrated into Embedded MATLAB analysis. If these calls do not directly affect any of the Simulink inputs or outputs, they

are eliminated from the generated code when generating code with Real-Time Workshop.

You can declare an Embedded MATLAB input to be a Simulink parameter instead of a port in the Model Explorer. The Embedded MATLAB Function block also supports inheritance of types and size for inputs, outputs, and parameters. If needed, you can also set these explicitly using the Model Explorer. See "Typing Function Arguments", "Sizing Function Arguments", and "Parameter Arguments in Embedded MATLAB Functions", for more detailed descriptions of variables that you use in Embedded MATLAB Functions.

Note that recursive calls are not allowed in Embedded MATLAB functions.

Data Type Support

The Embedded MATLAB Function block accepts inputs of any type that Simulink supports, including fixed-point and enumerated types. For more information, see "Data Types Supported by Simulink" in the Simulink documentation.

For more information on fixed-point support in Embedded MATLAB, refer to "Working with the Fixed-Point Embedded MATLAB Subset" in the Fixed-Point Toolbox™ documentation.

The Embedded MATLAB Function block supports Simulink frames. See "Frame-Based Signals" in the Signal Processing Blockset documentation for more information.

Parameters and Dialog Box

The Block Parameters dialog box for an Embedded MATLAB Function block is identical to the Block Parameters dialog box for a Subsystem block. See the reference page for the Subsystem, Atomic Subsystem, Nonvirtual Subsystem, CodeReuse Subsystem blocks for an identification of each field.

Characteristics

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

Enable

Purpose

Add enabling port to subsystem

Library

Ports & Subsystems

Description

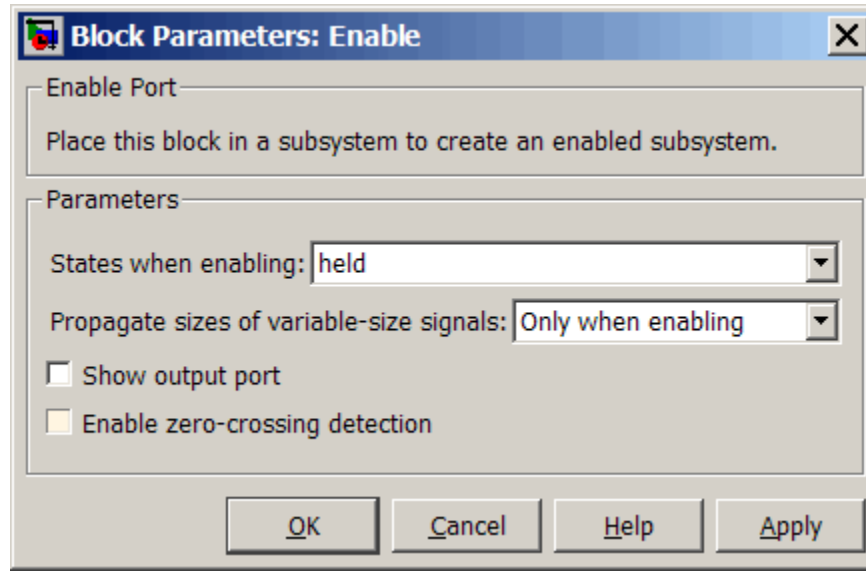
Adding an Enable block to a subsystem makes it an enabled subsystem. A subsystem can contain no more than one Enable block. An enabled subsystem executes while the input received at the Enable port is greater than zero.

At the start of a simulation, Simulink software initializes the states of blocks inside an enabled subsystem to their initial conditions.

Data Type Support

The input for the Enable block is the port that appears on the subsystem in which the Enable block resides. The data type for this input can be any numeric data type that Simulink supports, including fixed-point data types. For more information, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box



States when enabling

When an enabled subsystem executes after being disabled, specify what happens to the states of blocks in the enabled subsystem.

Settings

Default: `held`

`held`
: Holds the states at their previous values.

`reset`
: Resets the states to their initial conditions (zero if not defined).

Command-Line Information

Parameter: `StatesWhenEnabling`
Type: string
Value: `'held' | 'reset'`
Default: `'held'`

Propagate sizes of variable-size signals

Specify when to propagate a variable-size signal.

Settings

Default: `Only when enabling`

`Only when enabling`
: Propagates variable-size signals only when reenabling the subsystem containing the Enable Port block. When you select this option, sample time must be periodic.

`During execution`
: Propagates variable-size signals at each time step.

Command-Line Information

Parameter: `PropagateVarSize`
Type: string
Value: `'Only when enabling' | 'During execution'`
Default: `'Only when enabling'`

Show output port

Select this check box to output the enabling signal.

Settings

Default: On

 On

Shows the Enable block output port and outputs the enabling signal. Selecting this option allows the system to process the enabling signal.

 Off

Removes the output port from the Enable block.

Command-Line Information

Parameter: `ShowOutputPort`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

Enable zero-crossing detection

Select this check box to enable zero-crossing detection.

Settings

Default: On

 On

Detects zerocrossings.

 Off

Does not detect zerocrossing.

Command-Line Information

Parameter: `ZeroCross`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

Characteristics

Sample Time	Determined by the signal at the enable port
Dimensionalized	Yes
Virtual	Yes, if not connected directly to an Outport block For more information, see "Virtual Blocks" in the Simulink documentation.
Zero-Crossing Detection	Yes, if enabled

Enabled and Triggered Subsystem

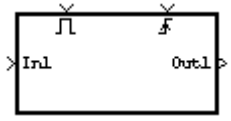
Purpose

Represent subsystem whose execution is enabled and 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 an enabled and triggered subsystem. For more information, see "Triggered and Enabled Subsystems" in the online Simulink help.

Purpose

Represent subsystem whose execution is enabled by external input

Library

Ports & Subsystems

Description

This block is a Subsystem block that is preconfigured to serve as the starting point for creating an enabled subsystem. For more information, see “Enabled Subsystems” in the “Creating a Model” chapter of the Simulink documentation.

Enumerated Constant

Purpose

Generate enumerated constant value

Library

Sources

Description

The Constant block generates a real or complex constant value. The block generates scalar, vector, or matrix output, depending on the dimensionality of the **Constant value** parameter and the setting of the **Interpret vector parameters as 1-D** parameter. Also, the block can generate either a sample-based or frame-based signal, depending on the setting of the **Sampling mode** 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, select the **Interpret vector parameters as 1-D** parameter. Otherwise, the block treats the **Constant value** parameter as a matrix.

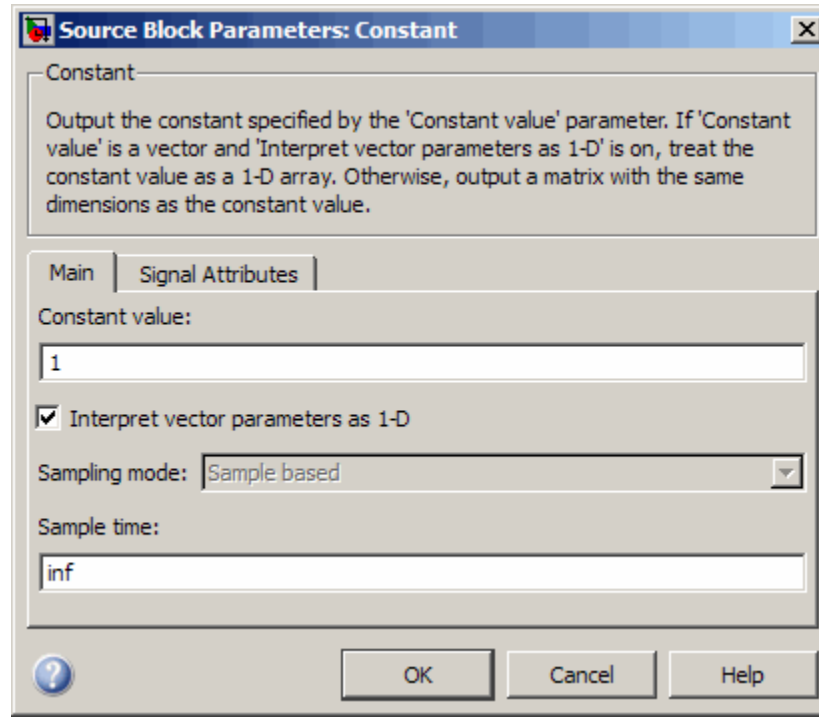
Data Type Support

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 data type supported by Simulink software, including fixed-point and enumerated data types.

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

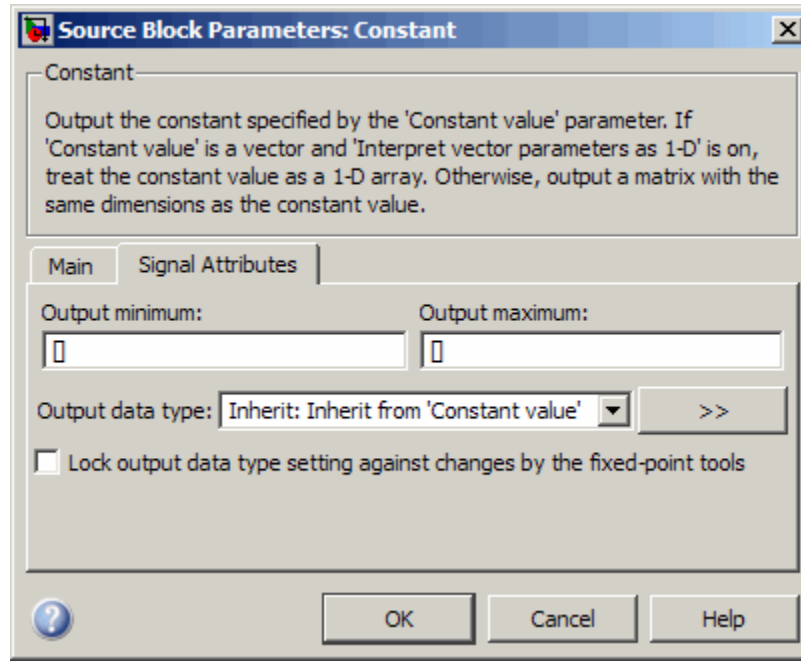
Parameters and Dialog Box

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



Opening this dialog box causes a running simulation to pause. See "Changing Source Block Parameters During Simulation" in the "Working with Blocks" chapter of the Simulink documentation.

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


Source Block Parameters: Constant
Constant
Output the constant specified by the 'Constant value' parameter. If 'Constant value' is a vector and 'Interpret vector parameters as 1-D' is on, treat the constant value as a 1-D array. Otherwise, output a matrix with the same dimensions as the constant value.
Main
Signal Attributes
Output minimum:
Output maximum:
Output data type:
Inherit: Inherit from 'Constant value'
>>
Lock output data type setting against changes by the fixed-point tools
OK
Cancel
Help

Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Constant value

Specify the constant value output of the block.

Settings

Default: 1

Minimum: value from the **Output minimum** parameter

Maximum: value from the **Output maximum** parameter

- You can enter any expression that MATLAB evaluates as a matrix, including the Boolean keywords `true` and `false`.
- This parameter is converted from its data type to the specified output data type offline using round toward nearest and saturation.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Interpret vector parameters as 1-D

Select this check box to output a vector of length `N` if the **Constant value** parameter evaluates to an `N`-element row or column vector.

Settings

Default: On

 On

Outputs a vector of length `N` if the **Constant value** parameter evaluates to an `N`-element row or column vector. For example, the block outputs a matrix of dimension `1-by-N` or `N-by-1`.

 Off

Does not output a vector of length `N` if the **Constant value** parameter evaluates to an `N`-element row or column vector.

If you clear this check box, you can interact with the **Sampling mode** parameter.

Dependencies

This parameter enables **Sampling mode**.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Sampling mode

Specify whether the output signal is `Sample based` or `Frame based`.

Settings

Default: `Sample based`

`Sample based`
: The output signal is sample-based.

`Frame based`
: The output signal is frame-based.

Tips

To generate frame-based signals, you must have the Signal Processing Blockset product installed.

For more information, see "Sample-Based Signals" and "Frame-Based Signals" in the Signal Processing Blockset User's Guide.

Dependencies

Clearing **Interpret vector parameters as 1-D** enables this parameter.

Selecting `Sample based` enables the following parameter:

- **Sample time**

Selecting `Frame based` enables the following parameter:

- **Frame period**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sample time

Specify the interval between times that the Constant block output can change during simulation (for example, due to tuning the **Constant value** parameter).

Settings

Default: `inf`

This setting indicates that the block output can never change. This setting speeds simulation and generated code by avoiding the need to recompute the block output.

See "How to Specify the Sample Time" in the online documentation for more information.

Dependency

Sampling mode enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Frame period

Specify the interval between frames that the Constant block output can change during simulation (for example, due to tuning the **Constant value** parameter).

Settings

Default: `inf`

Dependency

Sampling mode enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Output minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `Inherit: Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block), `Inherit: Inherit from 'Constant value'` (Constant block), `Inherit: Inherit via back propagation` (Data Type Conversion block), `Inherit: Same as input` (Saturation block)

`Inherit: Inherit via internal rule`
: This option appears for some blocks. Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the calculated 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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 software displays an error message in the Simulation Diagnostics Viewer.

`Inherit: Inherit from 'Constant value'`
: Use data type of **Constant value** parameter. This option appears for some blocks.

`Inherit: Inherit via back propagation`
: Use data type of the driving block.

`Inherit: Same as input`
: Use data type of sole input signal. This option appears for some blocks.

`Inherit: Same as first input`
: Use data type of first input signal. This option appears for some blocks.

`Inherit: Same as accumulator`
: Output data type is the same as accumulator data type. This option appears for some blocks.

`double`
: Output data type is `double`.

`single`
: Output data type is `single`.

`int8`
: Output data type is `int8`.

`uint8`
: Output data type is `uint8`.

`int16`
: Output data type is `int16`.

`uint16`
: Output data type is `uint16`.

`int32`
: Output data type is `int32`.

`uint32`
: Output data type is `unt32`.

`fixdt(1,16,0)`
: Output data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Output data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
: Use an enumerated data type, for example, `Enum: BasicColors`. This option appears for some blocks.

`<data type expression>`
: Use a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
Fixed-point data types.

`Enumerated`
Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `<class name>`

`Expression`
Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

See “Specifying a Fixed-Point Data Type” in the *Simulink User’s Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Characteristics

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

Environment Controller

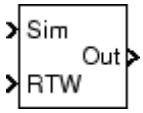
Purpose

Create branches of block diagram that apply only to simulation or only to code generation

Library

Signal Routing

Description



This block outputs the signal at its Sim port only if the model that contains it is being simulated. It outputs the signal at its RTW port only if code is being generated from the model. This allows you to create branches of a model's block diagram that apply only to simulation or only to code generation. The table below describes various scenarios where either the Sim or RTW port applies.

Scenario	Output
Normal mode simulation	Sim
Accelerator mode simulation	Sim
Rapid Accelerator mode simulation	RTW
Simulation of a referenced model (Normal or Accelerator modes)	Sim
Simulation of a referenced model (Processor-in-the-loop (PIL) mode)	RTW (uses the same code generated for a referenced model)
External mode simulation	RTW
Standard code generation	RTW
Code generation of a referenced model	RTW

Real-Time Workshop software does not generate code for blocks connected to the Sim port if these conditions hold:

- You select the **Inline parameters** check box on the **Optimization** pane of the Configuration Parameters dialog box.

- The blocks connected to the Sim port do not have external signals.

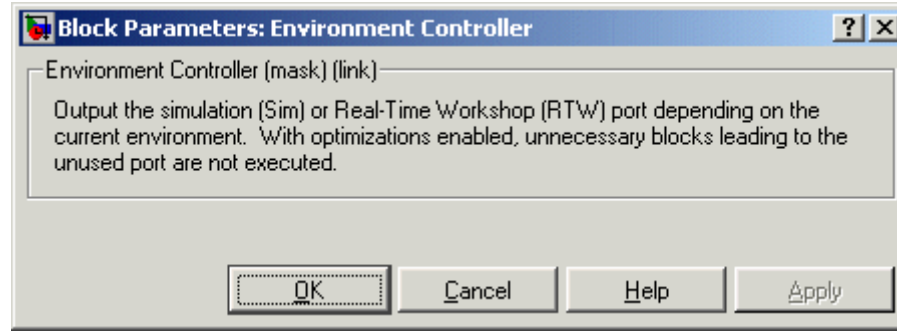
If you enable block reduction optimization, Simulink software eliminates blocks in the branch connected to the block's RTW port when compiling the model for simulation. For information about block reduction, see "Block reduction" in the online Simulink documentation.

Note Real-Time Workshop code generation eliminates the blocks connected to the Sim branch only if the Sim branch has the same signal dimensions as the RTW branch. Regardless of whether it eliminates the Sim branch, Real-Time Workshop uses the sample times on the Sim branch as well as the RTW branch to determine the fundamental sample time of the generated code and may, in some cases, generate sample-time handling code that applies only to sample times specified on the Sim branch.

Data Type Support

The Environment Controller block accepts signals of any data type supported by Simulink software. It outputs the type at its input.

Parameters and Dialog Box



Characteristics

Multidimensionalized	Yes

Extract Bits

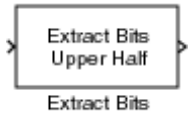
Purpose

Output selection of contiguous bits from input signal

Library

Logic and Bit Operations

Description



The Extract Bits block allows you to output a contiguous selection of bits from the stored integer value of the input signal. Use the **Bits to extract** parameter to define the method for selecting the output bits.

- Select `Upper half` to output the half of the input bits that contain the most significant bit. If there is an odd number of bits in the input signal, the number of output bits is given by the equation

  $$\text{number of output bits} = \text{ceil}(\text{number of input bits}/2)$$

- Select `Lower half` to output the half of the input bits that contain the least significant bit. If there is an odd number of bits in the input signal, the number of output bits is given by the equation

  $$\text{number of output bits} = \text{ceil}(\text{number of input bits}/2)$$

- Select `Range starting with most significant bit` to output a certain number of the most significant bits of the input signal. Specify the number of most significant bits to output in the **Number of bits** parameter.
- Select `Range ending with least significant bit` to output a certain number of the least significant bits of the input signal. Specify the number of least significant bits to output in the **Number of bits** parameter.
- Select `Range of bits` to indicate a series of contiguous bits of the input to output in the **Bit indices** parameter. You indicate the range in `[start end]` format, and the indices of the input bits are labeled contiguously starting at 0 for the least significant bit.

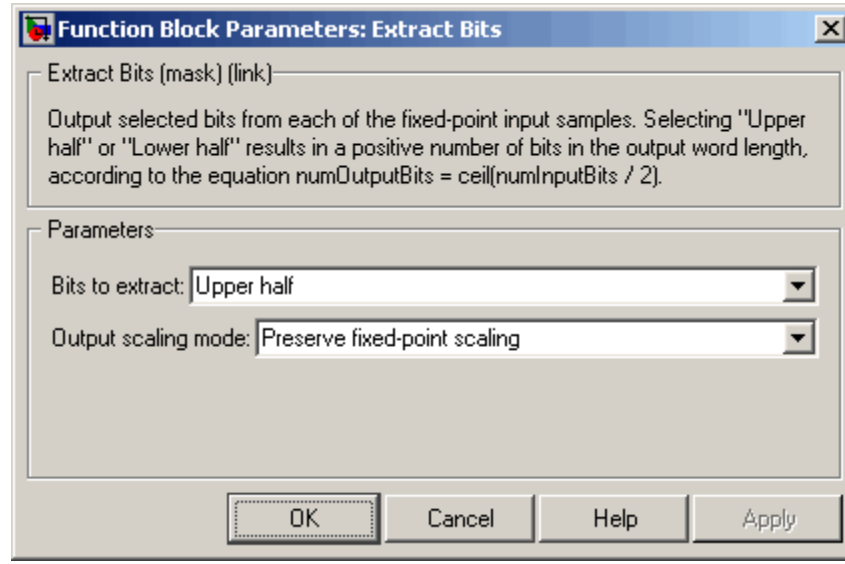
Data Type Support

The Extract Bits block accepts inputs of any numeric data type supported by Simulink software, including fixed-point data types. Floating-point inputs are passed through the block unchanged. `Boolean` inputs are treated as `uint8` signals.

Note Performing bit operations on a signed integer is difficult. You can avoid difficulty by converting the data type of your input signals to unsigned integer types.

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

Parameters and Dialog Box



Bits to extract

Select the method for extracting bits from the input signal.

Number of bits

(Not shown on dialog above.) Select the number of bits to output from the input signal. Signed integer data types can have no less then two bits in them. Unsigned data integer types can be as short as a single bit.

This parameter is only visible if you select `Range starting with most significant bit` or `Range ending with least significant bit` for the **Bits to extract** parameter.

Bit indices

(Not shown on dialog above.) Specify a contiguous range of bits of the input signal to output. Specify the range in `[start end]` format. The indices are assigned to the input bits starting with 0 at the least significant bit.

This parameter is only visible if you select `Range of bits` for the **Bits to extract** parameter.

Output scaling mode

Select the scaling mode to use on the output bits selection:

- When you select `Preserve fixed-point scaling`, the fixed-point scaling of the input is used to determine the output scaling during the data type conversion.
- When you select `Treat bit field as an integer`, the fixed-point scaling of the input is ignored, and only the stored integer is used to compute the output data type.

Example

Consider an input signal that is represented in binary by `110111001`:

- If you select `Upper half` for the **Bits to extract** parameter, the output is `11011` in binary.
- If you select `Lower half` for the **Bits to extract** parameter, the output is `11001` in binary.

- If you select `Range starting with most significant bit` for the **Bits to extract** parameter, and specify `3` for the **Number of bits** parameter, the output is `110` in binary.
- If you select `Range ending with least significant bit` for the **Bits to extract** parameter, and specify `8` for the **Number of bits** parameter, the output is `10111001` in binary.
- If you select `Range of bits` for the **Bits to extract** parameter, and specify `[4 7]` for the **Bit indices** parameter, the output is `1011` in binary.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited
Scalar Expansion	N/A
States	None
Dimensionalized	Inherited
Multidimensionalized	Yes
Zero Crossing	No

Fcn

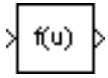
Purpose

Apply specified expression to input

Library

User-Defined Functions

Description



The Fcn block applies the specified mathematical expression to its input. The expression can be made up of one or more of these components:

- `u` — The input to the block. If `u` is a vector, `u(i)` represents the `i`th element of the vector; `u(1)` or `u` alone represents the first element.
- Numeric constants
- Arithmetic operators (`+ - * / ^`)
- Relational operators (`== != > < >= <=`) — The expression returns 1 if the relation is true; otherwise, it returns 0.
- Logical operators (`&& || !`) — The expression returns 1 if the relation is true; otherwise, it returns 0.
- Parentheses
- Mathematical functions — `abs`, `acos`, `asin`, `atan`, `atan2`, `ceil`, `cos`, `cosh`, `exp`, `fabs`, `floor`, `hypot`, `ln`, `log`, `log10`, `pow`, `power`, `rem`, `sgn`, `sin`, `sinh`, `sqrt`, `tan`, and `tanh`.

Note The Fcn block does not support `round` and `fix`. Use the Rounding Function block to apply these rounding modes.

- Workspace variables — Variable names that are not recognized in the preceding list of items are passed to MATLAB for evaluation. Matrix or vector elements must be specifically referenced (e.g., `A(1,1)` instead of `A` for the first element in the matrix).

The Fcn block observes the following rules of operator precedence:

1 ( )

2 ^

3 + - (unary)

4 !

5 * /

6 + -

7 > < <= >=

8 == !=

9 &&

10 ||

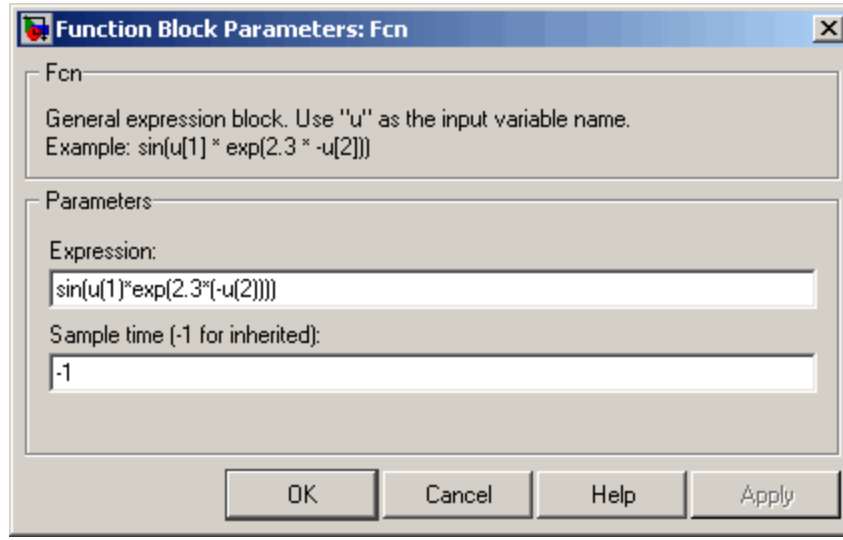
The expression differs from a MATLAB expression in that the expression cannot perform matrix computations. Also, this block does not support the colon operator (:).

Block input can be a scalar or vector. The output is always a scalar. For vector output, consider using the Math Function block. If a block input is a vector and the function operates on input elements individually (for example, the `sin` function), the block operates on only the first vector element.

Data Type Support

The Fcn block accepts and outputs signals of type `single` or `double`.

Parameters and Dialog Box



Expression

The mathematical expression applied to the input. Expression components are listed above. The expression must be mathematically well formed (i.e., matched parentheses, proper number of function arguments, etc.).

Note You cannot tune the expression during accelerated-mode simulation (see "Accelerating Models"), in referenced models executing in Accelerator mode (see "Referencing a Model", or in generated code.

The Fcn block does not support custom storage classes. See "Creating and Using Custom Storage Classes".

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the 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	No

First-Order Hold

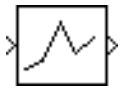
Purpose

Implement first-order sample-and-hold

Library

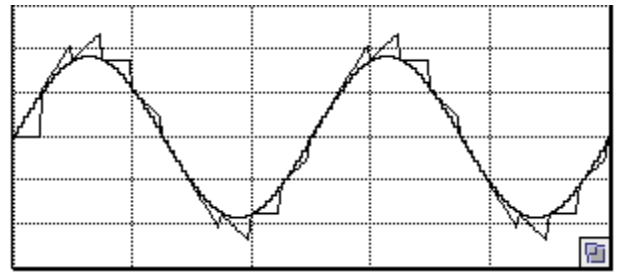
Discrete

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.

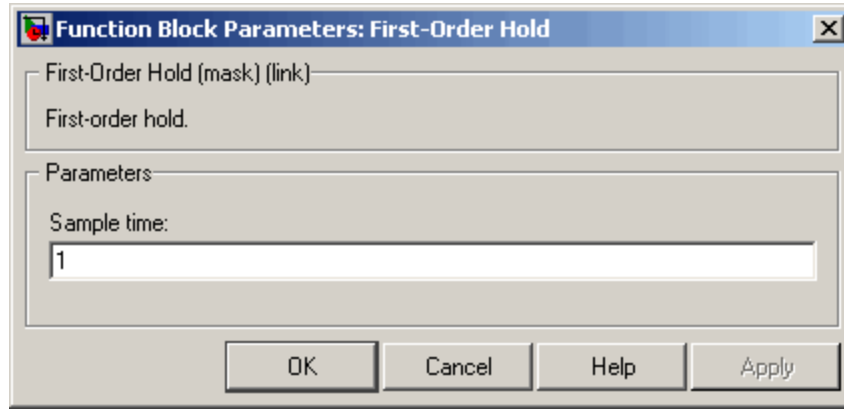
This figure compares the output from a Sine Wave block and a First-Order Hold block.



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 "How to Specify the 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

Fixed-Point State-Space

Purpose

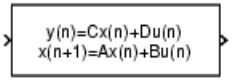
Implement discrete-time state space

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 an n-by-1 vector.
- The input u must be an m-by-1 vector.
- The output y must be an r-by-1 vector.

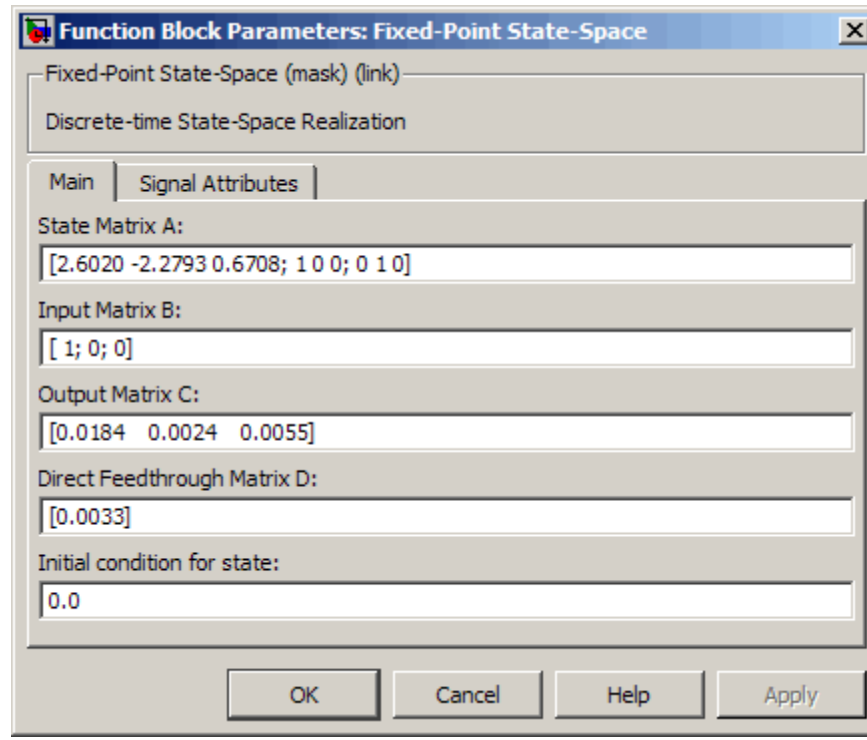
The block accepts one input and generates one output. The block determines the input vector width by the number of columns in the B and D matrices. Similarly, the block determines the output vector width by the number of rows in the C and D matrices.

Data Type Support

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

Parameters and Dialog Box

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



State Matrix A
Specify the matrix of states.

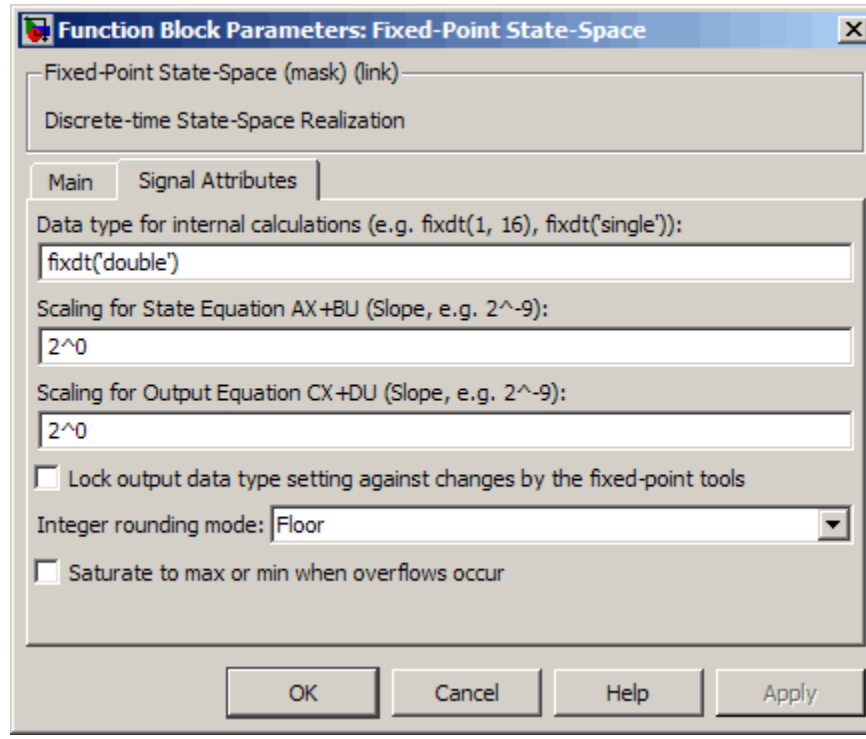
Input Matrix B
Specify the column vector of inputs.

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 Attributes** pane of the Fixed-Point State-Space block dialog box 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 data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate. Otherwise, they wrap.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

For Iterator

Purpose

Repeatedly execute contents of subsystem at current time step until iteration variable exceeds specified iteration limit

Library

Ports & Subsystems/For Iterator Subsystem

Description

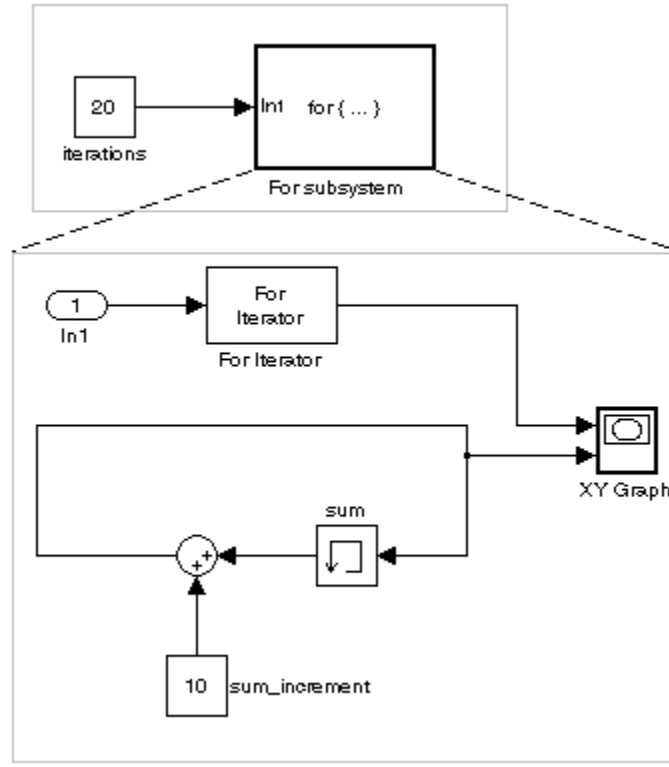
The For Iterator block, when placed in a subsystem, repeatedly executes the contents of the subsystem at the current time step until an iteration variable exceeds a specified iteration limit. You can use this block to implement the block diagram equivalent of a `for` loop in the C programming language.

The output of a For Iterator subsystem can not be a function-call signal. Simulink software will display an error message if the simulation is run or the diagram updated.

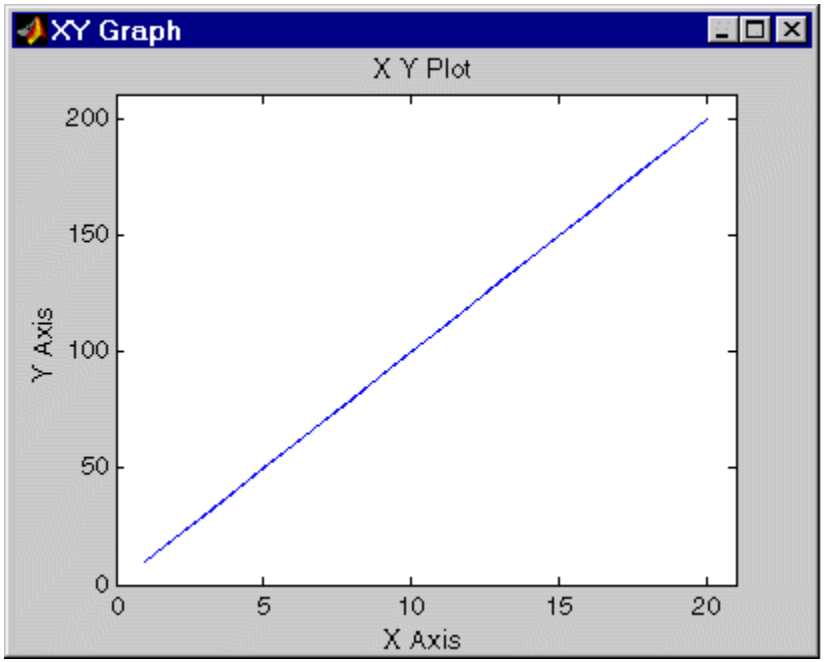
The block's parameter dialog allows you to specify the maximum value of the iteration variable or an external source for the maximum value and an optional external source for the next value of the iteration variable. If you do not specify an external source for the next value of the iteration variable,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

```
sum = 0;
iterations = 20;
sum_increment = 10;
for (i = 0; i < iterations; i++) {
 sum = sum + sum_increment;
}
```

Note Placing a For Iterator block in a subsystem makes it an atomic subsystem if it is not already an atomic subsystem.

Data Type Support

The following rules apply to the data type of the number of iterations (N) input port:

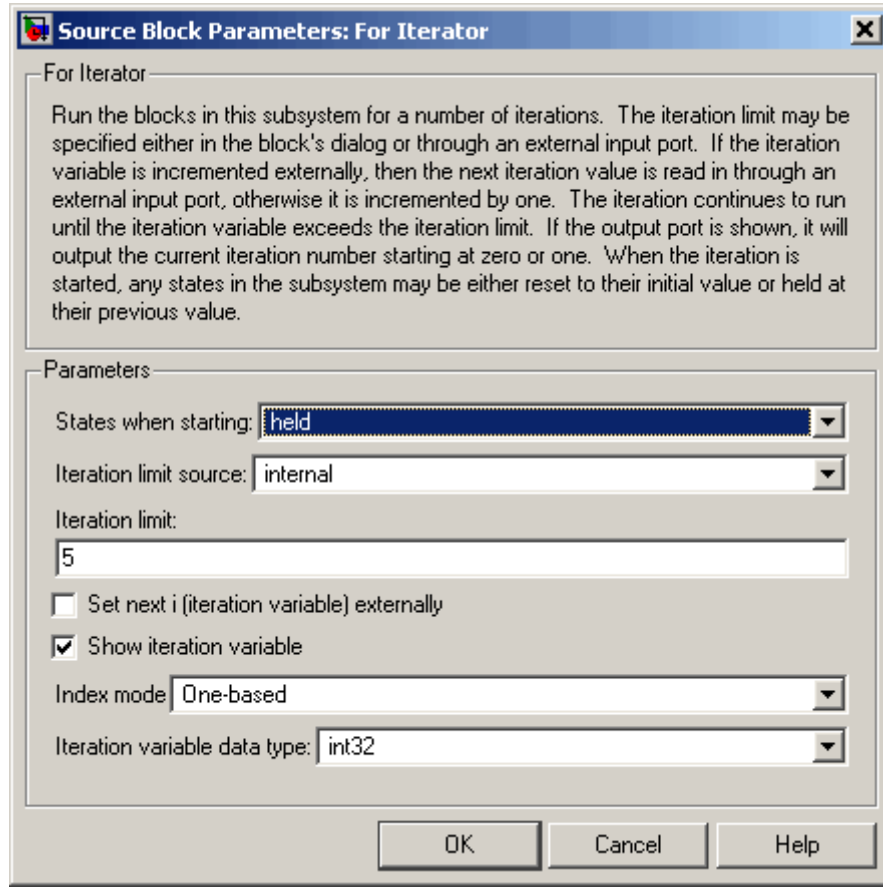
- The input port accepts data of mixed numeric types.
- If the input port value is noninteger, it is first truncated to an integer.
- Internally, the input value is cast to an integer of the type specified for the iteration variable output port.
- If no output port is specified, the input port value is cast to type `int32`.
- If the input port value exceeds the maximum value of the output port's type, it is truncated to that maximum value.

Data output for the iterator value can be selected as `double`, `int32`, `int16`, or `int8` in the Block Properties dialog.

The following rules apply to the iteration variable input port.

- It can appear only if the iteration variable output port is enabled.
- The data type of the iteration variable input port is the same as the data type of the iteration variable output port.

Parameters and Dialog Box



States when starting

Set this field to `reset` if you want the states of the For subsystem to be reinitialized before the first iteration at each time step. Otherwise, set this field to `held` (the default) to make sure that these subsystem states retain their values from the last iteration at the previous time step.

Iteration limit source

If you set this field to `internal`, the value of the **Iteration limit** field determines the number of iterations. If you set this field to `external`, the signal at the For Iterator block's `N` port determines the number of iterations. The iteration limit source must reside outside the For Iterator subsystem.

Iteration limit

Set the number of iterations by specifying a number or a named constant. This field appears only if you selected `internal` for the **Iteration limit source** field. This parameter supports storage classes. You can define the named constant in the base workspace of the Model Explorer as a `Simulink.Parameter` object of the built-in storage class `Define (custom)` type. For more information, see "Applying a Custom Storage Class to a Parameter" in the Real-Time Workshop Embedded Coder documentation.

Set next i (iteration variable) externally

This option can be selected only if you select the **Show iteration variable** option. If you select this option, the For Iterator block displays an additional input for connecting an external iteration variable source. The value of the input at the current iteration is used as the value of the iteration variable at the next iteration.

Show iteration variable

If you select this check box, the For Iterator block outputs its iteration value.

Index mode

If you set this field to `Zero-based`, the iteration number starts at zero. If you set this field to `One-based`, the iteration number starts at one.

Iteration variable data type

Set the type for the iteration value output from the iteration number port to `double`, `int32`, `int16`, or `int8`.

Characteristics

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

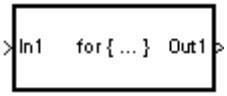
Purpose

Represent subsystem that executes repeatedly during simulation time step

Library

Ports & Subsystems

Description



The For Iterator Subsystem block is a Subsystem block that is preconfigured to serve as a starting point for creating a subsystem that executes repeatedly during a simulation time step.

For more information, see the For Iterator block in the online Simulink block reference and "Modeling Control Flow Logic" in the Simulink documentation.

When using simplified initialization mode, you cannot place any block needing elapsed time within an Iterator Subsystem. In simplified initialization mode, Iterator subsystems do not maintain elapsed time, so Simulink will report an error if any such block (such as the Discrete-Time Integrator block) is placed within the subsystem. For more information on simplified initialization modes, see "Underspecified initialization detection".

From

Purpose

Accept input from Goto block

Library

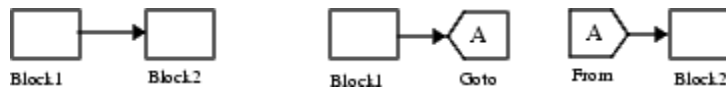
Signal Routing

Description

The From block accepts a signal from a corresponding Goto block, then passes it as output. The data type of the output is the same as that of the input from the Goto block. From and Goto blocks allow you to pass a signal from one block to another without actually connecting them. To associate a Goto block with a From block, enter the Goto block's tag in the **Goto Tag** parameter.

A From block can receive its signal from only one Goto block, although a Goto block can pass its signal to more than one From block.

This figure shows that using a Goto block and a From block is equivalent to connecting the blocks to which those blocks are connected. In the model at the left, Block1 passes a signal to Block2. That model is equivalent to the model at the right, which connects Block1 to the Goto block, passes that signal to the From block, then on to Block2.



The visibility of a Goto block tag determines the From blocks that can receive its signal. For more information, see Goto and Goto Tag Visibility. The block indicates the visibility of the Goto block tag:

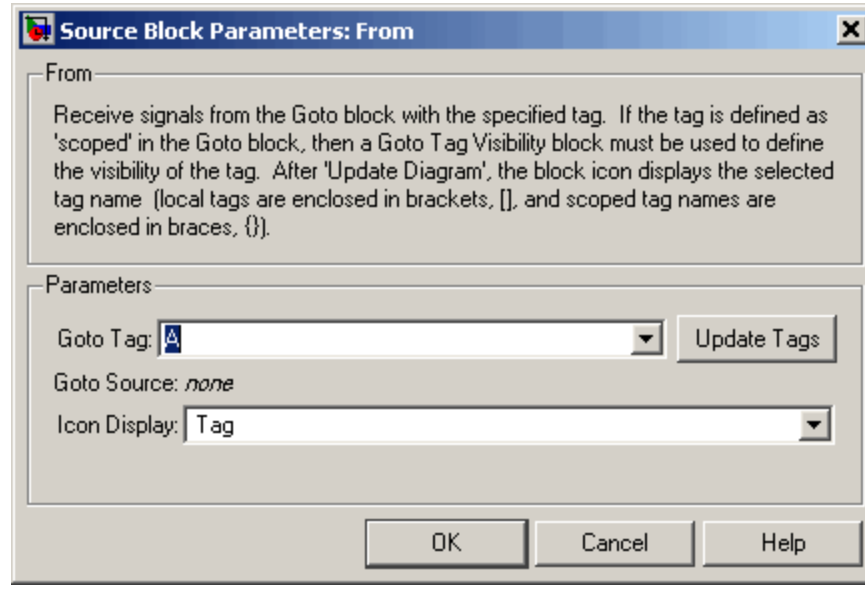
- A local tag name is enclosed in brackets ([]).
- A scoped tag name is enclosed in braces ({}).
- A global tag name appears without additional characters.

Data Type Support

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

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

Parameters and Dialog Box



Goto Tag

The tag of the Goto block that forwards its signal to this From block. To change the tag, select a new tag from this control's drop-down list. The drop-down list displays the Goto tags that the From block can currently see. An item labeled `<More Tags...>` appears at the end of the list the first time you display the list in a Simulink session. Selecting this item causes the block to update the tags list to include the tags of Goto blocks residing in library subsystems referenced by the model containing this From block. Simulink software displays a progress bar while building the list of library tags. Simulink software saves the updated tags list for the duration of the Simulink session or until the next time you select the adjacent **Update Tags** button. You need to update the tags list again in the current session only if the libraries referenced by the model have changed since the last time you updated the list.

Update Tags
Updates the list of tags visible to this From block, including tags residing in libraries referenced by the model containing this From block.

Goto Source
Path of the Goto block connected to this From block. Clicking the path displays and highlights the Goto block.

Icon Display
Specifies the text to display on the From block's icon. The options are the block's tag, the name of the signal that the block represents, or both the tag and the signal name.

Characteristics

Sample Time	Inherited from block driving the Goto block
Dimensionalized	Yes
Multidimensionalized	Yes
Virtual	Yes For more information, see "Virtual Blocks" in the Simulink documentation.

Purpose

Read data from MAT-file

Library

Sources

Description

The From File block reads scalar or vector data of type `double` from a MAT-file outputs the data as a signal. The block's icon shows the pathname of the file supplying the data. Simulink software reads the MAT-file into memory at the start of the simulation, automatically uncompressing it if it had previously been saved by MATLAB in a compressed format. Therefore:

- Enough memory must be available at the start of simulation to contain the complete uncompressed MAT-file.
- A From File block cannot read data from a MAT-file that is being written by a To File block during the current simulation.

The MAT-file contains the stored data as a matrix of two or more rows. The first element of each column contains a simulation time. The remainder of each column contains scalar or vector data for the time shown at the top of the column, one element for each data point in the input. The time values in the first row must increase monotonically. The matrix in the file 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 width of the output depends on the number of rows in the MAT-file. The block uses the time data at the top of each column to determine when to output the data values in the column, but does not output the time value itself. This means that given a MAT-file containing m rows, the block outputs a vector of length m-1, consisting of data from all but the first row of each column.

See “Importing Data from a Workspace” for guidelines on choosing time vectors for discrete systems.

Missing and Duplicate Time Stamps

If an output value is needed at a time that falls between two values in the MAT-file, the value is linearly interpolated between the appropriate values. If the required time is less than the first time value or greater than the last time value in the MAT-file, Simulink software extrapolates, using the first two or last two data points to compute a value.

If the matrix includes two or more columns at the same time value, the output is the data point for the first such 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.

Using Data Saved by a To File Block

The From File block can read data written by a To File block without any modifications to the data or other special provisions.

Using Data Saved by a To Workspace Block

The From File block can read data written by a To Workspace block subject to the following requirements:

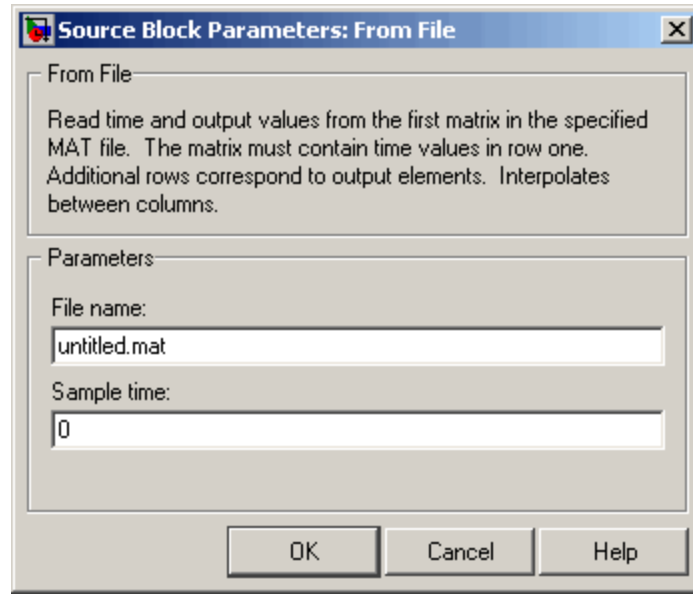
- The data must include the simulation times. The easiest way to include time data in the simulation output is to specify a variable for time on the **Data Import/Export** pane of the **Configuration Parameters** dialog box. See “Data Import/Export Pane” for more information.
- The data must be the transposition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

- The data in the file must be scalar or vector data of type `double`.

Data Type Support

The From File block can read data only in MAT-file format. The block can output only vector and scalar data of type `double`. The block cannot output matrix signals or complex data.

Parameters and Dialog Box



File name

The fully qualified pathname or file name of the MAT-file that contains the data used as input. On UNIX® systems, 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 software 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 software 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. The default is `0`, which specifies continuous sample time: the MAT-file is read at the base (fastest) rate of the model. See "How to Specify the Sample Time" for more information.

If the specified **Sample time** requires data at a time for which the MAT-file contains no matching time stamp, Simulink software extrapolates or interpolates to obtain the needed data, as described in "Missing and Duplicate Time Stamps" on page 2-476. If the MAT-file contains columns with time stamps that the specified **Sample time** never requires, the data points in columns with those time stamps are ignored.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	1-D array only
Zero Crossing	No

See Also

"Importing and Exporting Data", From Workspace, To File, To Workspace

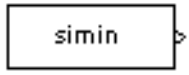
Purpose

Read data from workspace

Library

Sources

Description



The From Workspace block reads data from a workspace and outputs the data as a signal. The block's **Data** parameter specifies the workspace data using a MATLAB expression that evaluates to one of the following:

- A two-dimensional matrix:
  - The first element of each matrix row is a time stamp.
  - The rest of each row is a scalar or vector of signal values.

  The leftmost element of each row is the time stamp of the value(s) in the rest of the row.
- A structure in Structure with Time format, which contains:
  - A vector named `time`, which contains time stamps
  - An array or matrix named `signals.values`, which contains scalars or vectors of signal values.

  The `n`'th `time` element is the time stamp of the `n`'th `signals.values` element. See "Importing Data from a Workspace" for a description of `Structure with Time` format.
- A `Simulink.Timeseries` object.

The From Workspace icon displays the expression specified in the **Data** parameter. The Simulink software evaluates this expression as described in "Resolving Symbols".

The format of a matrix or structure read by a From Workspace block is the same as that used to load root-level input port data from the workspace. You must use the Structure with Time format or a time series object to load matrix (2-D) data from the workspace. See "Importing Data from a Workspace" for guidelines on choosing time vectors for discrete systems. See the documentation of the `sim`

command for some data import capabilities that are available only for programmatic simulation.

Limitation: You cannot use a From Workspace block to import fixed-point data that is contained in a structure. Consider using a `Simulink.Timeseries` object instead of a structure.

Using Data Saved by a To File Block

The From Workspace block requires data written by the To File block to be transposed before the From Workspace block can read it. The source file contains consecutive time stamps at the beginnings of columns, followed by the corresponding data. The transposed data contains consecutive time stamps at the beginning of rows, followed by the corresponding data. To provide the required format, use MATLAB commands to `load` and transpose (`'`) the MAT-file (see "Reshaping a Matrix"). You can then use a From Workspace block to access the data loaded into the workspace. Resave the transposed data if needed to avoid retransposing it again later.

Using Data Saved by a To Workspace Block

The From Workspace block requires data written by a To Workspace block to be written in `Structure with Time` format. Use this format to save sample-based data if you intend to use a From Workspace block to read the data back into another simulation. See "Importing Data from a Workspace" for a description of `Structure with Time` format.

Limitation: You cannot use a From Workspace block to import fixed-point data that is contained in a structure. Consider using a `Simulink.Timeseries` object instead of a structure.

Interpolating Missing Data Values

If you select the **Interpolate data** option, the block uses linear Lagrangian interpolation to compute data values for time steps that occur between time steps for which the workspace supplies data. In particular, the block linearly interpolates a missing data point from the two known data points between which it falls. For example, suppose the block reads the following time series from the workspace:

```
time:      1   2   3    4
signal:   253 254  ?  256
```

The block would output:

```
time:      1   2   3    4
signal:   253 254 255  256
```

If you clear the **Interpolate data** option, the block uses the most recent data value supplied from the workspace to provide any missing data values. For example, the result of the incomplete set of signal values shown above would be:

```
time:      1   2   3    4
signal:   253 254 254  256
```

Interpolating Integer Data

If the input data type is an integer type and an interpolated data point exceeds the data type's range, the block sets the missing data point to be the maximum value that the data type can represent. Similarly, if the interpolated or extrapolated value is less than the minimum value that the data type can represent, the block sets the missing data point to the minimum value that the data type can represent. For example, suppose that the data type is `uint8` and the value interpolated for a missing data point is 256.

```
time:      1   2   3   4
signal:   253 254 255  ?
```

In this case, the block sets the value of the missing point to 255, the largest value that can be represented by the `uint8` data type:

```
time:      1   2   3    4
signal:   253 254 255  255
```

Interpolating Boolean Data

If the input data is boolean, the block uses the value of the nearest workspace data point as the value of missing data point when

determining missing data points that fall between the first and last known points. For example, suppose the workspace supplies values at time steps 1 and 4 but not at 2 and 3:

```
time:     1 2 3 4
signal:   1 ? ? 0
```

In this case, the block would use the value of data point 1 as the value of data point 2 and the value of data point 4 as the value of data point 3:

```
time:     1 2 3 4
signal:   1 1 0 0
```

The block uses the value of the last known data point as the value of time steps that occur after the last known data point.

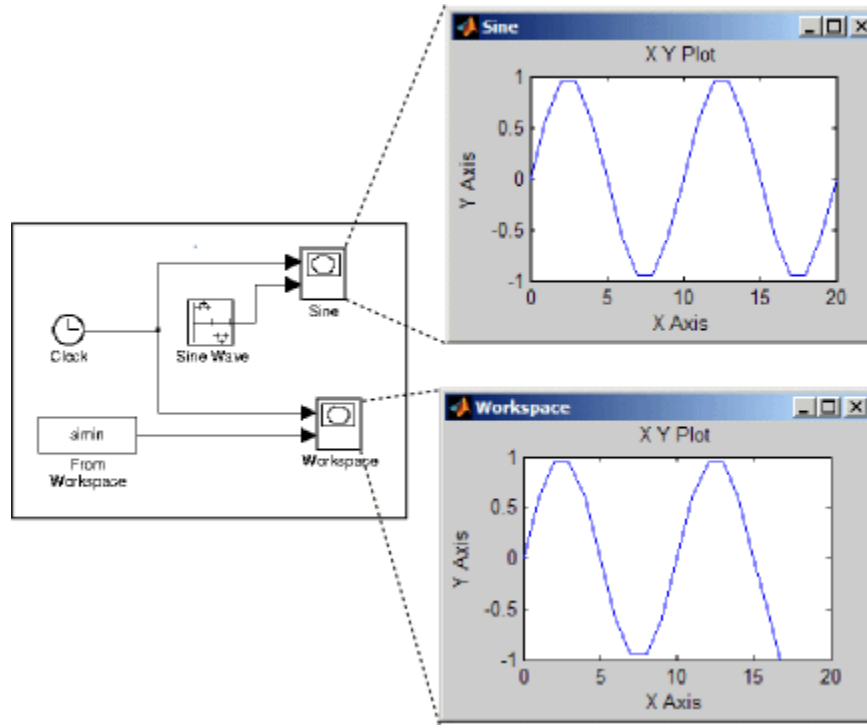
Specifying Output After Final Data

The block's **Form output after final data value by** and **Interpolate data** parameters work together to determine the block's output after the last time step for which workspace data is available. The following table describes the block output based on the values of the two options:

Form Output Option	Interpolate Option	Block Output After Final Data
Extrapolate	On	Extrapolated from final data value
	Off	Error
Setting to zero	On	Zero
	Off	Zero
Holding final value	On	Final value from workspace
	Off	Final value from workspace

Form Output Option	Interpolate Option	Block Output After Final Data
Cyclic repetition	On	Error
	Off	Repeated from workspace if the workspace data is in structure-without-time format. Error otherwise.

For example, if **Form output after final data value by** is `Extrapolate` and **Interpolate data** is selected, the block uses the last two known data points to extrapolate data points that occur after the last known point. Consider the following model.



In this model, the From Workspace block reads data from the workspace consisting of the output of the Simulink Sine block sampled at one-second intervals. The workspace contains the first 16 samples of the output. The top and bottom X-Y plots display the output of the Sine Wave and From Workspace blocks, respectively, from 0 to 20 seconds. The straight line in the output of the From Workspace block reflects the block's linear extrapolation of missing data points at the end of the simulation.

Detection of Zero Crossings

If the input array contains more than one entry for the same time step, Simulink software detects a zero crossing at that time step. For example, suppose the input array has this data:

```
time:     0 1 2 2 3
signal:   2 3 4 5 6
```

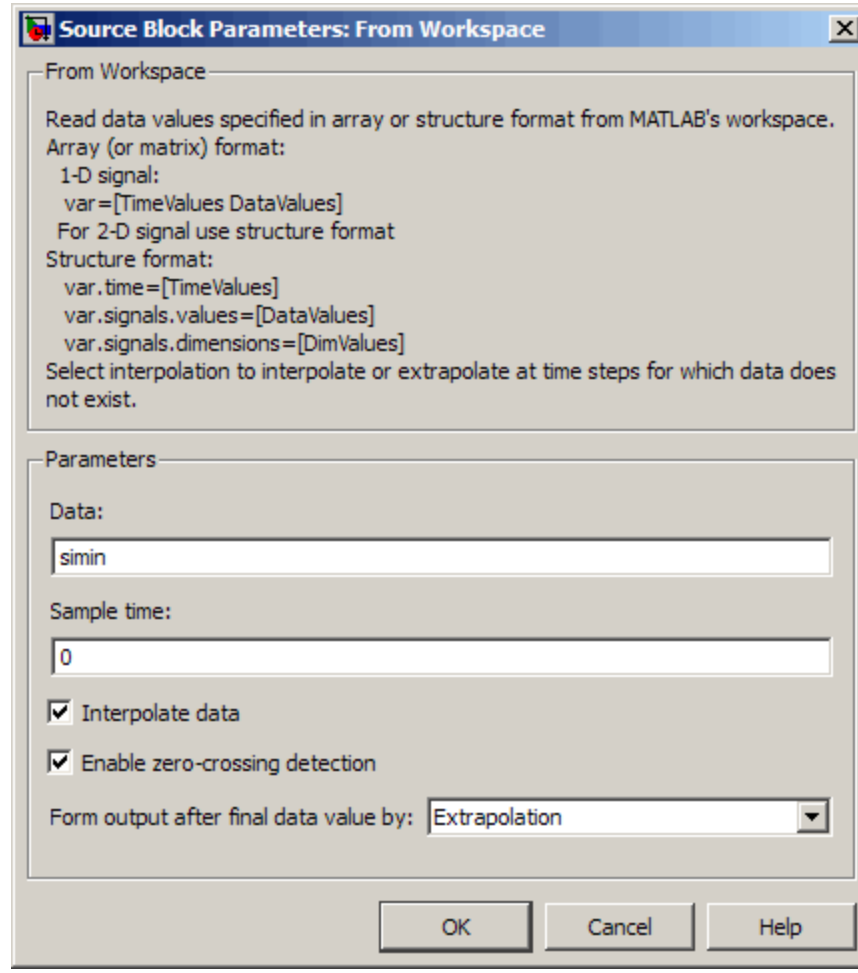
At time 2, there is a zero crossing from input signal discontinuity. The effect of this zero crossing depends on the value of the **Enable zero-crossing detection** parameter. See "Zero-Crossing Detection" for more information.

Data Type Support

The From Workspace block accepts from the workspace and outputs real or complex signals of any type supported by Simulink software, including fixed-point and enumerated data types. Real signals of type `double` can be in either structure or matrix format. Complex signals and real signals of any type other than `double` must be in structure format.

Limitation: You cannot use a From Workspace block to import fixed-point data that is contained in a structure. Consider using a `Simulink.Timeseries` object instead of a structure.

Parameters and Dialog Box



Data

An expression that evaluates to an array or a structure containing an array of simulation times and corresponding signal values. For example, suppose that the workspace contains a column vector of times named `T` and a column vector of corresponding signal values

named `U`. Entering the expression `[T U]` for this parameter yields the required input array. If the required signal-versus-time array or structure already exists in the workspace, enter the name of the structure or matrix in this field.

Sample time
Sample rate of data from the workspace. See "How to Specify the Sample Time" in the online documentation for more information.

Interpolate data
This option causes the block to linearly interpolate at time steps for which no corresponding workspace data exists. Otherwise, the current output equals the output at the most recent time for which data exists.

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

Form output after final data value by
Select method for generating output after the last time point for which data is available from the workspace.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	Yes
Multidimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled.

See Also

"Importing and Exporting Data", From File, To File, To Workspace

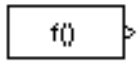
Purpose

Execute function-call subsystem specified number of times at specified rate

Library

Ports & Subsystems

Description

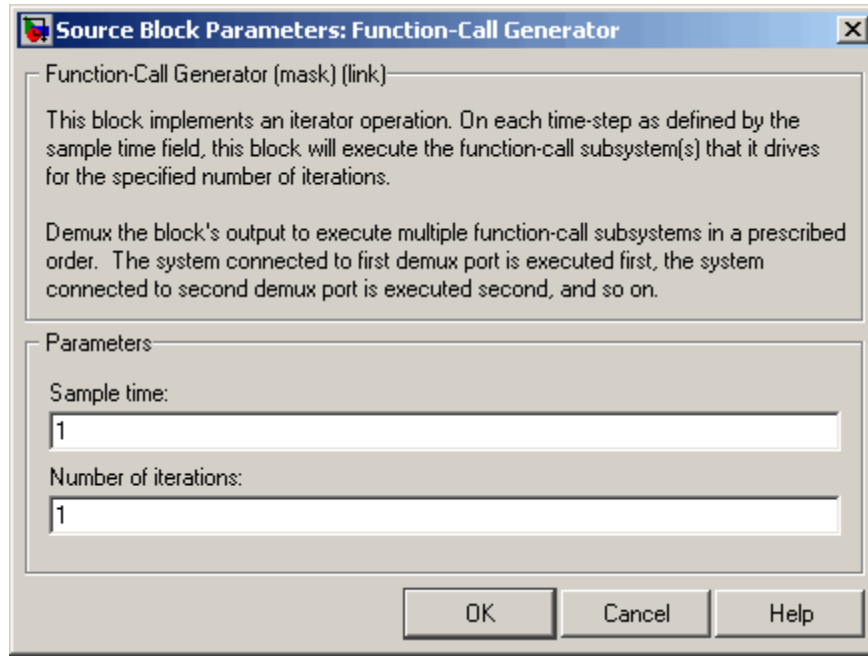


The Function-Call Generator block executes a function-call subsystem (for example, a Stateflow® chart acting as a function-call subsystem) at the rate you specify for the **Sample time** parameter. To execute multiple function-call subsystems in a prescribed order, first connect a Function-Call Generator block to a Demux block that has as many output ports as there are function-call subsystems to control. Then connect the output ports of the Demux block to the systems to control. The system connected to the first demux port executes first, the system connected to the second demux port executes second, and so on.

Data Type Support

The Function-Call Generator block outputs a signal of type `fcn_call`.

Parameters and Dialog Box



Sample time

The time interval between samples. See "How to Specify the Sample Time" in the online documentation for more information.

Number of iterations

Number of times to execute the block per time step. The value of this parameter can be a vector where each element of the vector specifies a number of times to execute a function-call subsystem. The total number of times that a function-call subsystem executes per time step equals the sum of the values of the elements of the generator signal entering its control port.

Suppose that you specify the number of iterations to be [2 2] and connect the output of this block to the control port of a function-call subsystem. In this case, the function-call subsystem executes four times at each time step.

Characteristics

Direct Feedthrough	No
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	Yes
Zero-Crossing Detection	No

Function-Call Subsystem

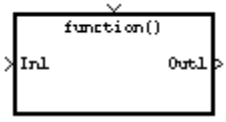
Purpose

Represent subsystem that can be invoked as function by another block

Library

Ports & Subsystems

Description



The Function-Call Subsystem block is a Subsystem block that is preconfigured to serve as a starting point for creating a function-call subsystem. For more information, see “Function-Call Subsystems” in the “Creating a Model” chapter of the Simulink documentation.

Purpose

Multiply input by constant

Library

Math Operations

Description

The Gain block multiplies the input by a constant value (gain). The input and the gain can each be a scalar, vector, or matrix.

You specify the value of the gain in the **Gain** parameter. The **Multiplication** parameter lets you specify element-wise or matrix multiplication. For matrix multiplication, this parameter also lets you indicate the order of the multiplicands.

The gain is converted from doubles to the data specified in the block mask offline using round-to-nearest and saturation. The input and gain are then multiplied, and the result is converted to the output data type using the specified rounding and overflow modes.

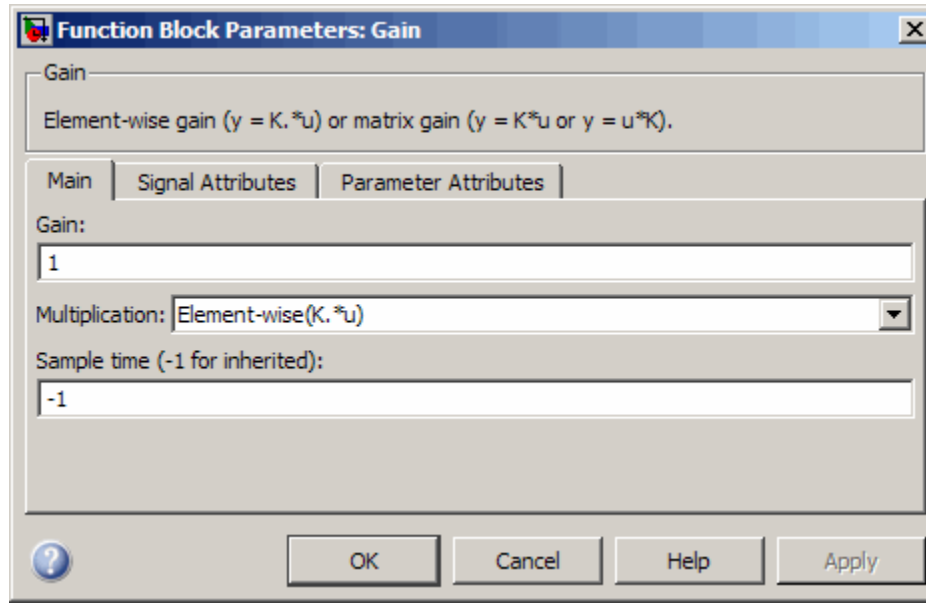
Data Type Support

The Gain block accepts a real or complex scalar, vector, or matrix of any numeric data type supported by Simulink software. The Gain block supports fixed-point data types. If the input of the Gain block is real and the gain is complex, the output is complex.

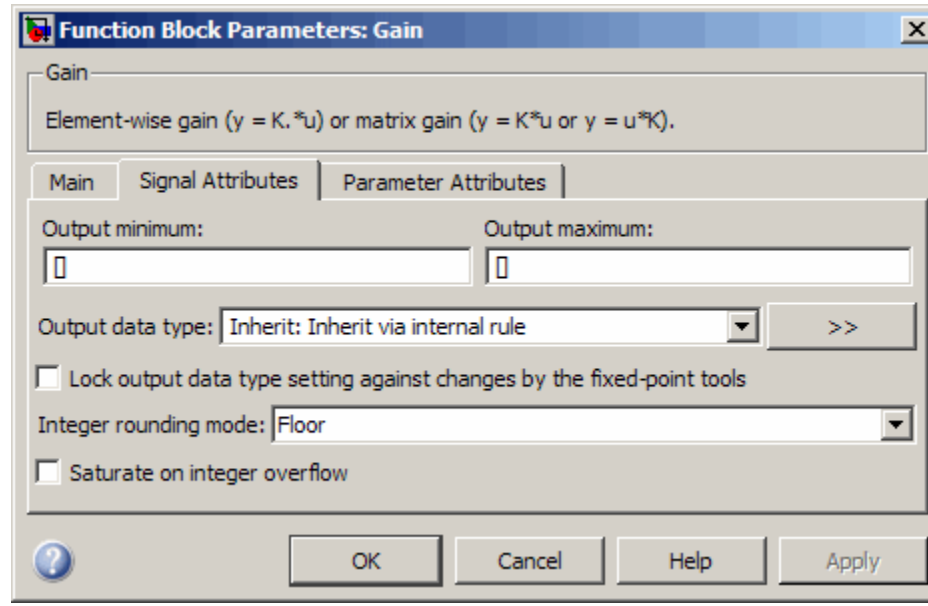
For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

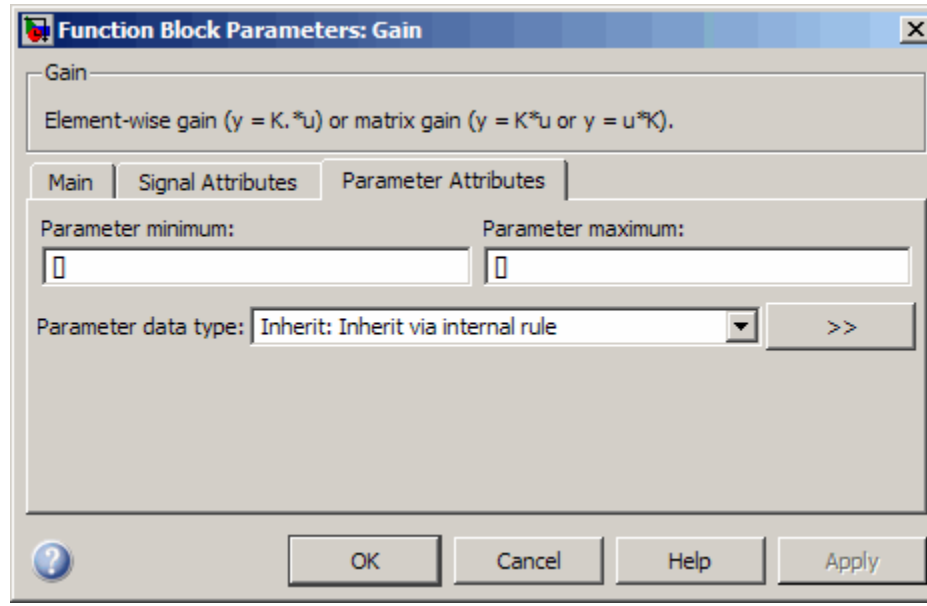
The **Main** pane of the Gain block dialog box appears as follows:



The **Signal Attributes** pane of the Gain block dialog box appears as follows:



The **Parameter Attributes** pane of the Gain block dialog box appears as follows:


Function Block Parameters: Gain
Gain
Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).
Main
Signal Attributes
Parameter Attributes
Parameter minimum:
Parameter maximum:
Parameter data type:
Inherit: Inherit via internal rule
>>
OK
Cancel
Help
Apply

Gain

Specify the value by which to multiply the input.

Settings

Default: 1

Minimum: value of **Parameter minimum** parameter

Maximum: value of **Parameter maximum** parameter

The gain may be a scalar, vector, or matrix.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Multiplication

Specify the multiplication mode.

Settings

Default: `Element-wise(K.*u)`

`Element-wise(K.*u)`
: Each element of the input is multiplied by each element of the gain. The block performs expansions, if necessary, so that the input and gain have the same dimensions.

`Matrix(K*u)`
: The input and gain are matrix multiplied with the input as the second operand.

`Matrix(u*K)`
: The input and gain are matrix multiplied with the input as the first operand.

`Matrix(K*u)(u vector)`
: The input and gain are matrix multiplied with the input as the second operand. This mode is identical to `Matrix(K*u)`, except for how dimensions are determined.

  Suppose that `K` is an `m-by-n` matrix. `Matrix(K*u)(u vector)` sets the input to a vector of length `n` and the output to a vector of length `m`. In contrast, `Matrix(K*u)` uses propagation to determine dimensions for the input and output. For an `m-by-n` gain matrix, the input can propagate to an `n-by-q` matrix, and the output becomes an `m-by-q` matrix.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Specify the time interval between samples.

Settings

Default: -1

To inherit the sample time, set this parameter to `-1`.

See "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `Inherit: Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block), `Inherit: Inherit from 'Constant value'` (Constant block), `Inherit: Inherit via back propagation` (Data Type Conversion block), `Inherit: Same as input` (Saturation block)

`Inherit: Inherit via internal rule`
: This option appears for some blocks. Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the calculated 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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 software displays an error message in the Simulation Diagnostics Viewer.

`Inherit: Inherit from 'Constant value'`
: Use data type of **Constant value** parameter. This option appears for some blocks.

`Inherit: Inherit via back propagation`
: Use data type of the driving block.

`Inherit: Same as input`
: Use data type of sole input signal. This option appears for some blocks.

`Inherit: Same as first input`
: Use data type of first input signal. This option appears for some blocks.

`Inherit: Same as accumulator`
: Output data type is the same as accumulator data type. This option appears for some blocks.

`double`
: Output data type is `double`.

`single`
: Output data type is `single`.

`int8`
: Output data type is `int8`.

`uint8`
: Output data type is `uint8`.

`int16`
: Output data type is `int16`.

`uint16`
: Output data type is `uint16`.

`int32`
: Output data type is `int32`.

`uint32`
: Output data type is `unt32`.

`fixdt(1,16,0)`
: Output data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Output data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
: Use an enumerated data type, for example, `Enum: BasicColors`. This option appears for some blocks.

`<data type expression>`
: Use a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Floor`

`Ceiling`
: Round both positive and negative numbers toward positive infinity.

`Convergent`
: Round number to the nearest representable value. If a tie occurs, round to the nearest even stored value.

`Floor`
: Round both positive and negative numbers toward negative infinity.

`Nearest`
: Round number to the nearest representable value. If a tie occurs, round toward positive infinity.

`Round`
: Round number to the nearest representable value. If a tie occurs, round positive numbers toward positive infinity and round negative numbers toward negative infinity.

`Simplest`
: This option provides for an optimization of the rounding code for several blocks.

`Zero`
: Round number toward zero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Rounding” in the *Simulink Fixed Point User’s Guide*.

Saturate on integer overflow

Specify whether overflows saturate.

Settings

Default: Off

 On

Overflows saturate.

 Off

Overflows do not saturate.

Tips

- When you select this check box, saturation applies to every internal operation on the block, not just the output or result.
- In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Parameter minimum

Specify the minimum value of the gain.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Automatic scaling of fixed-point data types

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Parameter maximum

Specify the maximum value of the gain.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Automatic scaling of fixed-point data types

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Parameter data type

Specify the data type of the **Gain** parameter.

Settings
Default:

`Inherit: Inherit via internal rule`
A rule that inherits a data type

`Inherit: Same as input`
Use data type of sole input signal

`double`
Data type is `double`

`single`
Data type is `single`

`int8`
Data type is `int8`

`uint8`
Data type is `uint8`

`int16`
Data type is `int16`

`uint16`
Data type is `uint16`

`int32`
Data type is `int32`

`uint32`
Data type is `uint32`

`fixdt(1,16)`
Data type is `fixdt(1,16)`

`fixdt(1,16,0)`
Data type is `fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`
: Data type is `fixdt(1,16,2^0,0)`

`<data type expression>`
: The name of a data type object, for example `Simulink.NumericType`

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<class name>`

`Expression`
: Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

See “Specifying a Fixed-Point Data Type” in the *Simulink User’s Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of input and **Gain** parameter for `Element-wise(K.*u)` multiplication
Dimensionalized	Yes
Multidimensionalized	Yes, only if the **Multiplication** parameter specifies `Element-wise(K.*u)`
Zero Crossing	No

Purpose

Pass block input to From blocks

Library

Signal Routing

Description

The Goto block passes its input to its corresponding From blocks. The input can be a real- or complex-valued signal or vector of any data type. From and Goto blocks allow you to pass a signal from one block to another without actually connecting them.

A Goto block can pass its input signal to more than one From block, although a From block can receive a signal from only one Goto block. The input to that Goto block is passed to the From blocks associated with it as though the blocks were physically connected. Goto blocks and From blocks are matched by the use of Goto tags, defined in the **Tag** parameter.

The **Tag Visibility** parameter determines whether the location of From blocks that access the signal is limited:

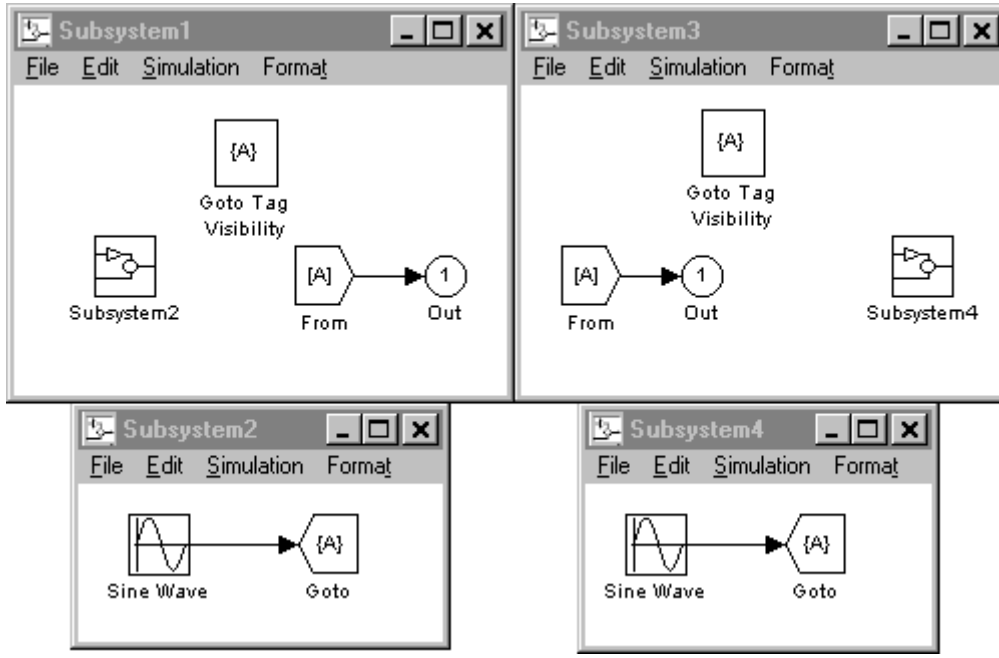
- `local`, the default, means that From and Goto blocks using the same tag must be in the same subsystem. A local tag name is enclosed in brackets ([]).
- `scoped` means that From and Goto blocks using the same tag must be in the same subsystem or at any level in the model hierarchy below the Goto Tag Visibility block that does not entail crossing a nonvirtual subsystem boundary, i.e., the boundary of an atomic, conditionally executed, or function-call subsystem or a model reference. A scoped tag name is enclosed in braces ({}).
- `global` means that From and Goto blocks using the same tag can be anywhere in the model except in locations that span nonvirtual subsystem boundaries.

The rule that From-Goto block connections cannot cross nonvirtual subsystem boundaries has the following exception. A Goto block connected to a state port in one conditionally executed subsystem is visible to a From block inside another conditionally executed subsystem. For more information about conditionally executed subsystems, see

"Creating Conditional Subsystems" in the "Creating a Model" chapter of the Simulink documentation.

Note A `scoped` Goto block in a masked system is visible only in that subsystem and in the nonvirtual subsystems it contains. Simulink software generates an error if you run or update a diagram that has a Goto Tag Visibility block at a higher level in the block diagram than the corresponding `scoped` Goto block in the masked subsystem.

Use local tags when the Goto and From blocks using the same tag name reside in the same subsystem. You must use global or scoped tags when the Goto and From blocks using the same tag name reside in different subsystems. When you define a tag as global, all uses of that tag access the same signal. A tag defined as scoped can be used in more than one place in the model. This example shows a model that uses two scoped tags with the same name (A).

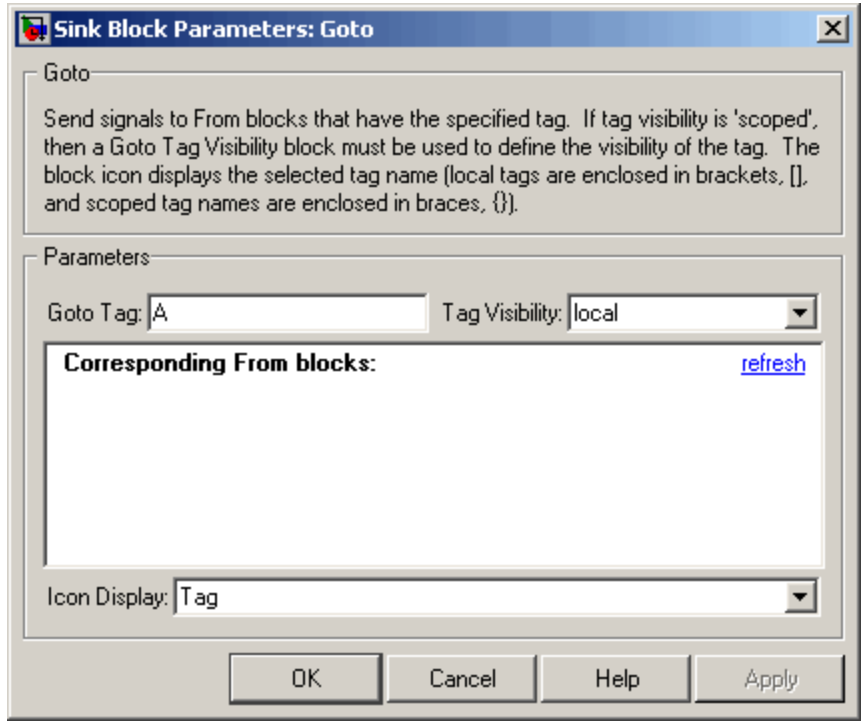


Data Type Support

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

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

Parameters and Dialog Box



Tag

The Goto block identifier. This parameter identifies the Goto block whose scope is defined in this block.

Tag Visibility

The scope of the Goto block tag: `local`, `scoped`, or `global`. The default is `local`.

Corresponding From blocks

List of the From blocks connected to this Goto block. Double-clicking any entry in this list displays and highlights the corresponding From block.

Icon Display

Specifies the text to display on the block's icon. The options are the block's tag, the name of the signal that the block represents, or both the tag and the signal name.

Characteristics

Sample Time	Inherited from driving block
Dimensionalized	Yes
Multidimensionalized	Yes
Virtual	Yes For more information, see "Virtual Blocks" in the Simulink documentation.

Goto Tag Visibility

Purpose

Define scope of Goto block tag

Library

Signal Routing

Description

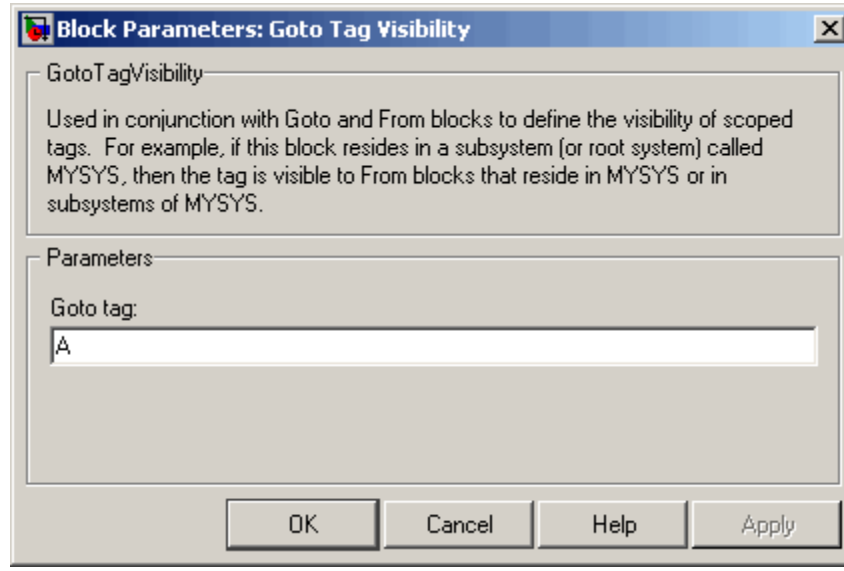
The Goto Tag Visibility block defines the accessibility of Goto block tags that have `scoped` visibility. The tag specified as the **Goto tag** parameter is accessible by From blocks in the same subsystem that contains the Goto Tag Visibility block and in subsystems below it in the model hierarchy.

A Goto Tag Visibility block is required for Goto blocks whose **Tag Visibility** parameter value is `scoped`. No Goto Tag Visibility block is needed if the tag visibility is either `local` or `global`. The block shows the tag name enclosed in braces ({}).

Data Type Support

Not applicable.

Parameters and Dialog Box



Goto tag
The Goto block tag whose visibility is defined by the location of this block.

Characteristics

Sample Time	N/A
Dimensionalized	N/A
Virtual	Yes For more information, see "Virtual Blocks" in the Simulink documentation.

Ground

Purpose

Ground unconnected input port

Library

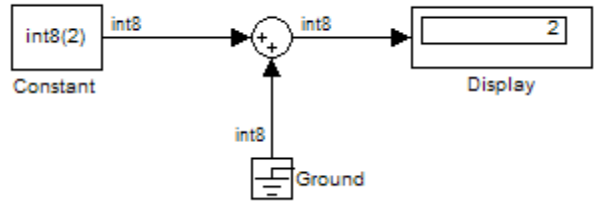
Sources

Description

The Ground block can connect blocks whose input ports do not connect to other blocks. If you run a simulation with blocks having unconnected input ports, Simulink software issues warnings. Using Ground blocks to ground those unconnected blocks can prevent warnings. The Ground block outputs a signal with zero value.

Data Type Support

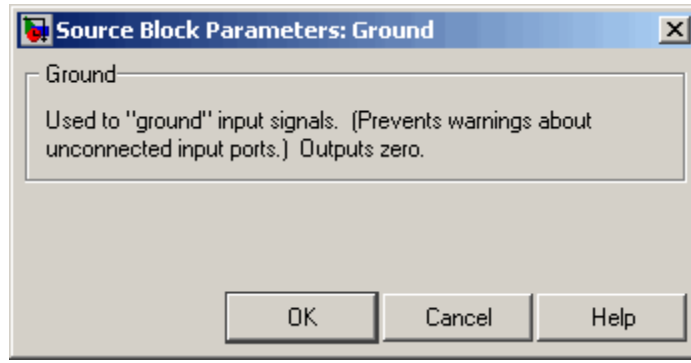
The Ground block outputs a signal of the same numeric data type as the port to which it connects. For example, consider the following model:



In this example, the output of the Constant block determines the data type (`int8`) of the port to which the Ground block is connected. That port in turn determines the type of the signal output by the Ground block.

The Ground block supports all data types supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Sample Time	Inherited from driven block
Dimensionalized	Yes
Multidimensionalized	Yes
Virtual	Yes For more information, see "Virtual Blocks" in the Simulink documentation.

Hit Crossing

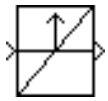
Purpose

Detect crossing point

Library

Discontinuities

Description



The Hit Crossing block detects when the input reaches the **Hit crossing offset** parameter value in the direction specified by the **Hit crossing direction** property.

The block accepts one input of type `double`. If you select the **Show output port** check box, the block output indicates when the crossing occurs. If the input signal is exactly the value of the offset value after the hit crossing is detected, the block continues to output a value of 1. If the input signals at two adjacent points bracket the offset value (but neither value is exactly equal to the offset), the block outputs a value of 1 at the second time step. If the **Show output port** check box is *not* selected, the block ensures that the simulation finds the crossing point but does not generate output. If the input signal is constant and equal to the offset value, the block outputs 1 only if the **Hit crossing direction** property is set to `either`.

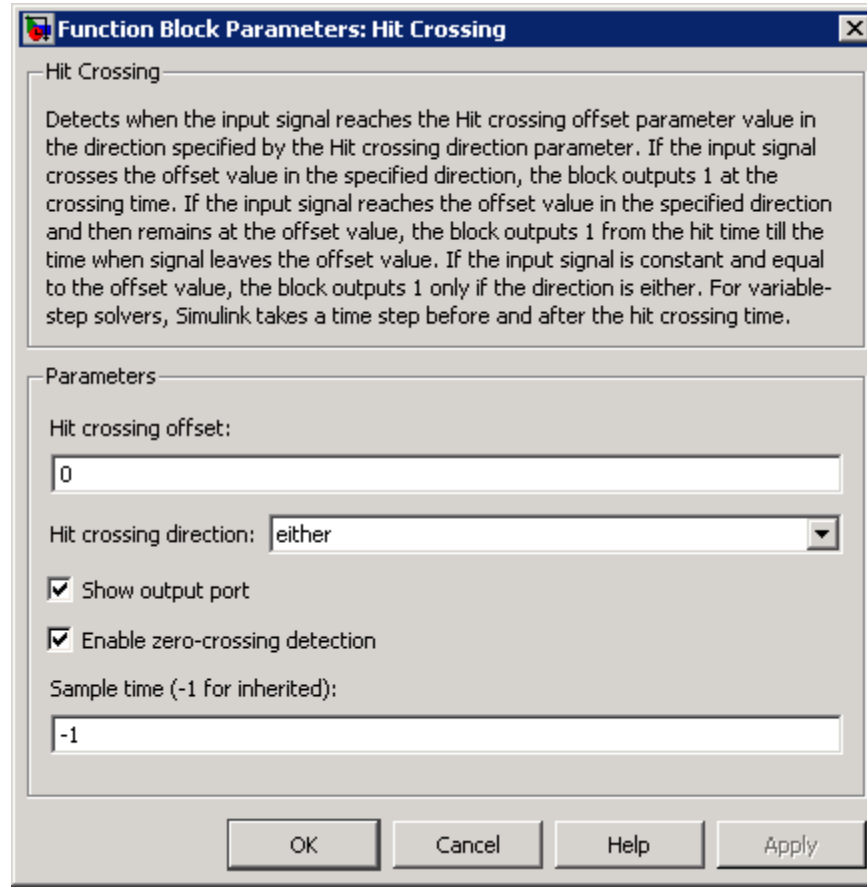
When the block's **Hit crossing direction** property is set to `either`, the block serves as an "Almost Equal" block, useful in working around limitations in finite mathematics and computer precision. Used for these reasons, this block might be more convenient than adding logic to your model to detect this condition.

The `hardstop` and `sldemo_clutch` demos illustrate the use of the Hit Crossing block. In the `hardstop` demo, the Hit Crossing block is in the Friction Model subsystem. In the `sldemo_clutch` demo, the Hit Crossing block is in the Friction Mode Logic/Lockup Detection subsystem.

Data Type Support

The Hit Crossing block outputs a signal of type `Boolean` if Boolean logic signals are enabled (see "Implement logic signals as Boolean data (vs. double)"). Otherwise, the block outputs a signal of type `double`.

Parameters and Dialog Box



Hit crossing offset
The value whose crossing is to be detected.

Hit crossing direction
The direction from which the input signal approaches the hit crossing offset for a crossing to be detected.

Show output port
If selected, draw an output port.

Enable zero-crossing detection

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

Sample time (-1 for inherited)

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

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes
Dimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled.

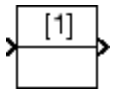
Purpose

Set initial value of signal

Library

Signal Attributes

Description



The IC block sets the initial condition of the signal at its input port, e.g., the value of the signal at the simulation start time (t_{start}). The block does this by outputting the specified initial condition when you start the simulation, regardless of the actual value of the input signal. Thereafter, the block outputs the actual value of the input signal.

Note If an IC block inherits or specifies a nonzero sample time offset (t_{offset}), the IC block outputs its initial value at time `t`,

$$t = n * t_{period} + t_{offset}$$

where `n` is the smallest integer such that $t \geq t_{start}$.

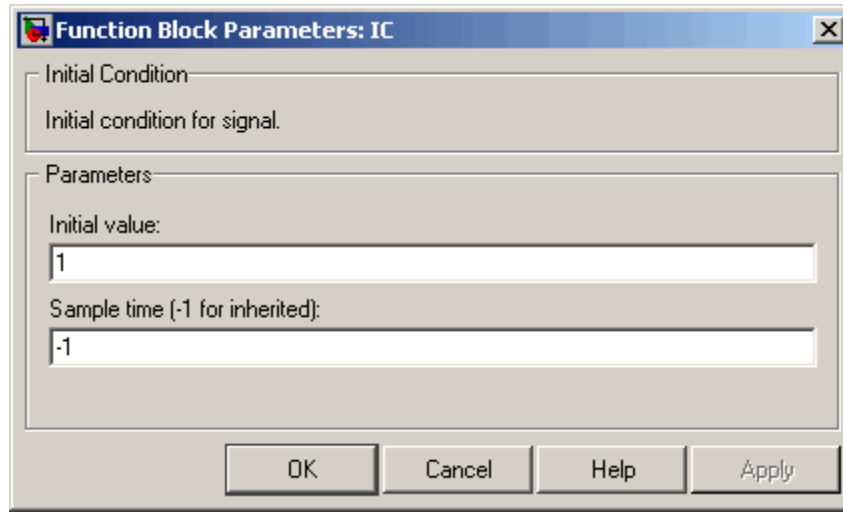
That is, the IC block outputs its initial value the first time blocks with sample time $[t_{period}, t_{offset}]$ execute, which can be after t_{start}.

The IC block is useful for providing an initial guess for the algebraic state variables in a loop. For more information, see "Algebraic Loops" in the "How Simulink Works" chapter of *Simulink User's Guide*.

Data Type Support

The IC block accepts and outputs signals of any Simulink built-in and fixed-point data type. The **Initial value** parameter accepts any built-in data type supported by Simulink software.

Parameters and Dialog Box



Initial value

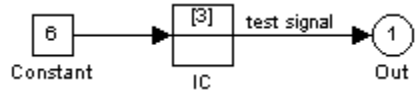
Specify the initial value for the input signal.

Sample time (-1 for inherited)

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

Examples

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



At t = 0, the signal value is 3. Afterward, 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
Multidimensionalized	Yes
Zero Crossing	No

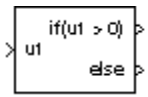
Purpose

Model `if-else` control flow

Library

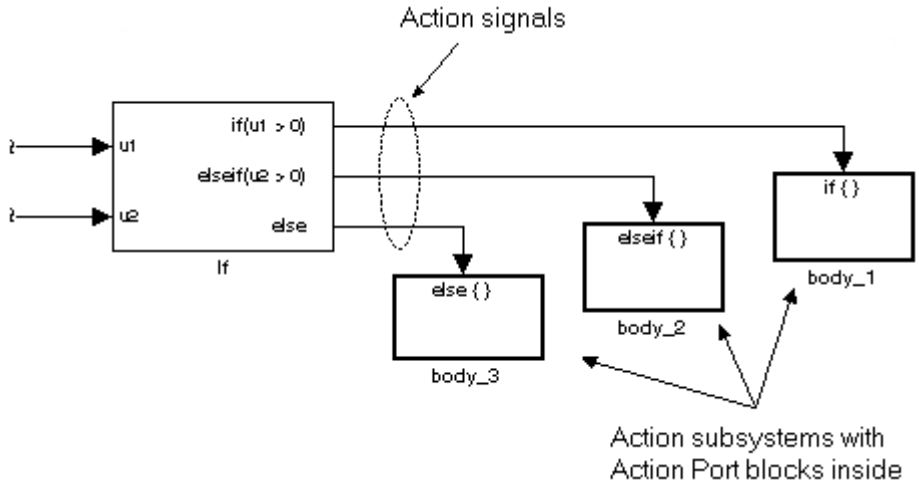
Ports & Subsystems

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.

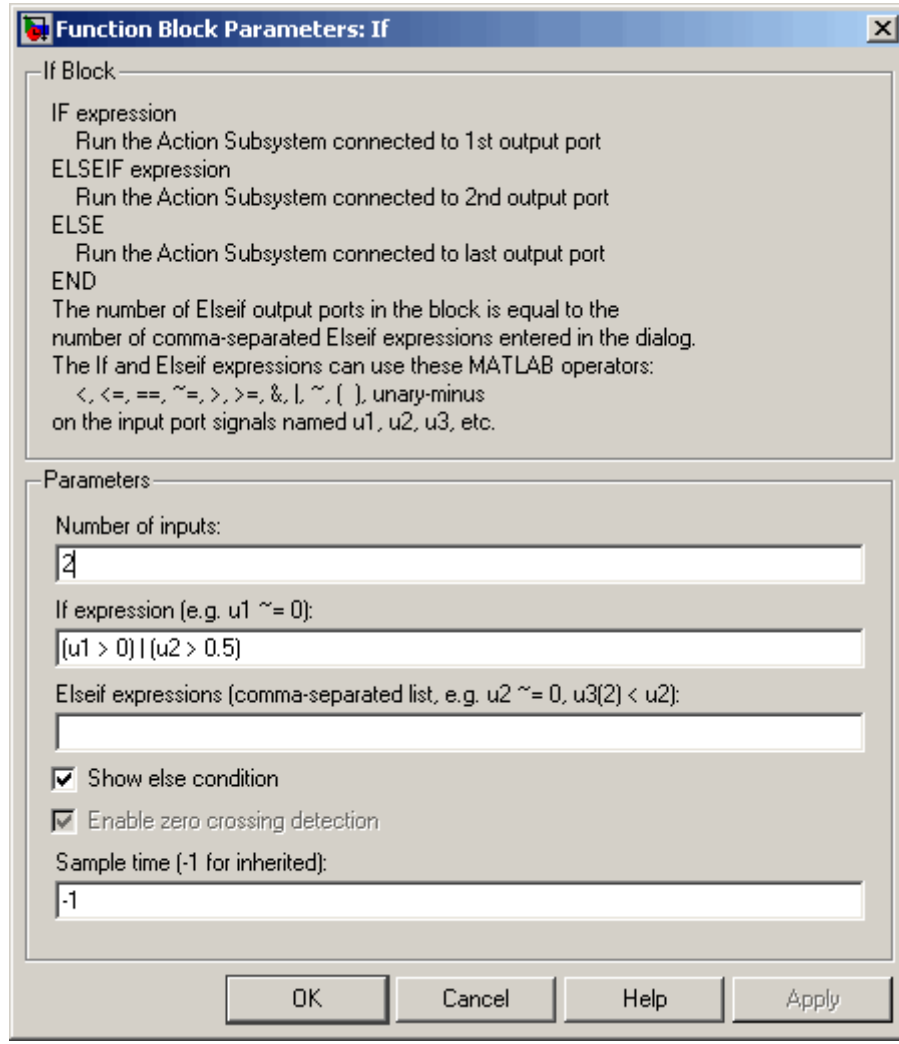
Note All blocks in an If Action Subsystem must run at the same rate as the driving If block. You can achieve this by setting each block's sample time parameter to be either inherited (`-1`) or the same value as the If block's sample time.

In the preceding example, the If Action subsystems are named `body_1`, `body_2`, and `body_3`.

Data Type Support

Inputs `u1,u2,...,un` can be scalars or vectors of any built-in Simulink data type and must all be of the same data type. The inputs cannot be of any user-defined type, such as an enumerated type. Outputs from the `if`, `else`, and `elseif` ports are Action signals to If Action subsystems that you create by using Action Port blocks and subsystems.

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.

If expression

The condition for the `if` output port. This condition appears on the If block adjacent to the `if` output port. The `if` expression can use any of the following operators: `<. <=, ==, ~=, >, >=, &, |, ~, (), unary-minus`. The If Action subsystem attached to the `if` port executes if its condition is true. The expression must not contain data type expressions, e.g., `int8(6)`, and must not reference workspace variables whose data type is other than `double` or `single`.

Note You cannot tune the **If expression** during accelerated-mode simulation (see "Accelerating Models"), in referenced models executing in Accelerator mode, or in code generated from the model. The If block also does not support custom storage classes.

Elseif expressions

A string list of `elseif` conditions delimited by commas. These conditions appear below the `if` port and above the `else` port if you select the **Show else condition** check box. `elseif` expressions can use any of the following operators: `<, <=, ==, ~=, >, >=, &, |, ~, (), unary-minus`. The If Action subsystem attached to an `elseif` port executes if its condition is true and all of the `if` and `elseif` conditions are false. The expression must not contain data type expressions, e.g., `int8(6)`, and must not reference workspace variables whose data type is other than `double` or `single`.

Note You cannot tune the **Elseif expression** during accelerated-mode simulation (see "Accelerating Models"), in referenced models executing in Accelerator mode, or in code generated from the model. The If block also does not support custom storage classes.

Show else condition
If you select this check box, an `else` port is created. The If Action subsystem attached to the `else` port executes if the `if` port and all the `elseif` ports are false.

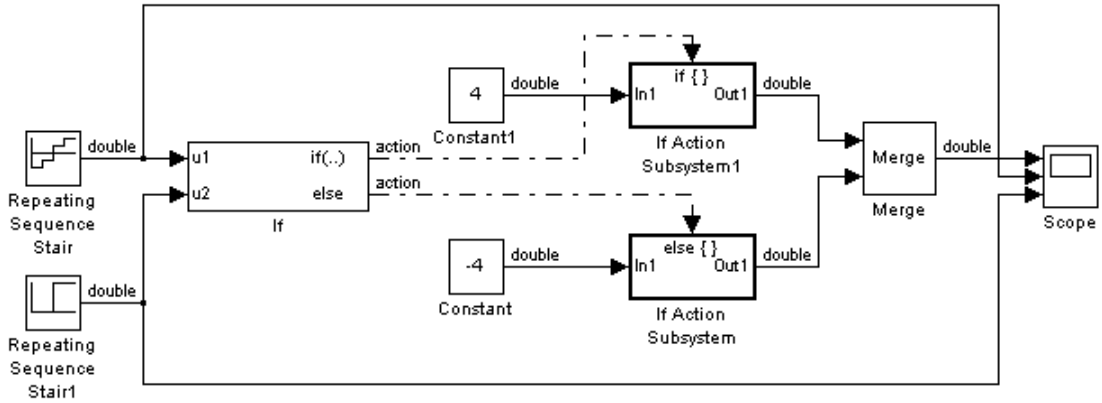
Enable zero-crossing detection
Select to enable zero-crossing detection. For more information, see "Zero-Crossing Detection" in the Simulink documentation.

Sample time
Specify the sample time of the input signal. See "How to Specify the Sample Time" in the online documentation for more information.

Examples

The If block does not directly support fixed-point data types. However, you can use the Compare To Constant block to work around this limitation.

For example, consider the following floating-point model.

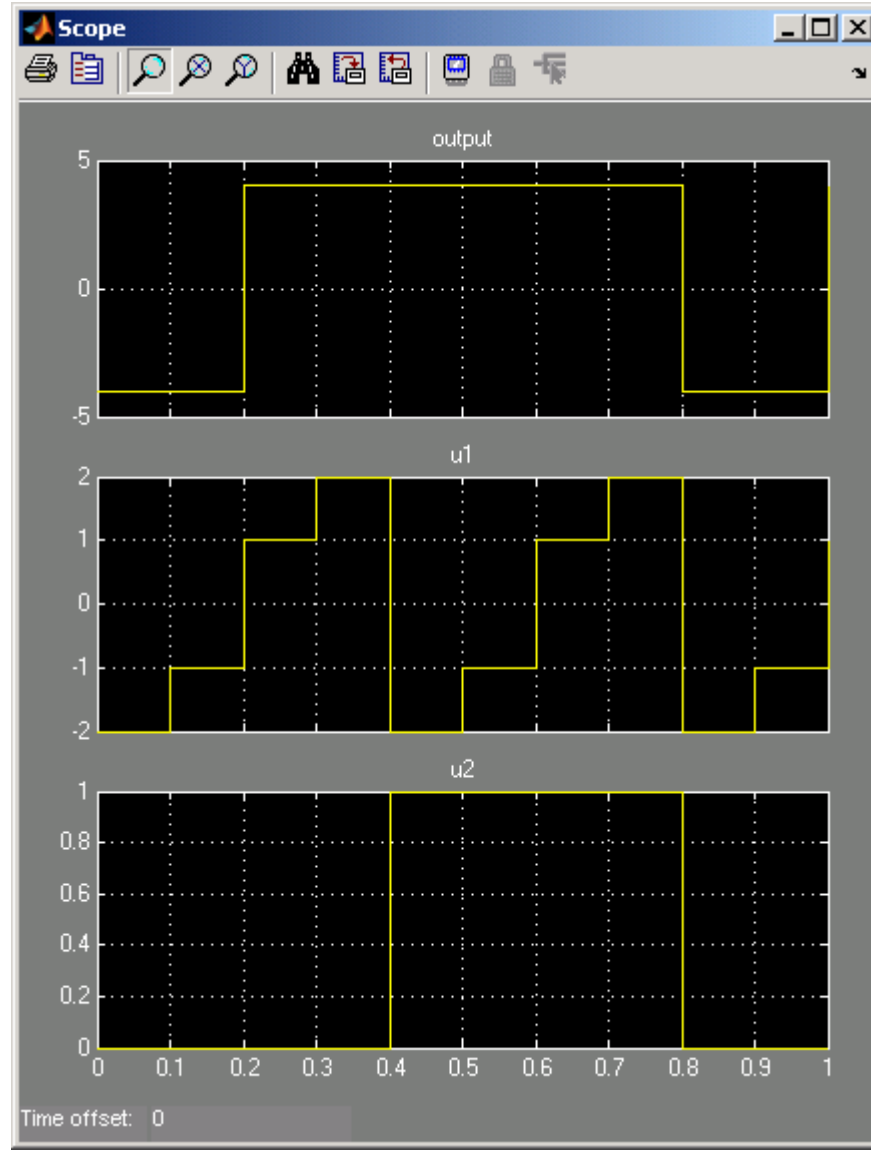


In this model, the If Action subsystems use their default configurations. The block and simulation parameters for the model are set to their default values except as follows:

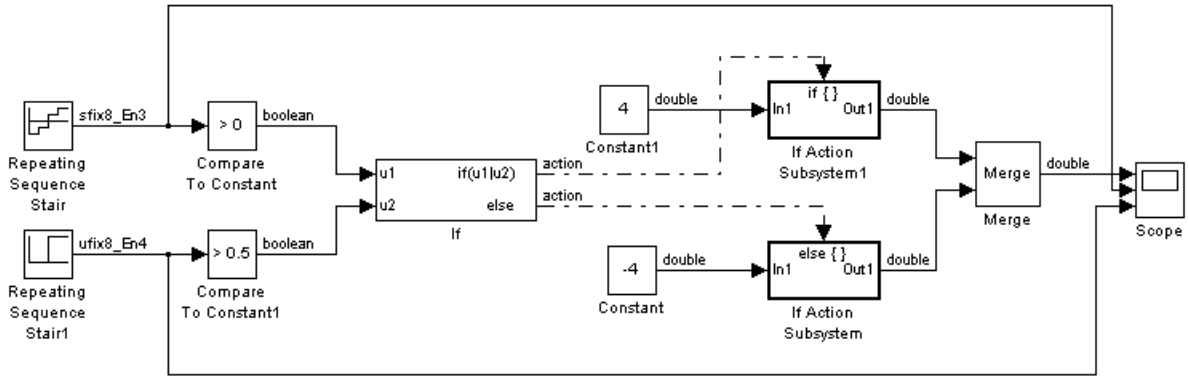
Block or Dialog Box	Parameter	Setting
Configuration Parameters — **Solver** pane	**Start time**	`0.0`
	Stop time	`1.0`
	Type	`Fixed-step`
	Solver	`Discrete (no continuous states)`
	Fixed-step size	`0.1`
Repeating Sequence Stair	**Vector of output values**	`[-2 -1 1 2].'`
Repeating Sequence Stair1	**Vector of output values**	`[0 0 0 0 1 1 1 1].'`
If	**Number of inputs**	`2`

Block or Dialog Box	Parameter	Setting
	If expression	`(u1 > 0) | (u2 > 0.5)`
	Show else condition	selected
Constant	**Constant value**	`-4`
Constant1	**Constant value**	`4`
Scope	**Number of axes**	`3`
	Time range	`1`

For this model, if input `u1` is greater than 0 or input `u2` is greater than 0.5, the output is 4. Otherwise, the output is -4. The Scope block shows the output, `u1`, and `u2`:


Scope
output
5
0
-5
u1
2
1
0
-1
-2
u2
1
0.8
0.6
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Time offset: 0

You can implement the same model with fixed-point data types:



The Repeating Sequence Stair blocks now output fixed-point data types.

The Compare To Constant blocks implement two parts of the **If expression** that is used in the If block in the floating-point version of the model, `(u1 > 0)` and `(u2 > 0.5)`. The `OR` operation, `(u1|u2)`, can still be implemented inside the If block. For a fixed-point model, the expression must be partially implemented outside of the If block as it is here.

The block and simulation parameters for the fixed-point model are the same as for the floating-point model with the following exceptions and additions:

Block	Parameter	Setting
Compare To Constant	**Operator**	`>`
	Constant value	`0`
	Output data type mode	`Boolean`

Block	Parameter	Setting
	Enable zero-crossing detection	off
Compare To Constant1	**Operator**	`>`
	Constant value	`0.5`
	Output data type mode	`Boolean`
	Enable zero-crossing detection	off
If	**Number of inputs**	`2`
	If expression	`u1|u2`

Characteristics

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

If Action Subsystem

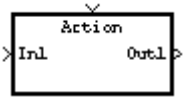
Purpose

Represent subsystem whose execution is triggered by If block

Library

Ports & Subsystems

Description



The If Action Subsystem block is a Subsystem block that is preconfigured to serve as a starting point for creating a subsystem whose execution is triggered by an If block.

Note All blocks in an If Action Subsystem must run at the same rate as the driving If block. You can achieve this by setting each block's sample time parameter to be either inherited (`-1`) or the same value as the If block's sample time.

For more information, see the If block and Modeling with Control Flow Blocks in the "Creating a Model" chapter of the Simulink documentation.

Purpose

Increase real world value of signal by one

Library

Additional Math & Discrete / Additional Math: Increment - Decrement

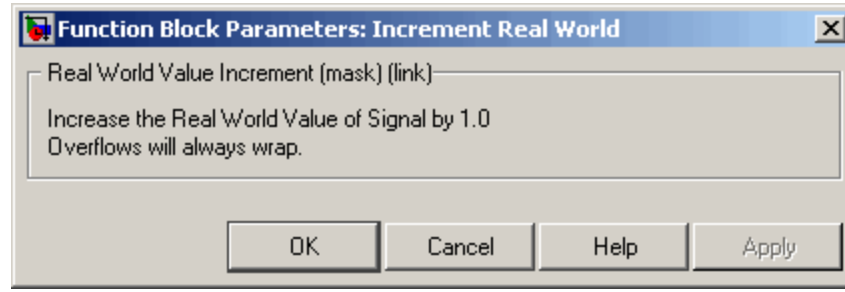
Description

The Increment Real World block increases the real world value of the signal by one. Overflows always wrap.

Data Type Support

The Increment Real World block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	No

See Also

Decrement Real World, Increment Stored Integer

Increment Stored Integer

Purpose

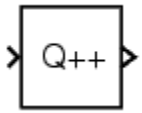
Increase stored integer value of signal by one

Library

Additional Math & Discrete / Additional Math: Increment - Decrement

Description

The Increment Stored Integer block increases the stored integer value of a signal by one.

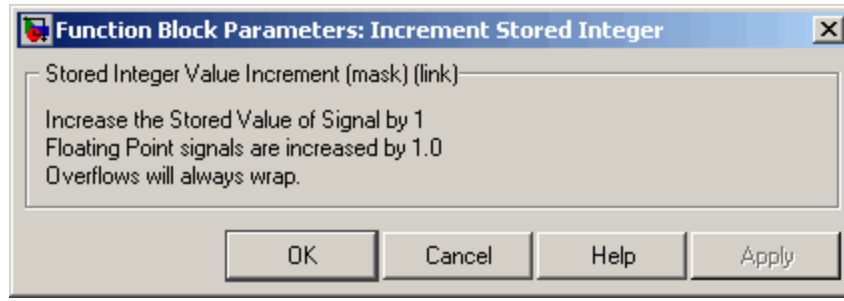


Floating-point signals are also increased by one, and overflows always wrap.

Data Type Support

The Increment Stored Integer block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	No

See Also

Decrement Stored Integer, Increment Real World

Purpose

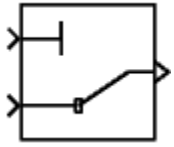
Switch output between different inputs based on value of first input

Library

Signal Routing

Description

The Index Vector block is an implementation of the Multiport Switch block. See Multiport Switch for more information.



Inport

Purpose

Create input port for subsystem or external input

Library

Ports & Subsystems, Sources

Description

Inport blocks are the links from outside a system into the system.

Simulink software assigns Inport block port numbers according to these rules:

- It automatically numbers the Inport blocks within a top-level system or subsystem sequentially, starting with 1.
- If you add an Inport block, the label is the next available number.
- If you delete an Inport block, other port numbers are automatically renumbered to ensure that the Inport blocks are in sequence and that no numbers are omitted.
- If you copy an Inport block into a system, its port number is *not* renumbered unless its current number conflicts with an Inport block already in the system. If the copied Inport block port number is not in sequence, renumber the block. Otherwise, you get an error message when you run the simulation or update the block diagram.

You can specify the dimensions of the input to the Inport block using the **Port dimensions** parameter. Entering a value of `-1` lets the Simulink software determine the port dimension.

The **Sample time** parameter is the rate at which the signal is coming into the system. A value of `-1` causes the block to inherit its sample time from the block driving it. You might need to set this parameter for

- Inport blocks in a top-level system
- Models with blocks where the Simulink software cannot determine the sample time, but these blocks drive Inport blocks.

For more information, see “How to Specify the Sample Time” in the *Simulink User’s Guide*.

Inport Blocks in a Top-Level System

Inport blocks in a top-level system have two uses:

- To supply external inputs from the workspace, use the Configuration Parameters dialog box (see "Importing Data from a Workspace") or the `ut` argument of the `sim` command (see `sim`) to specify the inputs.
- To provide a means for perturbation of the model by the `linmod` and `trim` analysis functions, use Inport blocks to inject inputs are injecte into the system.

Limitation: You cannot use a top-level (root) inport to supply fixed-point data that is contained in a structure. Consider using a `Simulink.Timeseries` object instead of a structure.

Inport Blocks in a Subsystem

Inport blocks in a subsystem represent inputs to the subsystem. A signal arriving at an input port on a Subsystem block flows out of the associated Inport block in that subsystem. The Inport block associated with an input port on a Subsystem block is the block whose **Port number** parameter matches the relative position of the input port on the Subsystem block. For example, the Inport block whose **Port number** parameter is `1` gets its signal from the block connected to the topmost port on the Subsystem block.

If you renumber the **Port number** of an Inport block, the block becomes connected to a different input port, although the block continues to receive its signal from the same block outside the subsystem.

The Inport block name appears in the Subsystem icon as a port label. To suppress display of the label, select the Inport block and choose **Format > Hide Name**.

You can use a subsystem inport to supply fixed-point data in a structure or any other format.

Creating Duplicate Inports

You can create any number of duplicates of an Inport block. The duplicates are graphical representations of the original intended to simplify block diagrams by eliminating unnecessary lines. The duplicate has the same port number, properties, and output as the original. Changing properties of a duplicate changes properties of the original and vice versa.

To create a duplicate of an Inport block:

1. Select the block.
2. Select **Edit > Copy** in the Model Editor.
3. Place your pointer in the block diagram where you want to create the duplicate.
4. Select **Edit > Paste Duplicate Inport** in the Model Editor.

Connecting Buses to Root Level Inports

If you want a root level Inport of a model to produce a bus signal, you must select the Inport's **Specify properties via bus object** parameter and set the Inport's **Bus object for validating input bus** parameter to the name of a bus object that defines the bus that the Inport produces. If the bus contains mixed data types, the Inport's data type must be `auto`. See "Using Bus Objects" in the *Simulink User's Guide* for more information.

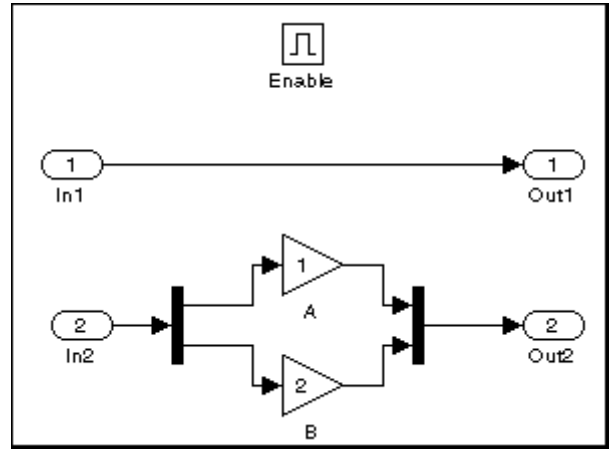
Data Type Support

The Inport block accepts complex or real signals of any data type supported by Simulink software, including fixed-point and enumerated data types. For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink".

Limitation: You cannot use a top-level (root) inport to supply fixed-point data that is contained in a structure. Consider using a `Simulink.Timeseries` object instead of a structure.

The numeric and data types of the block's output are the same as those of its input. You can specify the signal type, data type, and sampling mode of an external input to a root-level Inport block using the **Signal type**, **Data type**, and **Sampling mode** parameters.

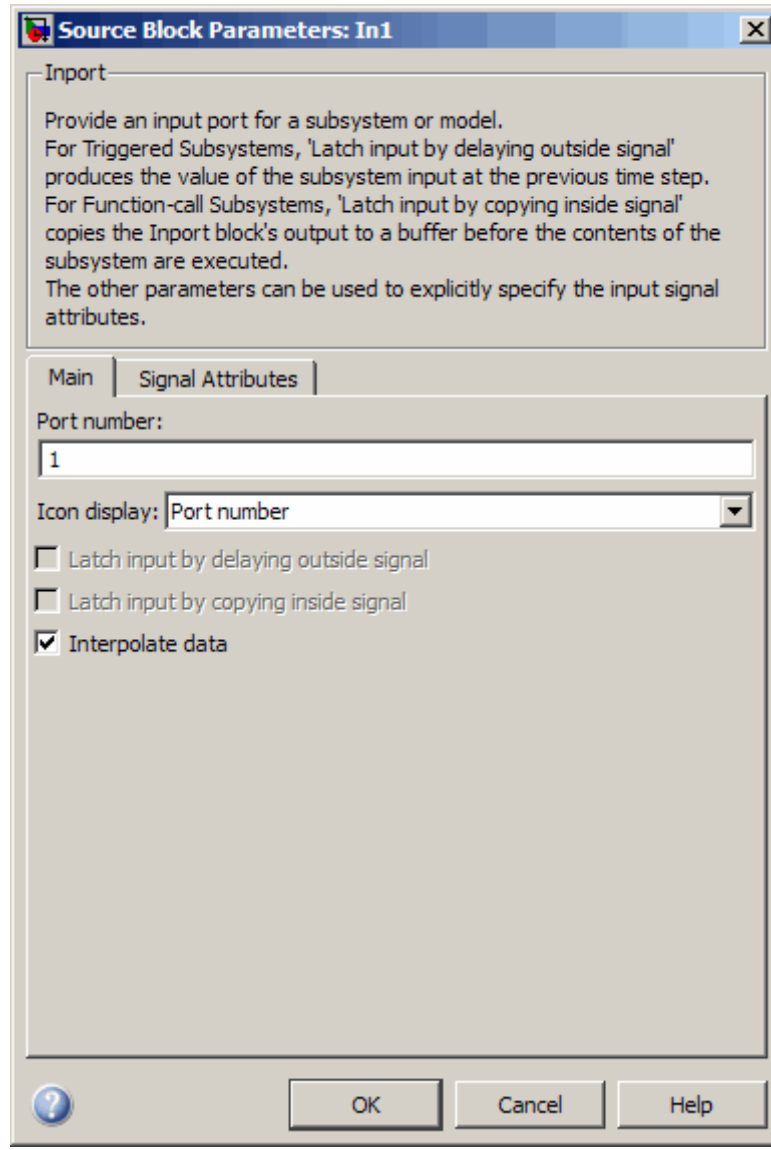
The elements of a signal array connected to a root-level Inport block must be of the same numeric and data types. Signal elements connected to a subsystem input port can be of differing numeric and data types except in the following circumstance: If the subsystem contains an Enable or Trigger block or is an Atomic Subsystem and the input port, or an element of the input port, is connected directly to an output port, the input elements must be of the same type. For example, consider the follow enabled subsystem.



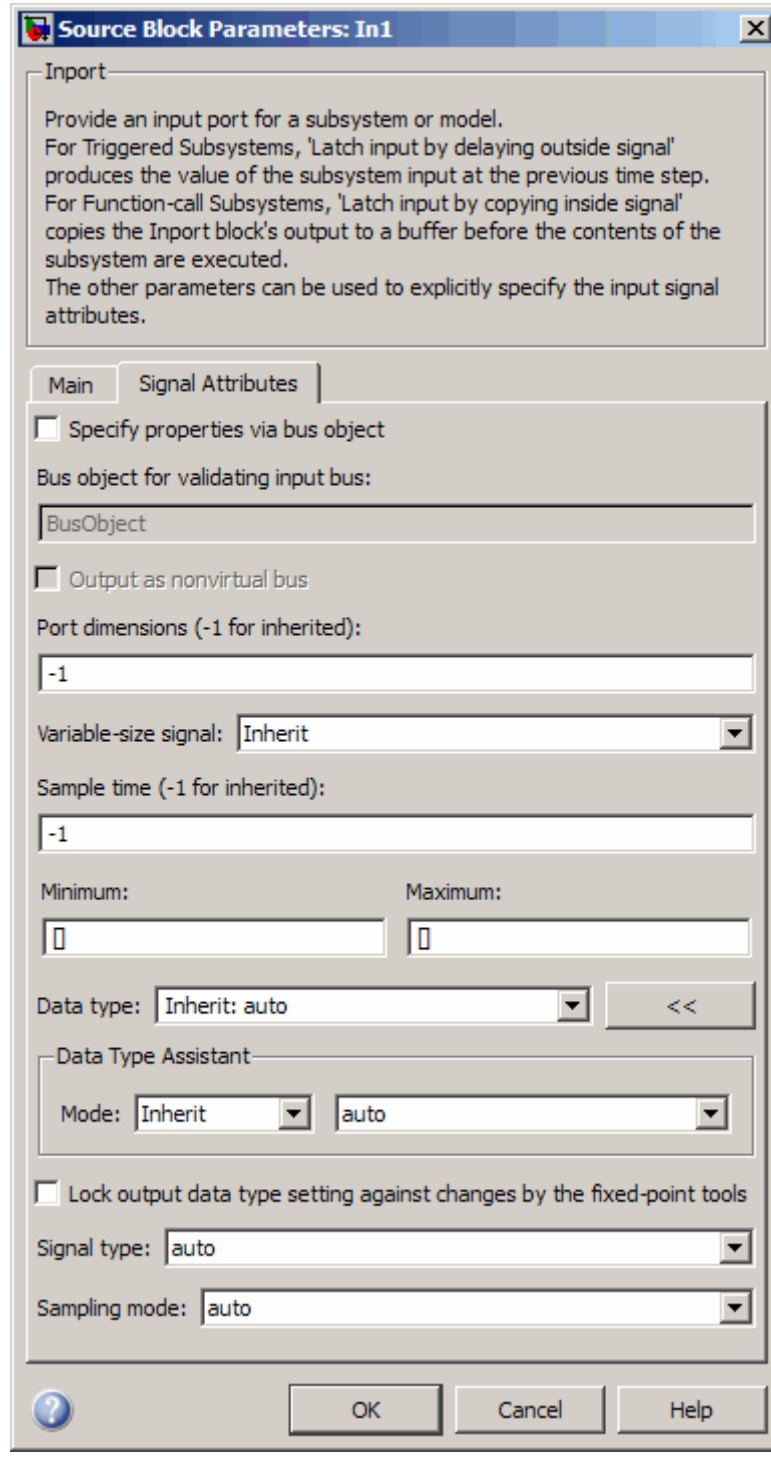
In this example, the elements of a signal vector connected to `In1` must be of the same type. The elements connected to `In2`, however, can be of differing types.

Parameters and Dialog Box

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



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


Source Block Parameters: In1
Inport
Provide an input port for a subsystem or model.
For Triggered Subsystems, 'Latch input by delaying outside signal'
produces the value of the subsystem input at the previous time step.
For Function-call Subsystems, 'Latch input by copying inside signal'
copies the Inport block's output to a buffer before the contents of the
subsystem are executed.
The other parameters can be used to explicitly specify the input signal
attributes.
Main
Signal Attributes
Specify properties via bus object
Bus object for validating input bus:
BusObject
Output as nonvirtual bus
Port dimensions (-1 for inherited):
-1
Variable-size signal:
Inherit
Sample time (-1 for inherited):
-1
Minimum:
Maximum:
Data type:
Inherit: auto
<<
Data Type Assistant
Mode:
Inherit
auto
Lock output data type setting against changes by the fixed-point tools
Signal type:
auto
Sampling mode:
auto
OK
Cancel
Help

- “Bias” on page 2-1321

Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Port number

Specify the port number of the block.

Settings

Default: 1

This parameter controls the order in which the port that corresponds to the block appears on the parent subsystem or model block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Icon display

Specify the information to be displayed on the icon of this input port.

Settings

Default: `Port number`

`Signal name`
: Display the name of the signal connected to this port (or signals if the input is a bus).

`Port number`
: Display port number of this port.

`Port number and signal name`
: Display both the port number and the names of the signals connected to this port.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Latch input by delaying outside signal

Output the value of the input signal at the previous time step.

Settings

Default: Off

☑ On

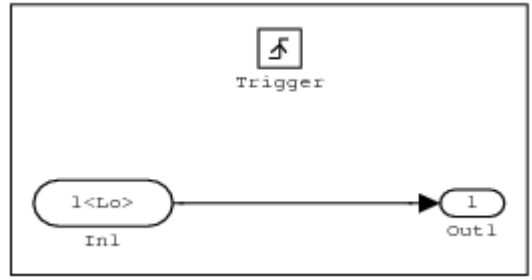
Output the value of the input signal at the previous time step.

☐ Off

Do not output the value of the input signal at the previous time step.

Tips

- This option applies only to triggered subsystems and is enabled only if the Inport block resides in a triggered subsystem.
- Selecting this check box enables Simulink software to resolve data dependencies among triggered subsystems that are part of a loop.
- Type `sl_subsys_semantics` at the MATLAB prompt for examples using latched inputs with triggered subsystems.
- The Inport block indicates that this option is selected by displaying <Lo>.



Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Latch input by copying inside signal

Copy the block's signal output into a buffer before executing the contents of the subsystem and use this copy as the block's output during execution of the subsystem.

Settings

Default: Off

☑ On

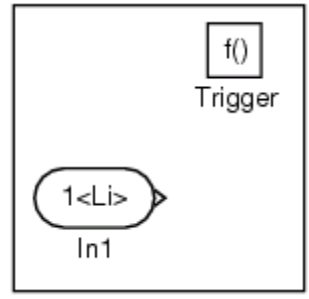
Copy the block's signal output into a buffer before executing the contents of the subsystem and use this copy as the block's output during execution of the subsystem.

☐ Off

Do not copy the block's signal output into a buffer before executing the contents of the subsystem to use as the block's output during execution of the subsystem.

Tips

- This parameter applies only to function-call subsystems and is enabled only if the Inport block resides in a function-call subsystem.
- This parameter ensures that the subsystem's inputs, including those generated within the subsystem's context, will not change during execution of the subsystem.
- The Inport block indicates that this option is selected by displaying <Li>.



Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Interpolate data

Cause the block to interpolate or extrapolate output at time steps for which no corresponding workspace data exists when loading data from the workspace.

Settings

Default: On

 On

Cause the block to interpolate or extrapolate output at time steps for which no corresponding workspace data exists when loading data from the workspace.

 Off

Do not cause the block to interpolate or extrapolate output at time steps for which no corresponding workspace data exists when loading data from the workspace.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Specify properties via bus object

Use a bus object to define the structure of the bus that is input or output by this block.

Settings

Default: Off

 On

Use a bus object to define the structure of the bus that is input or output by this block.

 Off

Do not use a bus object to define the structure of the bus that is input or output by this block.

Tips

Selecting this parameter is required if the bus is nonvirtual (including a nonvirtual bus that was converted from a virtual bus as described in "Automatic Bus Conversion") and is optional otherwise.

Dependencies

This parameter enables **Bus object for validating input bus**.

This parameter enables **Output as nonvirtual bus in parent model**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Bus object for validating input bus

Specify the name of the bus object that defines the structure that a bus must have to connect to this port.

Settings

Default: `BusObject`

A bus object is an instance of class `Simulink.Bus` that is defined in the base workspace.

Tips

At the beginning of a simulation or when you update the model's diagram, Simulink software checks whether the bus connected to this port has the specified structure. If not, Simulink software displays an error message.

Dependencies

Specify properties via bus object enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output as nonvirtual bus

Output a nonvirtual bus.

Settings

Default: Off

 On

Output a nonvirtual bus.

 Off

Output a virtual bus.

Tips

- 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.
- All signals in a nonvirtual bus must have the same sample time, even if the elements of the associated bus object specify inherited sample times. Any bus operation that would result in a nonvirtual bus that violates this requirement generates an error. Therefore, if you select this option all signals in the bus must have the same sample time. You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus, to allow the signal or bus to be included in a nonvirtual bus.

Dependency

Specify properties via bus object enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Port dimensions (-1 for inherited)

Specify the dimensions of the input signal to the block.

Settings

Default: `-1`

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

Dependency

Clearing **Specify properties via bus object** enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Variable-size signal

Specify the type of signals allowed into this port.

Settings

Default: `Inherit`

`Inherit`
: Allow variable-size and fixed-size signals.

`No`
: Do not allow variable-size signals.

`Yes`
: Allow only variable-size signals.

Dependencies

When the signal at this port is a variable-size signal, the **Port dimensions** parameter specifies the maximum dimensions of the signal.

Command-Line Information

Parameter: `VarSizeSig`
Type: string
Value: `'Inherit'| 'No' | 'Yes'`
Default: `'Inherit'`

Sample time (-1 for inherited)

Specify the time interval between samples.

Settings

Default: `-1`

To inherit the sample time, set this parameter to `-1`.

See "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Signal type

Specify the numeric type of the external input.

Settings

Default: `auto`

`auto`
: Accept either `real` or `complex` as the numeric type.

`real`
: Specify the numeric type as a real number.

`complex`
: Specify the numeric type as a complex number.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sampling mode

Specify whether the output signal is `Sample based` or `Frame based`.

Settings

Default: `auto`

`auto`
: Accept any sampling mode.

`Sample based`
: The output signal is sample-based.

`Frame based`
: The output signal is frame-based.

Tips

To generate frame-based signals, you must have the Signal Processing Blockset product installed.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Data type

Specify the output data type of the external input.

Settings

Default: `Inherit: auto`

`Inherit: auto`
A rule that inherits a data type

`double`
Data type is `double`.

`single`
Data type is `single`.

`int8`
Data type is `int8`.

`uint8`
Data type is `uint8`.

`int16`
Data type is `int16`.

`uint16`
Data type is `uint16`.

`int32`
Data type is `int32`.

`uint32`
Data type is `uint32`.

`boolean`
Data type is `boolean`.

`fixdt(1,16,0)`
Data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
Data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
Data type is enumerated.

`<data type expression>`
The name of a data type object, for example `Simulink.NumericType`

Tips

This parameter can also be an expression that evaluates to a data type, for example, `float('single')`

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<class name>`

`Expression`
: Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

See “Specifying a Fixed-Point Data Type” in the *Simulink User’s Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	Yes
Multidimensionalized	Yes
Virtual	Yes, if the block does *not* reside in a conditionally-executed or atomic subsystem and does *not* connect directly to an Outport block For more information, see "Virtual Blocks" in the Simulink documentation.

See Also

Outport

Integer Delay

Purpose

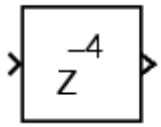
Delay signal N sample periods

Library

Discrete

Description

The Integer Delay block delays its input by N sample periods.



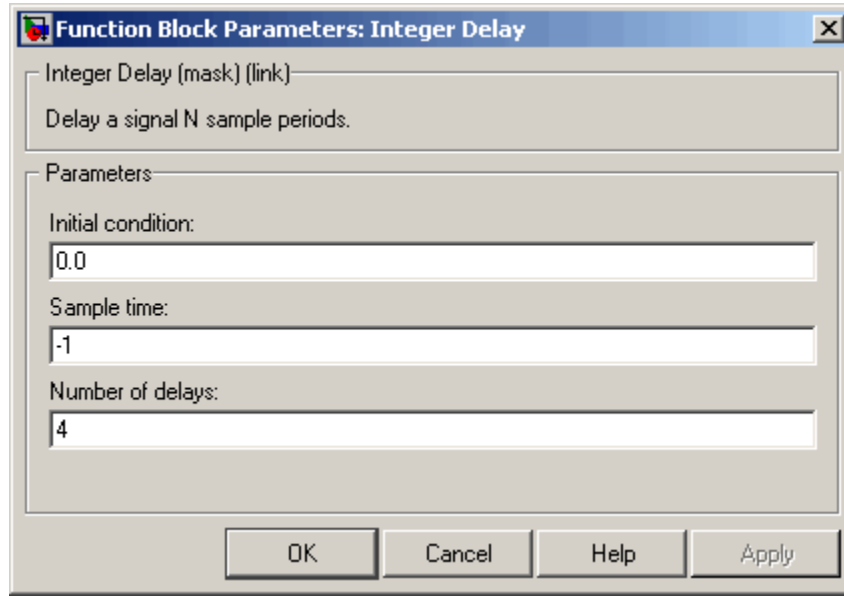
The block accepts one input and generates one output, both of which can be scalar or vector. If the input is a vector, all elements of the vector are delayed by the same sample period.

Data Type Support

The Integer Delay block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

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 "How to Specify the Sample Time" in the online documentation for more information.

Number of delays

The number of periods to delay the input signal.

Note This number cannot exceed the maximum value of a 32-bit integer divided by the input data type size (in bytes).

Characteristics

Direct Feedthrough	No
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of input or initial conditions

Purpose

Integrate 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 software can use a number of different numerical integration methods to compute the Integrator block's output, each with advantages in particular applications. Use the **Solver** pane of the Configuration Parameters dialog box (see "Solver Pane") to select the technique best suited to your application.

Simulink software 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 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.

Use the parameter dialog box to:

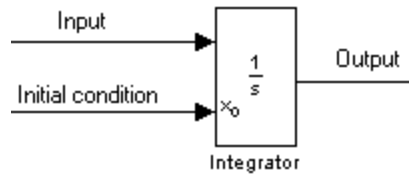
- Define upper and lower limits on the integral
- Create an input that resets the block's output (state) to its initial value, depending on how the input changes
- Create an optional state output so that the value of the block's output can trigger a block reset

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** 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.



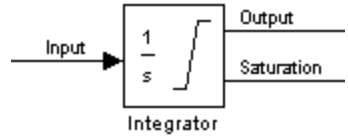
Note If the integrator limits its output (see "Limiting the Integral" on page 2-595), the initial condition must fall inside the integrator's saturation limits. If the initial condition is outside the block saturation limits, the block displays an error message.

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. This action 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 block determines output as follows:

- When the integral is less than or equal to the **Lower saturation limit**, 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**, 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.



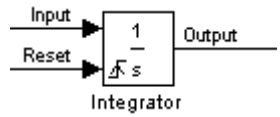
The signal has one of three values:

- 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 check box, 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.



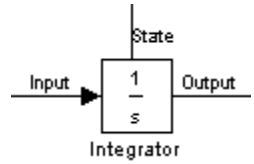
- Select `rising` to reset the state when the reset signal rises from a zero to a positive value or from a negative to a positive value.
- Select `falling` to reset the state when the reset signal falls from a positive value to zero or from a positive to a negative value.
- Select `either` to reset the state when the reset signal changes from a zero to a nonzero value or changes sign.
- Select `level` to reset the state when the reset signal is nonzero at the current time step or changes from nonzero at the previous time step to zero at the current time step.
- Select `level hold` to reset the state when the reset signal is nonzero at the current time step.

The reset port has direct feedthrough. If the block output feeds back into this port, either directly or through a series of blocks with direct feedthrough, an algebraic loop results (see "Algebraic Loops"). Use the Integrator block's state port to feed back the block's output without creating an algebraic loop.

Note To be compliant with the Motor Industry Software Reliability Association (MISRA®) software standard, your model must use Boolean signals to drive the external reset ports of Integrator blocks.

About the State Port

Selecting the **Show state port** check box on the Integrator block's parameter dialog box causes an additional output port, the state port, to appear at the top of the Integrator block.



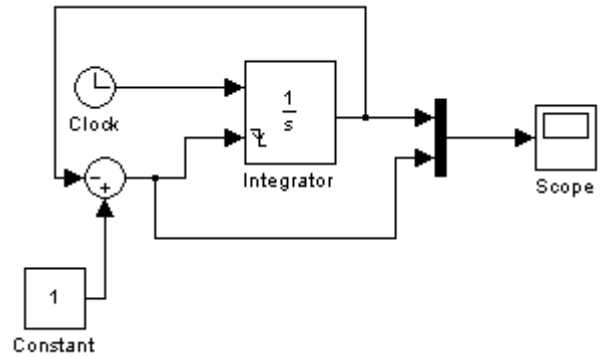
The output of the state port is the same as the output of the block's standard output port except for the following case. If the block is reset in the current time step, the output of the state port is the value that would have appeared at the block's standard output if the block had not been reset. The state port's output appears earlier in the time step than the output of the Integrator block's output port. Use the state port to avoid creating algebraic loops in these modeling scenarios:

- Self-resetting integrators (see "Creating Self-Resetting Integrators" on page 2-598)
- Handing off a state from one enabled subsystem to another (see "Handing Off States Between Enabled Subsystems" on page 2-599)

Note When updating a model, Simulink software checks that the state port applies to one of these two scenarios. If not, an error message appears. Also, you cannot log the output of this port in a referenced model that executes in Accelerator mode. If logging is enabled for the port, Simulink software generates a "signal not found" warning during execution of the referenced model.

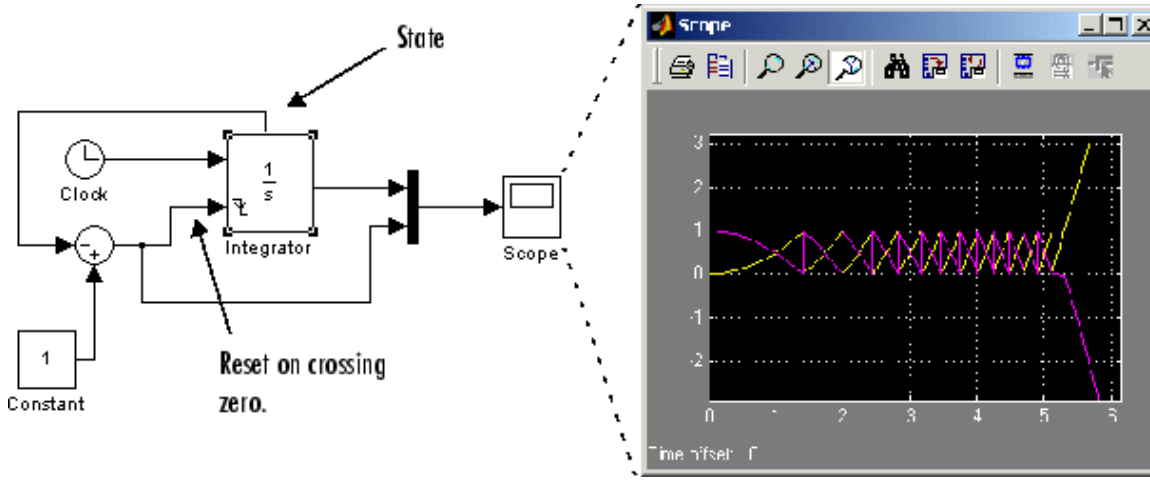
Creating Self-Resetting Integrators

The Integrator block's state port helps you avoid an algebraic loop 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. However, the model creates an algebraic loop. To compute the integrator block's output, Simulink software needs to know the value of the block's reset signal, and vice versa. Because the two values are mutually dependent, Simulink software cannot determine either. Therefore, an error message appears if you try to simulate or update this model.

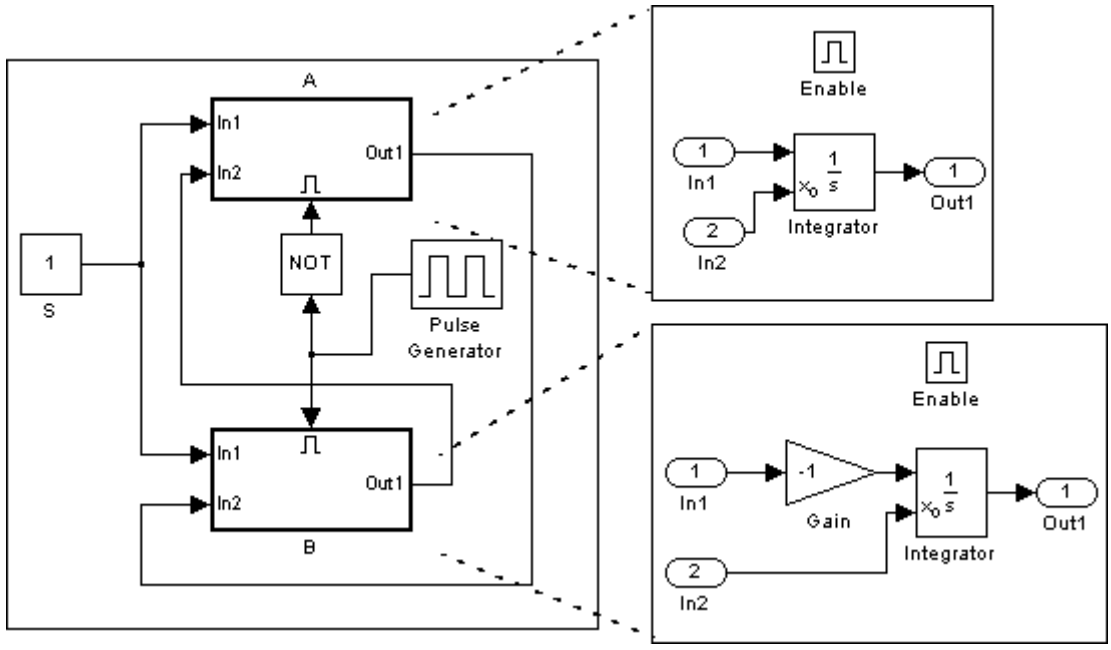
The following model uses the integrator's state port to avoid the algebraic loop.



In this version, the value of the reset signal depends on the value of the state port. The value of the state port is available earlier in the current time step than the value of the integrator block's output port. Thus, Simulink software can determine whether the block needs to be reset before computing the block's output, thereby avoiding the algebraic loop.

Handing Off States Between Enabled Subsystems

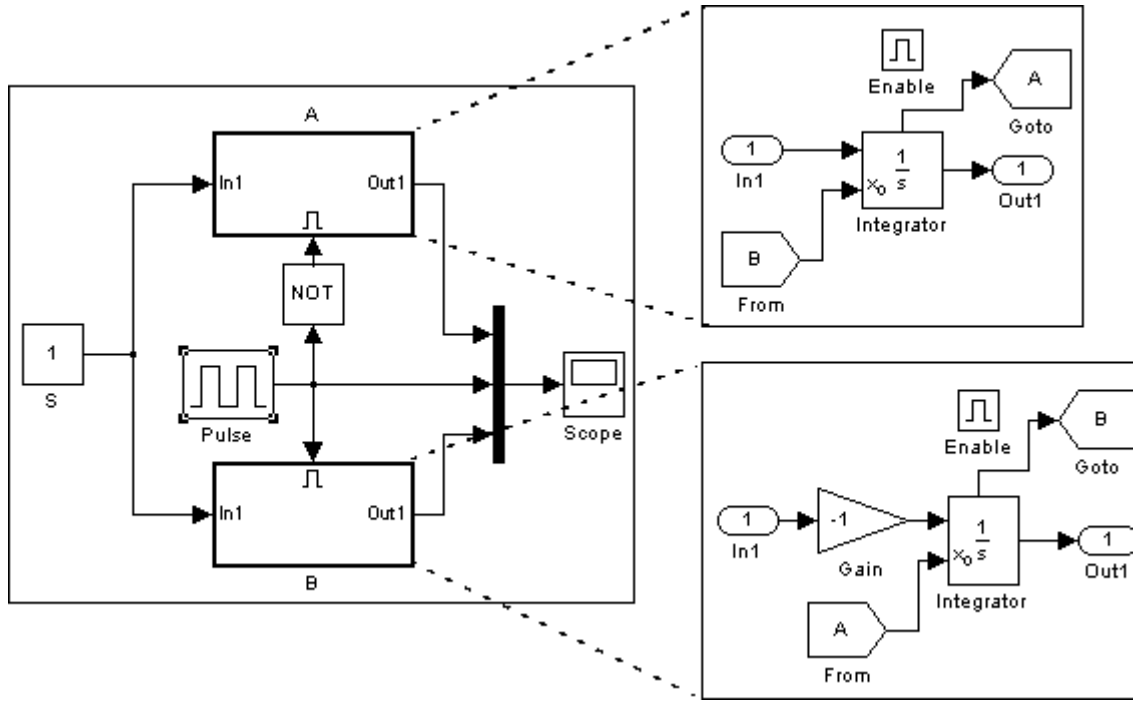
The state port helps you avoid an algebraic loop when passing a state between two enabled subsystems. Consider, for example, the following model.



In this model, a constant input signal drives two enabled subsystems that integrate the signal. A pulse generator generates an enabling signal that causes execution to alternate between the two subsystems. The enable port of each subsystem is set to reset, which causes the subsystem to reset its integrator when it becomes active. Resetting the integrator causes the integrator to read the value of its initial condition port. The initial condition port of the integrator in each subsystem is connected to the output port of the integrator in the other subsystem.

This connection is intended to enable continuous integration of the input signal as execution alternates between two subsystems. However, the connection creates an algebraic loop. To compute the output of A, Simulink software needs to know the output of B, and vice versa. Because the outputs are mutually dependent, Simulink software cannot compute them. Therefore, an error message appears if you try to simulate or update this model.

The following version of the same model uses the integrator state port to avoid creating an algebraic loop when handing off the state.



In this model, the initial condition of the integrator in A depends on the value of the state port of the integrator in B, and vice versa. The values of the state ports are updated earlier in the simulation time step than the values of the integrator output ports. Thus, Simulink software can compute the initial condition of either integrator without knowing the final output value of the other integrator. For another example of using the state port to hand off states between conditionally executed subsystems, see the `sldemo_clutch` model.

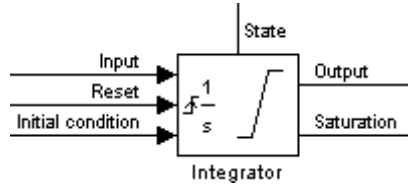
Note Simulink software does not permit three or more enabled subsystems to hand off a model state. If Simulink software detects that a model is handing off a state among more than two enabled subsystems, it generates an error.

Specifying the Absolute Tolerance for the Block's Outputs

By default Simulink software uses the absolute tolerance value specified in the Configuration Parameters dialog box (see "Specifying Variable-Step Solver Error Tolerances") to compute the output of the Integrator block. If this value does not provide sufficient error control, specify a more appropriate value in the **Absolute tolerance** field of the Integrator block's dialog box. The value that you specify is used to compute all of the block's outputs.

Selecting All Options

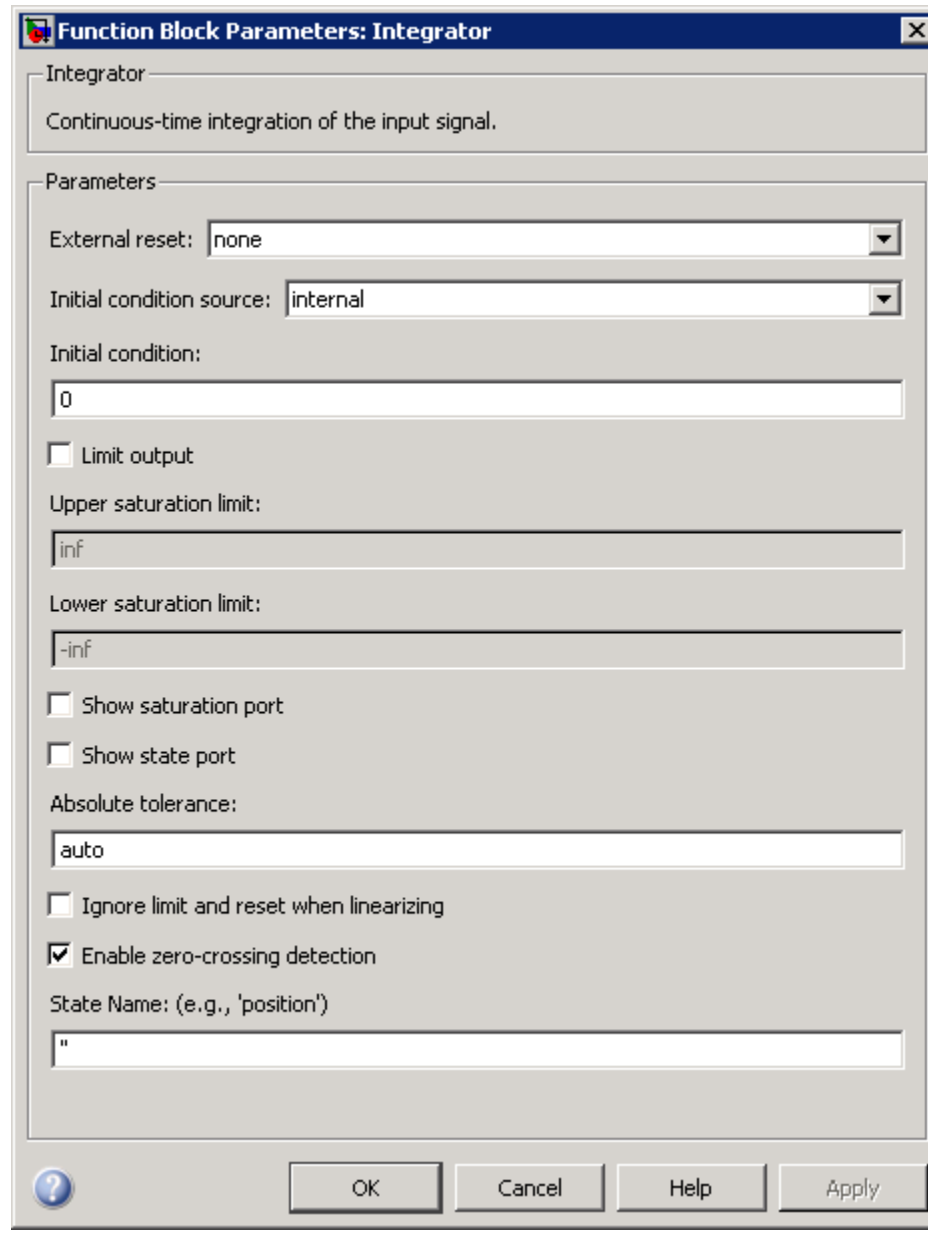
When you select all options, the block icon looks like this.



Data Type Support

The Integrator block accepts and outputs signals of type `double` on its data ports. The external reset port accepts signals of type `double` or `Boolean`.

Parameters and Dialog Box



External reset

Reset the states to their initial conditions when a trigger event occurs in the reset signal.

Settings

Default: `none`

`none`
: Do not reset the state to initial conditions.

`rising`
: Reset the state when the reset signal rises from a zero to a positive value or from a negative to a positive value.

`falling`
: Reset the state when the reset signal falls from a positive value to zero or from a positive to a negative value.

`either`
: Reset the state when the reset signal changes from a zero to a nonzero value or changes sign.

`level`
: Reset the state when the reset signal is nonzero at the current time step or changes from nonzero at the previous time step to zero at the current time step.

`level hold`
: Reset the state when the reset signal is nonzero at the current time step.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Initial condition source

Get the initial conditions of the states.

Settings

Default: `internal`

`internal`
: Get the initial conditions of the states from the **Initial condition** parameter.

`external`
: Get the initial conditions of the states from an external block.

Tips

Simulink software does not allow the initial condition of this block to be `inf` or `NaN`.

Dependencies

Selecting `internal` enables the **Initial condition** parameter.

Selecting `external` disables the **Initial condition** parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Initial condition

Specify the states' initial conditions.

Settings

Default: `0`

Tips

Simulink software does not allow the initial condition of this block to be `inf` or `NaN`.

Dependencies

Setting **Initial condition source** to `internal` enables this parameter.

Setting **Initial condition source** to `external` disables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Limit output

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

Settings

Default: Off

 On

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

 Off

Do not limit the block's output to a value between the **Lower saturation limit** and **Upper saturation limit** parameters.

Dependencies

This parameter enables **Upper saturation limit**.

This parameter enables **Lower saturation limit**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Upper saturation limit

Specify the upper limit for the integral.

Settings

Default: `inf`

Minimum: value of **Output minimum** parameter

Maximum: value of **Output maximum** parameter

Dependencies

Limit output enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Lower saturation limit

Specify the lower limit for the integral.

Settings

Default: `-inf`

Minimum: value of **Output minimum** parameter

Maximum: value of **Output maximum** parameter

Dependencies

Limit output enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Show saturation port

Add a saturation output port to the block.

Settings

Default: Off

 On

Add a saturation output port to the block.

 Off

Do not add a saturation output port to the block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Show state port

Add an output port to the block for the block's state.

Settings

Default: Off

 On

Add an output port to the block for the block's state.

 Off

Do not add an output port to the block for the block's state.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Absolute tolerance

Specify the absolute tolerance for computing the block output.

Settings

Default: `auto`

- You can enter `auto` or a numeric value.
- If you enter `auto`, Simulink uses the absolute tolerance value in the Configuration Parameters dialog box (see "Solver Pane") to compute the block output.
- If you enter a numeric value, that value overrides the absolute tolerance in the Configuration Parameters dialog box.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Ignore limit and reset when linearizing

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.

Settings

Default: Off

 On

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.

 Off

Do not 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.

Tip

Use this check box to linearize a model around an operating point that causes the integrator to reset or saturate.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Enable zero-crossing detection

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

Settings

Default: On

 On

Use 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.

 Off

Do not use 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.

If you select this check box, **Limit output**, and zero-crossing detection for the model as a whole, the Integrator block uses zero crossings as described.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

State Name (e.g., 'position')

Assign a unique name to each state.

Settings

Default: `' '`

If this field is blank, no name assignment occurs.

Tips

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'`.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, `{'a', 'b', 'c'}`. Each name must be unique.
- The state names apply only to the selected block.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

  For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.
- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell array, or structure.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

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 Detection	Yes, if enabled and you select the **Limit output** option, one for detecting reset, one each to detect upper and lower saturation limits, and one when leaving saturation

See Also

Discrete-Time Integrator

Interpolation Using Prelookup

Purpose

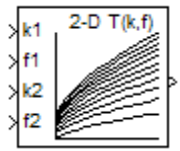
Use precalculated index and fraction values to accelerate approximation of N-dimensional function

Library

Lookup Tables

Description

Comparison with the Lookup Table (n-D) Block



The Interpolation Using Prelookup block works best with the Prelookup block. The Prelookup block calculates the index and interval fraction that specifies how its input value relates to the breakpoint data set. You feed the resulting index and fraction values into an Interpolation Using Prelookup block to interpolate an *n*-dimensional table. This combination of blocks performs the same operation that a single instance of the Lookup Table (n-D) block performs. However, the Prelookup and Interpolation Using Prelookup blocks offer greater flexibility that can provide more efficient simulation and code generation.

How The Block Works with a Prelookup Block

To use this block, you must define a set of output values as the **Table data** parameter. Typically, these table values correspond to the breakpoint data sets specified in Prelookup blocks. The Interpolation Using Prelookup block generates output by looking up or estimating table values based on index and interval fraction values fed from Prelookup blocks. The index and interval fraction appear on the Interpolation Using Prelookup block as `k` and `f`, respectively.

If the inputs...	The Interpolation Using Prelookup block...
Match the values of indices in breakpoint data sets	Outputs the table value at the intersection of the row, column, and higher dimension breakpoints

If the inputs...	The Interpolation Using Prelookup block...
Do not match the values of indices in breakpoint data sets, but are within range	Interpolates appropriate table values, using the **Interpolation method** you select
Do not match the values of indices in breakpoint data sets, and are out of range	Extrapolates the output value, using the **Extrapolation method** you select

How The Block Interpolates a Subset of Table Data

You can use the **Number of sub-table selection dimensions** parameter to specify that interpolation occur only on a subset of the table data. To activate this interpolation mode, set this parameter to a positive integer. This value defines the number of dimensions to select, starting from the highest dimension of table data. Therefore, the value must be less than or equal to the **Number of table dimensions**.

Suppose that you have 3-D table data in your Interpolation Using Prelookup block. The following behavior applies.

Number of Selection Dimensions	Action by the Block	Block Appearance
0	Interpolates the entire table and does not activate subtable selection	Does not change

Number of Selection Dimensions	Action by the Block	Block Appearance
1	Interpolates the first two dimensions and selects the third dimension	Displays an input port with the label `sel1` that you use to select and interpolate 2-D tables
2	Interpolates the first dimension and selects the second and third dimensions	Displays two input ports with the labels `sel1` and `sel2` that you use to select and interpolate 1-D tables

For an example of interpolating a subset of table data, type `sldemo_bpcheck` at the MATLAB command prompt.

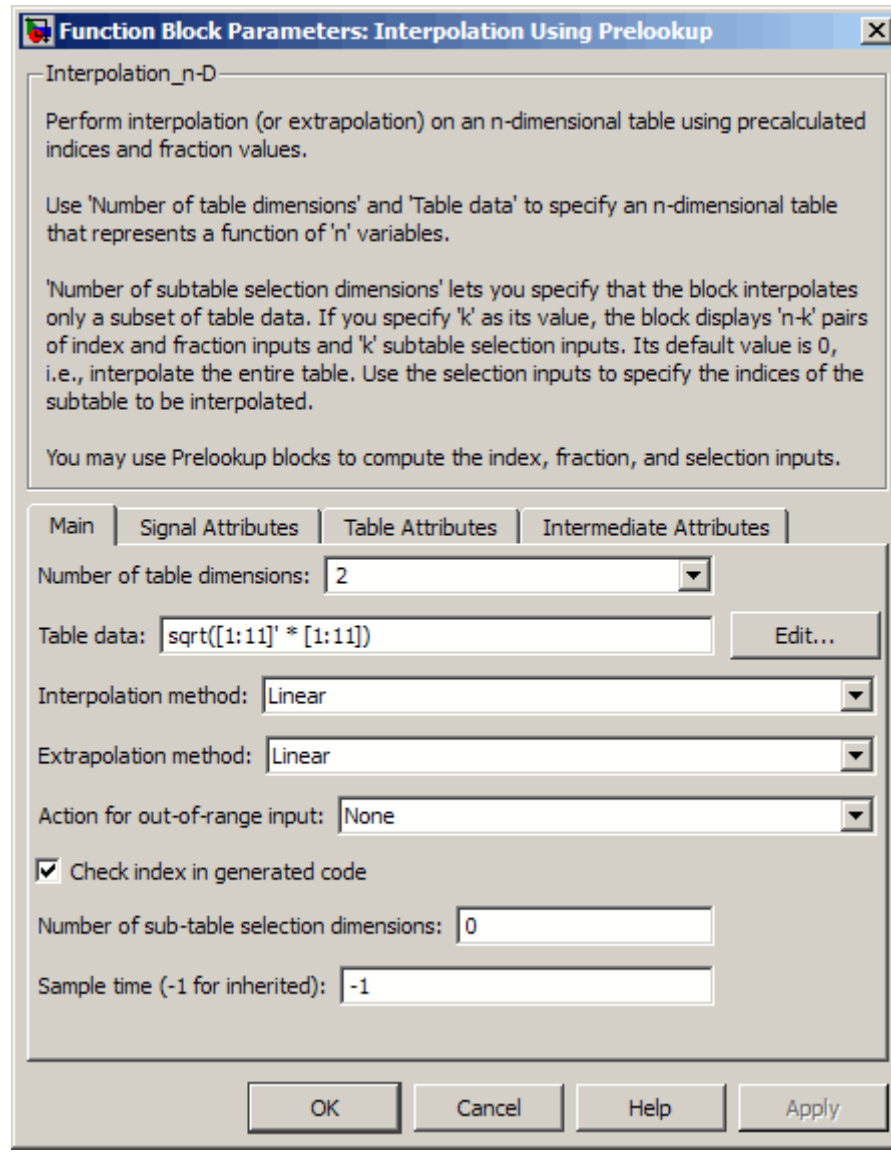
Data Type Support

The Interpolation Using Prelookup block accepts real signals of any numeric data type supported by Simulink software, except `Boolean`. The Interpolation Using Prelookup block supports fixed-point data types for signals, table data, and intermediate results.

For more information, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Interpolation Using Prelookup block dialog box appears as follows:



Number of table dimensions

The number of dimensions that the **Table data** parameter must have. This value defines the number of independent variables for the table and the number of inputs to the block. Enter an integer between `1` and `30` into this field.

Table data

The table of output values. During simulation, the matrix size must match the dimensions defined by the **Number of table dimensions** parameter. Note that during block diagram editing, you can enter an empty matrix (specified as `[]`) or an undefined workspace variable as the **Table data** parameter. Use this behavior to postpone specifying a correctly-dimensioned matrix for the **Table data** parameter and continue editing the block diagram. For information about how to construct multidimensional arrays in MATLAB software, see "Multidimensional Arrays" in the MATLAB online documentation.

Click the **Edit** button to open the Lookup Table Editor (see "Lookup Table Editor" in the Simulink documentation).

Interpolation method

`None - Flat` or `Linear`. See "Interpolation Methods" in the Simulink documentation for more information.

Extrapolation method

`None - Clip` or `Linear`. See "Extrapolation Methods" in the Simulink documentation for more information. The **Extrapolation method** parameter is visible only if you select `Linear` as the **Interpolation method** parameter.

Note The Interpolation Using Prelookup block does not support `Linear` extrapolation if its input or output signals specify integer or fixed-point data types.

Action for out-of-range input

Specifies whether to produce a warning or error message if the input is out of range. Options include:

- `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

Check index in generated code (Real-Time Workshop license required)

Specifies whether Real-Time Workshop generated code checks the validity of index values that feed this block.

This check box has no effect on generated code if one of the following is true:

- The Prelookup block feeds index values to the Interpolation Using Prelookup block.

  Because index values from the Prelookup block are always valid, no check code is necessary.

- The data type of the input signal restricts the data to valid index values.

  For example, unsigned integer data types guarantee nonnegative index values. Therefore, unsigned inputs do not require check code for negative values.

Valid index input may reach last index

Specifies how the block inputs for index (`k`) and interval fraction (`f`) access the last elements of the *n*-dimensional table in **Table data**. Index values are zero-based.

If this check box is...	The block returns the value of the last element in a dimension of its table when...
Selected	• `k` indexes the last table element in the corresponding dimension • `f` is `0`
Not selected	• `k` indexes the next-to-last table element in the corresponding dimension • `f` is `1`

This check box is visible only when:

- **Interpolation method** is `Linear`.
- **Extrapolation method** is `None - Clip`.

Tip If you select **Valid index input may reach last index** for an Interpolation Using Prelookup block, you must also select **Use last breakpoint for input at or above upper limit** for all Prelookup blocks that feed it. This action allows the blocks to use the same indexing convention when accessing the last elements of their **Breakpoint data** and **Table data** parameters.

Number of sub-table selection dimensions

Specifies the number of dimensions of the subtable that the block uses to compute the output. Follow these rules:

- To enable subtable selection, enter a positive integer.

  This integer must be less than or equal to the **Number of table dimensions**.

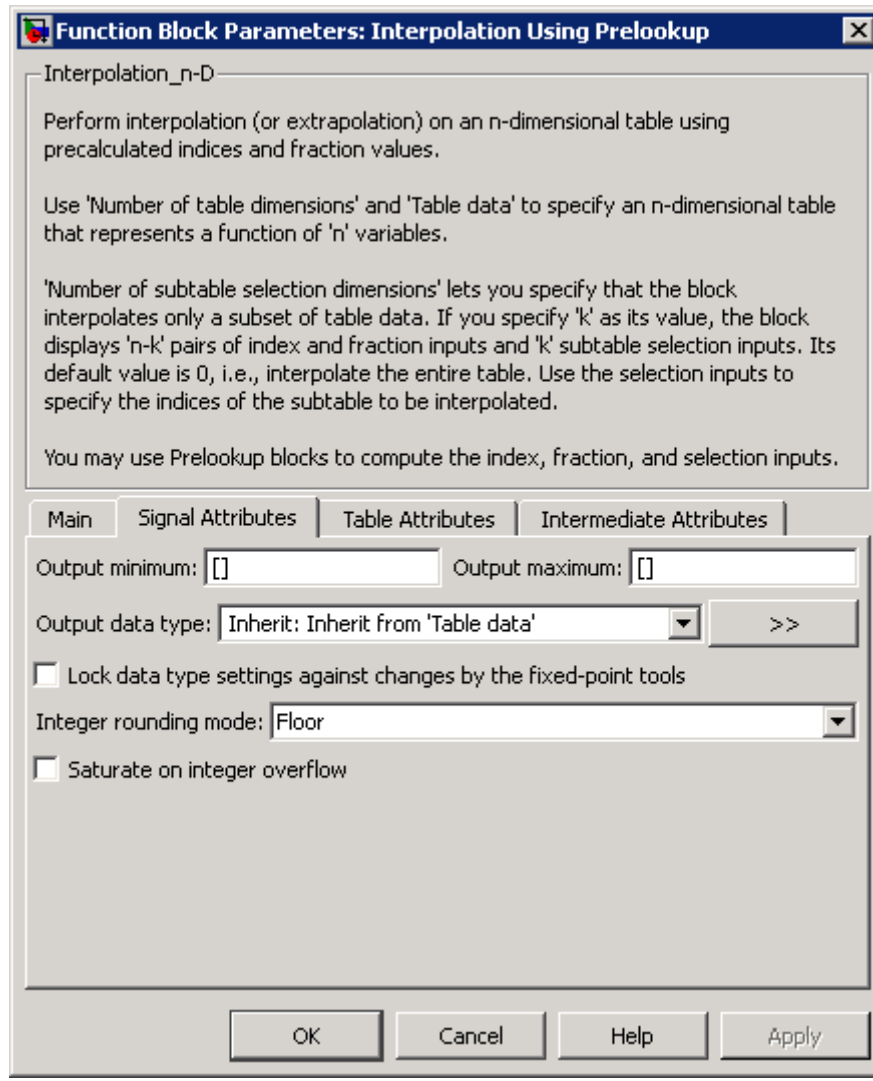
- To disable subtable selection, enter `0` (the default) to interpolate the entire table.

For more information, see “How The Block Interpolates a Subset of Table Data” on page 2-619.

Sample time

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

The **Signal Attributes** pane of the Interpolation Using Prelookup block dialog box appears as follows:



Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum

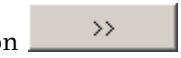
Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Lock data type settings against changes by the fixed-point tools
Select to lock all data type settings of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Block parameters such as **Table data** are always rounded to the nearest representable value. To control the rounding of a block parameter, enter an expression using a MATLAB rounding function into the mask field.

Saturate on integer overflow
Select to have overflows saturate. Otherwise, overflows wrap.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

The **Table Attributes** pane of the Interpolation Using Prelookup block dialog box appears as follows:

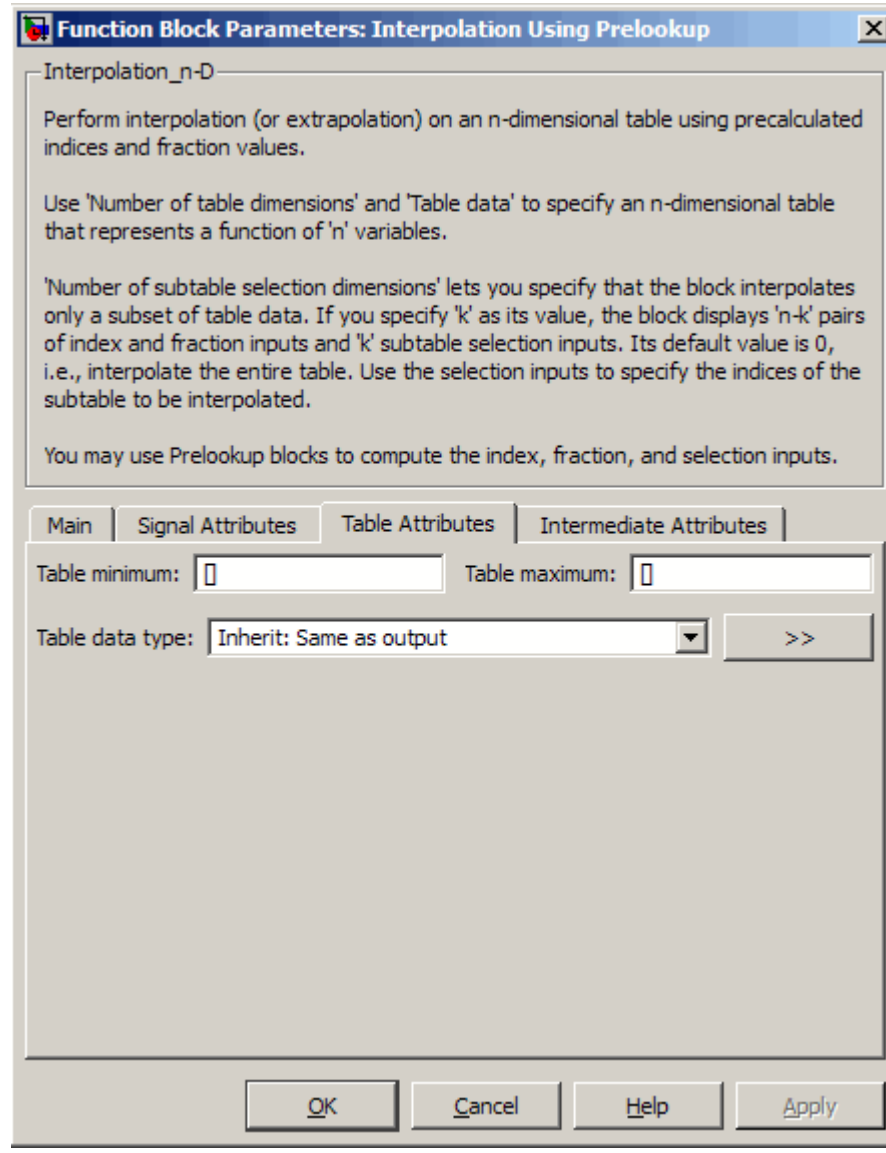


Table minimum

Specify the minimum value for table data. The default value, `[]`, is equivalent to `-Inf`.

Table maximum

Specify the maximum value for table data. The default value, `[]`, is equivalent to `Inf`.

Table data type

Specify the table data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as output`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

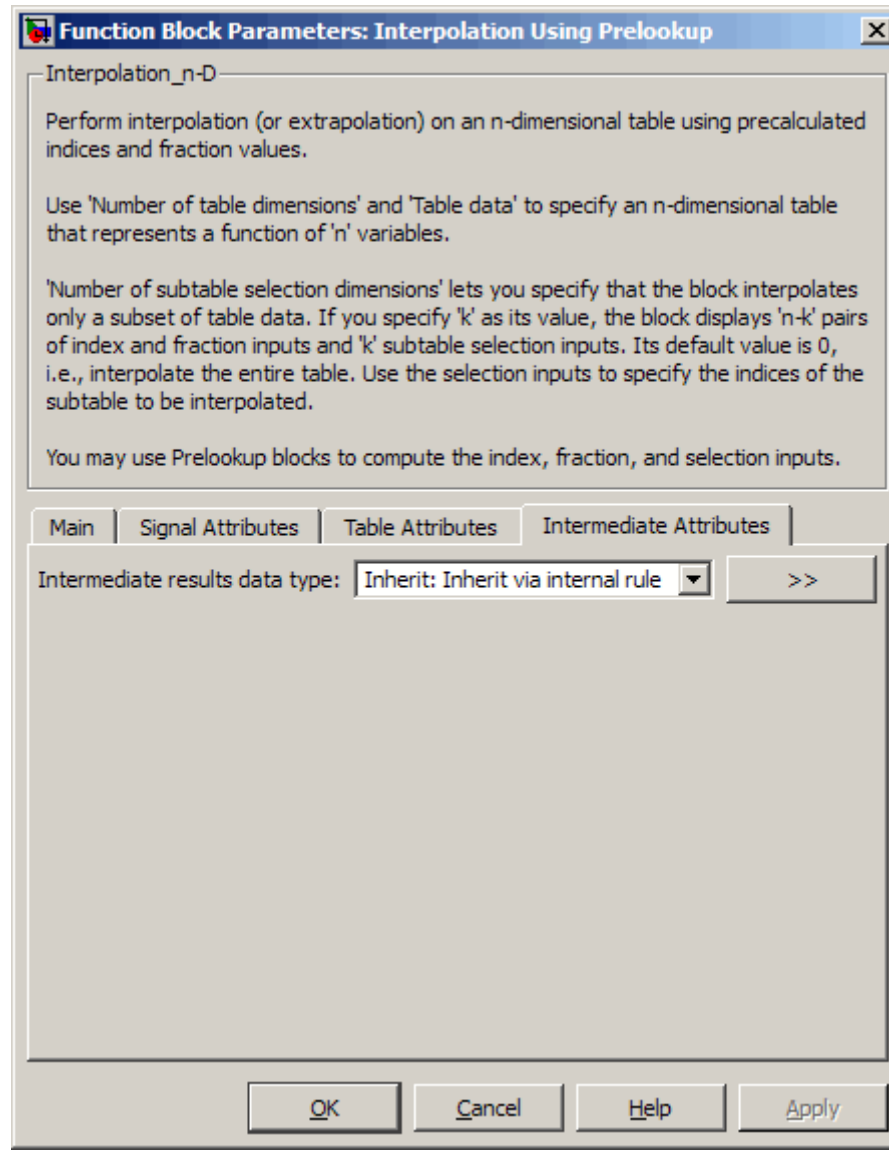
to display the **Data Type Assistant**, which helps you set the **Table data type** parameter.

Tip Specify a table data type different from the output data type for these cases:

- Lower memory requirement for storing table data that uses a smaller type than the output signal
- Sharing of prescaled table data between two Interpolation Using Prelookup blocks with different output data types
- Sharing of custom storage table data in Real-Time Workshop generated code for blocks with different output data types

---

The **Intermediate Attributes** pane of the Interpolation Using Prelookup block dialog box appears as follows:



Intermediate results data type

Specify the intermediate results data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as output`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Intermediate results data type** parameter.

Tip Use this parameter to specify higher precision for internal computations than for table data or output data.

Examples

See "Examples for Prelookup and Interpolation Blocks" in the Simulink documentation.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes
Dimensionalized	Yes
Zero-Crossing Detection	No

See Also

Prelookup

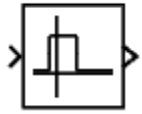
Purpose

Determine if signal is in specified interval

Library

Logic and Bit Operations

Description



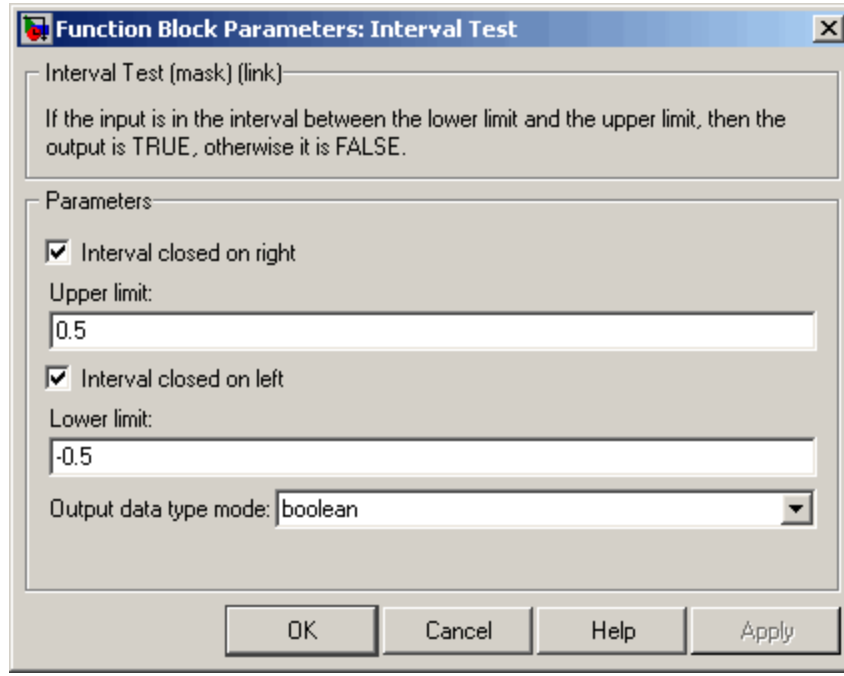
The Interval Test block outputs TRUE if the input is between the values specified by the **Lower limit** and **Upper limit** parameters. The block outputs FALSE if the input is outside those values. The output of the block when the input is equal to the **Lower limit** or the **Upper limit** is determined by whether the boxes next to **Interval closed on left** and **Interval closed on right** are selected in the dialog box.

Data Type Support

The Interval Test block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

Parameters and Dialog Box



Interval closed on right
When you select this check box, the **Upper limit** is included in the interval for which the block outputs TRUE.

Upper limit
The upper limit of the interval for which the block outputs TRUE.

Interval closed on left
When you select this check box, the **Lower limit** is included in the interval for which the block outputs TRUE.

Lower limit
The lower limit of the interval for which the block outputs TRUE.

Output data type mode
Select the output data type: `boolean` or `uint8`.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

Interval Test Dynamic

Interval Test Dynamic

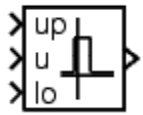
Purpose

Determine if signal is in specified interval

Library

Logic and Bit Operations

Description



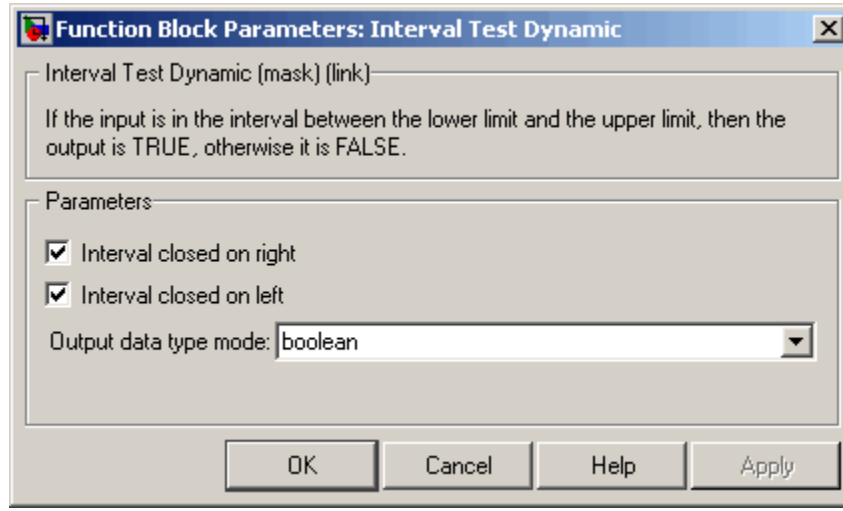
The Interval Test Dynamic block outputs TRUE if the input is between the values of the external signals up and lo. The block outputs FALSE if the input is outside those values. The output of the block when the input is equal to the signal up or the signal lo is determined by whether the boxes next to **Interval closed on left** and **Interval closed on right** are selected in the dialog box.

Data Type Support

The Interval Test Dynamic block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

Parameters and Dialog Box



Interval closed on right
When you select this check box, the value of the signal connected to the block's "up" input port is included in the interval for which the block outputs TRUE.

Interval closed on left
When you select this check box, the value of the signal connected to the block's "lo" input port is included in the interval for which the block outputs TRUE.

Output data type mode
Select the output data type: `boolean` or `uint8`.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

Interval Test

Level-2 M-File S-Function

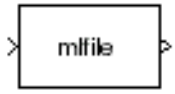
Purpose

Use Level-2 M-file S-function in model

Library

User-Defined Functions

Description



This block allows you to use a Level-2 M-file S-function (see "Writing Level-2 M-File S-Functions") in a model. To do this, create an instance of this block in the model. Then enter the name of the Level-2 M-File S-function in the **M-file name** field of the block's parameter dialog box.

Note Use the S-Function block to include a Level-1 M-file S-function in a block.

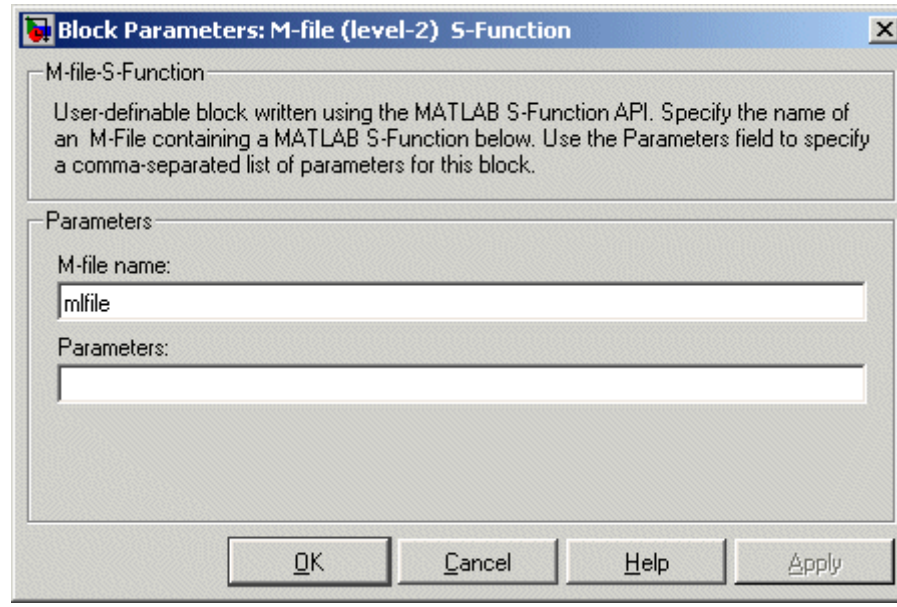
If the Level-2 M-file S-function defines any additional parameters, you can enter them in the **Parameters** field of the block's parameter dialog box. Enter them as MATLAB expressions that evaluate to their values in the order defined by the M-file S-function. Use commas to separate each expression.

If a model includes a Level-2 M-File S-Function block, and an error occurs in the S-function, the Level-2 M-File S-Function block displays M-file stack trace information for the error in a dialog box. Click **OK** to remove the dialog box.

Data Type Support

Depends on the M-file that defines the behavior of a particular instance of this block.

Parameters and Dialog Box



M-file name

Name of an M-file that defines the behavior of this block. The M-file must follow the Level-2 standard for writing M-file S-functions (see "Writing Level-2 M-File S-Functions").

Parameters

Values of the parameters of this block.

Characteristics

Direct Feedthrough	Depends on the M-file S-function
Sample Time	Depends on the M-file S-function
Scalar Expansion	Depends on contents M-file S-function
Dimensionalized	Depends on the M-file S-function

Multidimensionalized	Yes
Zero Crossing	No

Purpose

Perform specified logical operation on input

Library

Logic and Bit Operations

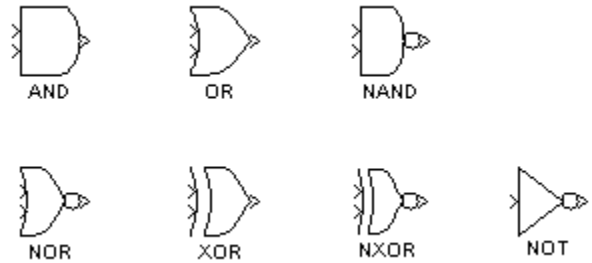
Description

The Logical Operator block performs the specified logical operation on its inputs. An input value is TRUE (1) if it is nonzero and FALSE (0) if it is zero.

You select the Boolean operation connecting the inputs with the **Operator** parameter list. If you select `rectangular` as the **Icon shape** property, the block updates to display the name of the selected operator. The supported operations are given below.

Operation	Description
AND	TRUE if all inputs are TRUE
OR	TRUE if at least one input is TRUE
NAND	TRUE if at least one input is FALSE
NOR	TRUE when no inputs are TRUE
XOR	TRUE if an odd number of inputs are TRUE
NXOR	TRUE if an even number of inputs are TRUE
NOT	TRUE if the input is FALSE

If you select `distinctive` as the **Icon shape**, the block's appearance indicates its function. Simulink software displays a distinctive shape for the selected operator, conforming to the IEEE Standard Graphic Symbols for Logic Functions:



The number of input ports is specified with the **Number of input ports** parameter. The output type is specified with the **Output data type** parameter. An output value is 1 if TRUE and 0 if FALSE.

Note The output data type should represent zero exactly. Data types that satisfy this condition include signed and unsigned integers, and any floating-point data type.

The size of the output depends on input vector size and the selected operator:

- If the block has more than one input, any nonscalar inputs must have the same dimensions. For example, if any input is a 2-by-2 array, all other nonscalar inputs must also be 2-by-2 arrays.

  Scalar inputs are expanded to have the same dimensions as the nonscalar inputs.

  If the block has more than one input, the output has the same dimensions as the inputs (after scalar expansion) and each output element is the result of applying the specified logical operation to the corresponding input elements. For example, if the specified operation is AND and the inputs are 2-by-2 arrays, the output is a 2-by-2 array whose top left element is the result of applying AND to the top left elements of the inputs, etc.

- For a single vector input, the block applies the operation (except the NOT operator) to all elements of the vector. The output is always a scalar.
- 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 logical complements of the input vector elements.

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.

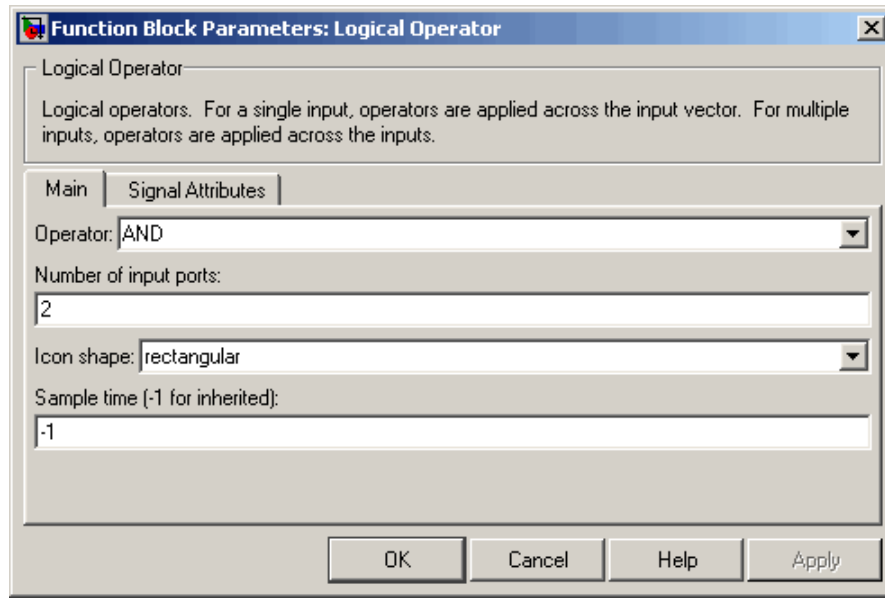
Data Type Support

The Logical Operator block accepts real signals of any numeric data type supported by Simulink software, including fixed-point data types.

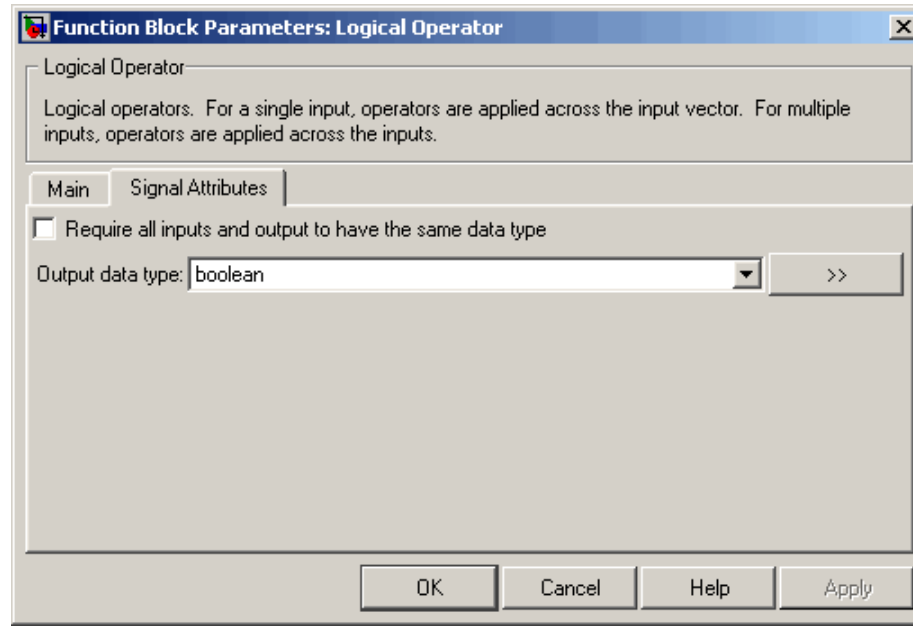
For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Logical Operator block dialog box appears as follows:



The **Signal Attributes** pane of the Logical Operator block dialog box appears as follows:



Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User’s Guide* for more information.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Operator

Select logical operator to apply to block inputs.

Settings

Default: AND

AND
: TRUE if all inputs are TRUE

OR
: TRUE if at least one input is TRUE

NAND
: TRUE if at least one input is FALSE

NOR
: TRUE when no inputs are TRUE

XOR
: TRUE if an odd number of inputs are TRUE

NXOR
: TRUE if an even number of inputs are TRUE

NOT
: TRUE if the input is FALSE

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Number of input ports

Specify number of block inputs.

Settings

Default: 2

- The value must be appropriate for the selected operator.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Icon shape

Specify shape of the block icon.

Settings

Default: `rectangular`

`rectangular`
: Result in a rectangular block that displays the name of the selected operator.

`distinctive`
: Use the graphic symbol for the selected operator as specified by the IEEE standard.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits or specify another appropriate sample time such as continuous or inherited.

Settings

Default: -1

By default, the block inherits its sample time based upon the context of the block within the model. To set a different sample time, enter a valid sample time based upon the table in "Types of Sample Time".

See also "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Require all inputs and output to have the same data type

Require all inputs and the output to have the same data type.

Settings

Default: Off

 On

Require all inputs and the output to have the same data type.

 Off

Do not require all inputs and the output to have the same data type.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `boolean`

`Inherit: Logical (see Configuration Parameters: Optimization)`

Use the **Implement logic signals as Boolean data** configuration parameter (see "Implement logic signals as Boolean data (vs. double)") to specify the output data type.

> **Note** This option supports models created before the `boolean` option was available. Use one of the other options, preferably `boolean`, for new models.

`boolean`

Output data type is `boolean`.

`fixdt(1,16)`

Output data type is `fixdt(1,16)`.

`<data type expression>`

Use the name of a data type object (for example, `Simulink.NumericType`)

> **Tip** To enter a built-in data type (`double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`), enclose the expression in single quotes. For example, enter `'double'` instead of `double`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
Fixed-point data types.

`Enumerated`
Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `<class name>`

`Expression`
Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Characteristics

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

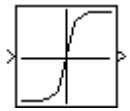
Purpose

Approximate one-dimensional function

Library

Lookup Tables

Description



The Lookup Table block computes an approximation to a 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* (i.e., the value of the next element in the vector is greater than the value of the preceding element) after conversion to the input's fixed-point data type. However, the `x` data vector may be *monotonically increasing* (i.e., the value of the next element in the vector is greater than or equal to the value of the preceding element) if all of the following apply:

- The input and output signals are both either `single` or `double`.
- The lookup method is `Interpolation-Extrapolation`.

Tip Evenly-spaced breakpoints can make Real-Time Workshop generated code division-free. For more information, see `fixpt_evenspace_cleanup` in the Simulink documentation and "Identify questionable fixed-point operations" in the Real-Time Workshop documentation.

For more information about size and monotonicity requirements, see "Characteristics of Lookup Table Data" in the *Simulink User's Guide*. To learn how to model a discontinuous function using a Lookup Table block, see "Representation of Discontinuities in Lookup Tables".

You define the table by specifying the **Vector of input values** parameter as a `1-by-n` vector and the **Table data** parameter as a

`1-by-n` vector. The block generates output based on the input values using one of these methods selected from the **Lookup method** parameter list:

- `Interpolation-Extrapolation` — This default method 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.

Note If the **Lookup method** parameter specifies `Interpolation-Extrapolation`, Real-Time Workshop code generation can occur for this block only if its input and output signals have the same floating-point data type.

- `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.

Note If the input value is exactly halfway between two points in the **Vector of input values**, the nearest input is the point with the higher numerical value.

Suppose that you define the **Vector of input values** as [0 2] and the **Table data** as [0 1]. If your input value is 1, the `Use Input Nearest` method chooses the nearest input to be 2, and the corresponding output is 1.

- `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 The `Use Input Nearest`, `Use Input Below`, and `Use Input Above` methods perform the same action when the input `x` matches a breakpoint value.

The Lookup Table icon displays a graph of the input vector versus the output vector. If you change a parameter on the block's dialog box, the graph is automatically redrawn when you click the **OK** or **Apply** button.

To avoid parameter saturation errors, the Simulink Fixed Point software's automatic scaling script employs a special rule for the Lookup Table block. `autofixexp` modifies the scaling by using the output lookup values in addition to the logged minimum and maximum simulation

values. This prevents the data from being saturated to different values. The lookup values are given by the **Table data** parameter.

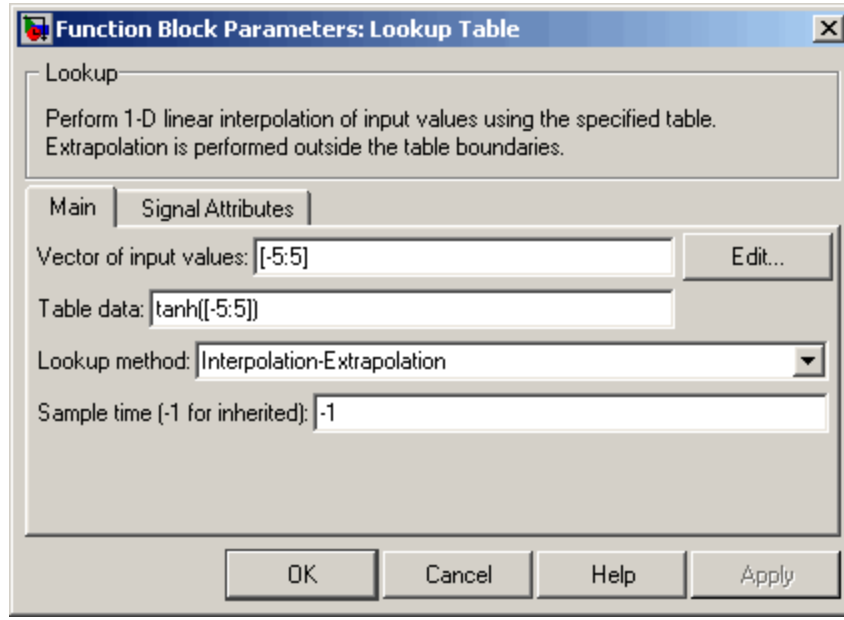
Data Type Support

The Lookup Table block supports all numeric data types supported by Simulink software, including fixed-point data types.

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

Parameters and Dialog Box

The **Main** pane of the Lookup Table block dialog box appears as follows:



Vector of input values

Specify the vector of input values. The input values vector must be the same size as the **Table data**. Also, the input values vector must be strictly monotonically increasing after conversion to the input's fixed-point data type. However, the input values vector may be monotonically increasing if the input and output signals

are both either `single` or `double`, and if the lookup method is `Interpolation-Extrapolation`. Note that due to quantization, the input values vector may be strictly monotonic in doubles format, but not so after conversion to a fixed-point data type.

The **Vector of input values** parameter is converted offline to the data type of the input signal using round-to-nearest and saturation.

Click the **Edit** button to open the Lookup Table Editor (see "Lookup Table Editor" in the online Simulink documentation).

Table data

Specify the vector of output values. The table data must be the same size as the **Vector of input values**.

The **Table data** parameter is converted offline to the **Output data type** using round-to-nearest and saturation.

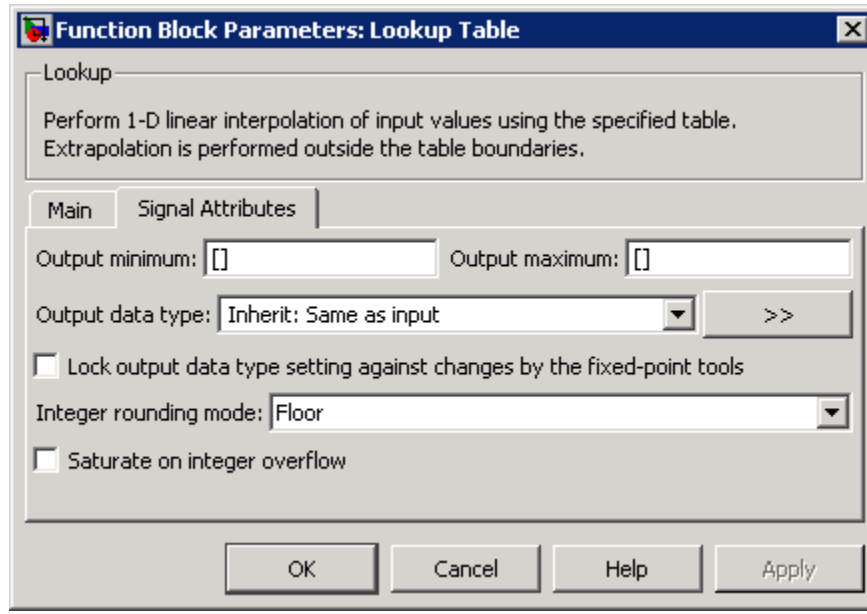
Lookup method

Specify the lookup method. See Description for a discussion of the options for this parameter. For an example that demonstrates values that the Lookup Table block returns based on different lookup methods, see "Example Output for Lookup Methods" in *Simulink User's Guide*.

Sample time (-1 for inherited)

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

The **Signal Attributes** pane of the Lookup Table block dialog box appears as follows:



Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

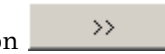
- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")

- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point lookup table calculations that occur during simulation or execution of code generated from the model. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Note that this option does not affect rounding of block parameters values, such as **Table data**. Simulink software rounds such values to the nearest representable integer value. To control the rounding of a block parameter, enter an expression using a

MATLAB rounding function into the parameter's edit field on the block dialog box.

Saturate on integer overflow
Select to have overflows saturate. Otherwise, they wrap.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	Yes
Zero-Crossing Detection	No

See Also

Lookup Table (2-D), Lookup Table (n-D)

Purpose

Approximate two-dimensional function

Library

Lookup Tables

Description

The Lookup Table (2-D) block computes an approximation to a function `z = f(x,y)` given `x`, `y`, `z` data points. The first input port corresponds to the first table dimension, `x`. (See "How to Rotate a Block" in the Simulink documentation for a description of the port order for various block orientations.)

The **Row index input values** parameter is a 1-by-m vector of `x` data points, the **Column index input values** parameter is a 1-by-n vector of `y` data points, and the **Table data** parameter is an m-by-n matrix of `z` data points. Both the row and column vectors must be *monotonically increasing* (i.e., the value of the next element in the vector is greater than or equal to the value of the preceding element). However, these vectors must be *strictly monotonically increasing* (i.e., the value of the next element in the vector is greater than the value of the preceding element) in the following cases:

- The input and output data types are both fixed-point.
- The input and output data types are different.
- The lookup method is not `Interpolation-Extrapolation`.
- The matrix of output values is complex.
- Minimum, maximum, and overflow logging is on.

Tip Evenly-spaced breakpoints can make Real-Time Workshop generated code division-free. For more information, see `fixpt_evenspace_cleanup` in the Simulink documentation and "Identify questionable fixed-point operations" in the Real-Time Workshop documentation.

The block generates output based on the input values using one of these methods selected from the **Lookup method** parameter list:

- `Interpolation-Extrapolation` — This default method performs linear interpolation and extrapolation of the inputs.
  - If the inputs match row and column parameter values, the output is the value at the intersection of the row and column.
  - If the inputs do not match row and column parameter values, then the block generates output by linearly interpolating between the appropriate row and column values. If either or both block inputs are less than the first or greater than the last row or column values, the block extrapolates using the first two or last two points.

Note If the **Lookup method** parameter specifies `Interpolation-Extrapolation`, Real-Time Workshop can generate code for this block only if its input and output signals have the same floating-point data type.

- `Interpolation-Use End Values` — This method performs linear interpolation as described above but does not extrapolate outside the end points of `x` and `y`. Instead, the end-point values are used.
- `Use Input Nearest` — This method does not interpolate or extrapolate. Instead, the elements in `x` and `y` nearest the current inputs are found. The corresponding element in `z` is then used as the output.
- `Use Input Below` — This method does not interpolate or extrapolate. Instead, the elements in `x` and `y` nearest and below the current inputs are found. The corresponding element in `z` is then used as the output. If there are no elements in `x` or `y` below the current inputs, then the nearest elements are found.
- `Use Input Above` — This method does not interpolate or extrapolate. Instead, the elements in `x` and `y` nearest and above the current

inputs are found. The corresponding element in `z` is then used as the output. If there are no elements in `x` or `y` above the current inputs, then the nearest elements are found.

Note The `Use Input Nearest`, `Use Input Below`, and `Use Input Above` methods perform the same action when the input `x` matches a breakpoint value.

For information about creating a table with step transitions, see "Representation of Discontinuities in Lookup Tables" in the *Simulink User's Guide*.

To avoid parameter saturation errors, the Simulink Fixed Point software's automatic scaling script employs a special rule for the Lookup Table (2-D) block. `autofixexp` modifies the scaling by using the output lookup values in addition to the logged minimum and maximum simulation values. The output lookup values are converted to the specified output data type. This action prevents the data from being saturated to different values.

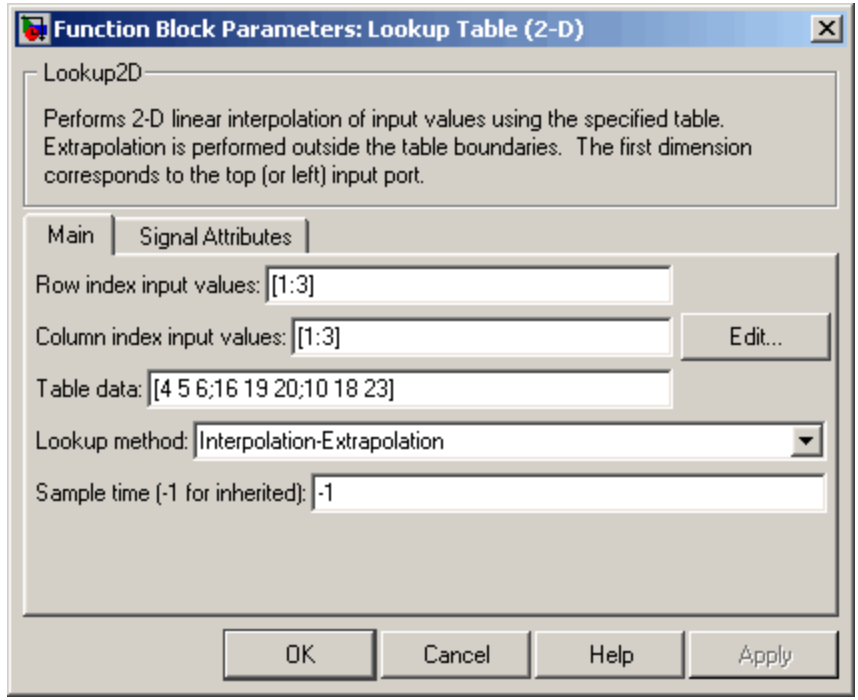
Data Type Support

The Lookup Table (2-D) block supports all numeric data types supported by Simulink software, including fixed-point data types.

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

Parameters and Dialog Box

The **Main** pane of the Lookup Table (2-D) block dialog box appears as follows:



Row index input values

The row values for the table, entered as a vector. The vector values must increase monotonically.

The **Row index input values** parameter is converted offline to the data type of the corresponding input signal using round-to-nearest and saturation.

Column index input values

The column values for the table, entered as a vector. The vector values must increase monotonically.

The **Column index input values** parameter is converted offline to the data type of the corresponding input signal using round-to-nearest and saturation.

Click the **Edit** button to open the Lookup Table Editor (see "Lookup Table Editor" in the online Simulink documentation).

Table data

The table of output values. The matrix size must match the dimensions defined by the **Row** and **Column** parameters.

The **Table data** parameter is converted offline to the **Output data type** using round-to-nearest and saturation.

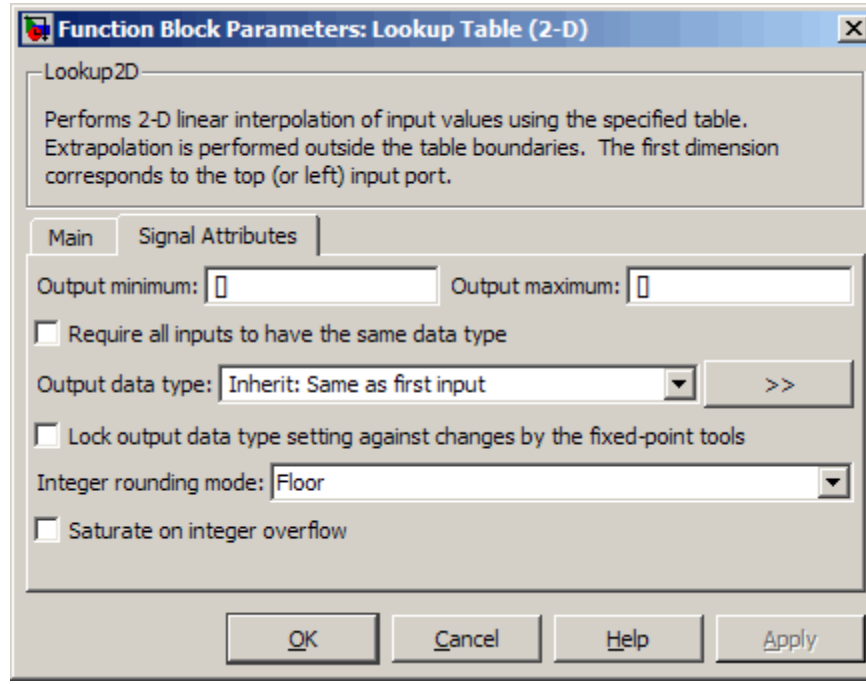
Lookup method

Specify the lookup method. See Description for a discussion of the options for this parameter.

Sample time (-1 for inherited)

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

The **Signal Attributes** pane of the Lookup Table (2-D) block dialog box appears as follows:



Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

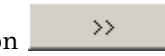
- Parameter range checking (see "Checking Parameter Values")

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Require all inputs to have the same data type
Select to require all inputs to have the same data type.

Output data type
Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools
Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Block parameters such as **Table data** always round to the nearest representable value. To control the rounding of a block parameter, enter an expression using a MATLAB rounding function into the mask field.

Saturate on integer overflow

Select to have overflows saturate. Otherwise, they wrap.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

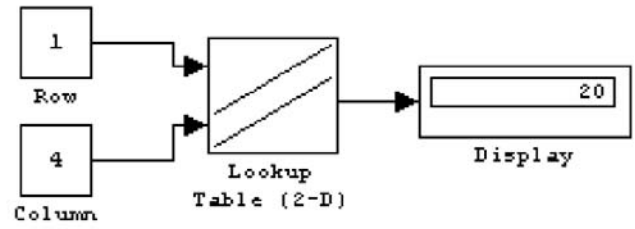
Examples

In this example, the block parameters are:

```
Row index input values:     [1 2]
Column index input values:  [3 4]
Table data:                 [10 20; 30 40]
```

The first figure shows the block outputting a value at the intersection of block inputs that match row and column values. The first input is 1 and the second input is 4. These values select the table value at the intersection of the first row (row parameter value 1) and second column (column parameter value 4).

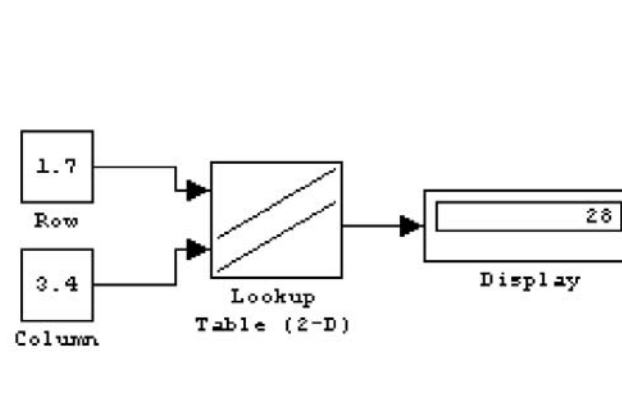
	3	4
1	10	20
2	30	40



In the second figure, the first input is 1.7 and the second is 3.4. These values cause the block to interpolate between row and column values,

as shown in the table at the left. The value at the intersection (28) is the output value.

	3	3.4	4
1	10	14	20
1.7	24	28	34
2	30	34	40



Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of one input if the other is a vector
Dimensionalized	Yes
Zero-Crossing Detection	No

See Also

Lookup Table, Lookup Table (n-D)

Lookup Table (n-D)

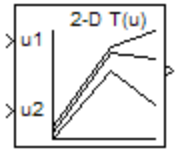
Purpose

Approximate N-dimensional function

Library

Lookup Tables

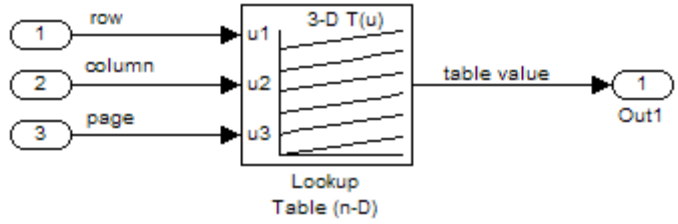
Description



How This Block Differs from Other Lookup Table Blocks

The Lookup Table (n-D) block evaluates a sampled representation of a function in N variables $y = F(x1, x2, x3, \ldots, xn)$, where the function F can be empirical. The block maps inputs to an output value by looking up or interpolating a table of values you define with block parameters. The block supports flat (constant), linear, and cubic-spline interpolation methods. You can apply these methods to a table of any dimension from 1 through 30.

In the following block, the first input identifies the first dimension (row) breakpoints, the second input identifies the second dimension (column) breakpoints, and so on.



See "How to Rotate a Block" in the *Simulink User's Guide* for a description of the port order for various block orientations.

Specification of Breakpoint and Table Data

The following block parameters define the breakpoint and table data.

Block Parameter	Purpose
Number of table dimensions	Specifies the number of dimensions of your lookup table.

Block Parameter	Purpose
Table and Breakpoints (BP) > BP	Specifies a breakpoint vector that corresponds to each dimension of your lookup table.
Table and Breakpoints (BP) > Table	Defines the associated set of output values.

Tip Evenly-spaced breakpoints can make Real-Time Workshop generated code division-free. For more information, see `fixpt_evenspace_cleanup` in the Simulink documentation and "Identify questionable fixed-point operations" in the Real-Time Workshop documentation.

How the Block Generates Output

The Lookup Table (n-D) block generates output by looking up or estimating table values based on the input values:

If the inputs...	The Lookup Table (n-D) block...
Match the values of indices in breakpoint data sets	Outputs the table value at the intersection of the row, column, and higher dimension breakpoints
Do not match the values of indices in breakpoint data sets, but are within range	Interpolates appropriate table values, using the **Interpolation method** you select
Do not match the values of indices in breakpoint data sets, and are out of range	Extrapolates the output value, using the **Extrapolation method** you select

Other Blocks That Perform Equivalent Operations

You can use the Interpolation Using Prelookup block with the Prelookup block to perform the equivalent operation of a Lookup Table (n-D) block. This combination of blocks offers greater flexibility that can result in more efficient simulation performance for linear interpolations in certain circumstances.

When the lookup operation is a simple array access that does not require interpolation, use the Direct Lookup Table (n-D) block. For example, if you have an integer value `k` and you want the `k`th element of a table, *y = table(k)*, interpolation is unnecessary.

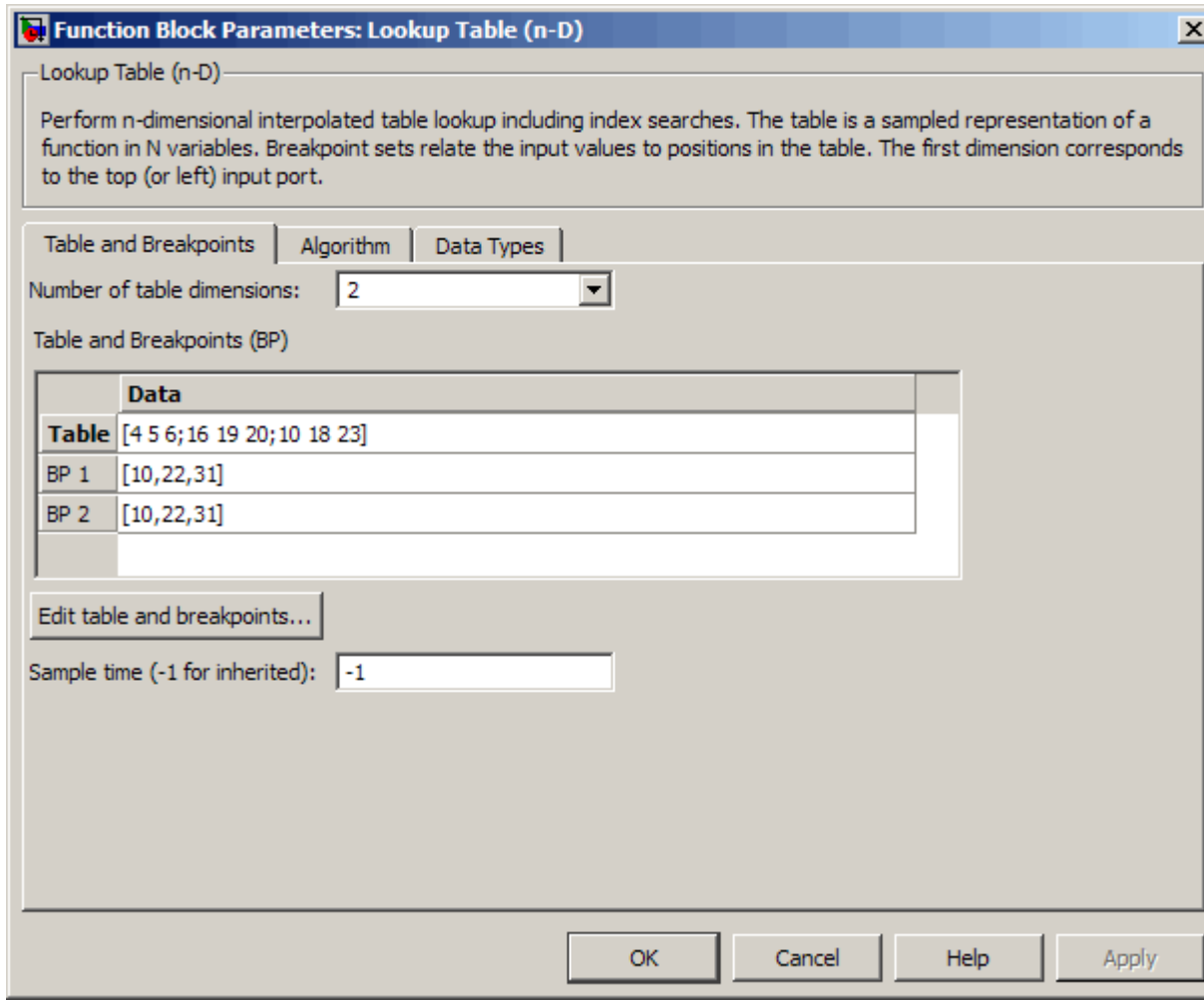
Data Type Support

The Lookup Table (n-D) block supports all numeric data types supported by Simulink software, including fixed-point data types. For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the Simulink documentation.

Inputs for indexing must be real, but table data can be complex.

Parameters and Dialog Box

The **Table and Breakpoints** pane of the Lookup Table (n-D) block dialog box appears as follows:



Number of table dimensions

Enter the number of dimensions of the lookup table by specifying an integer from `1` to `30`. This parameter determines:

- The number of independent variables for the table and the number of block inputs
- The number of breakpoint sets to specify in **Table and Breakpoints (BP)**

Table and Breakpoints (BP)

In the **Table** row, enter the table of output values.

Tip During simulation, the matrix size must match the dimensions defined by the **Number of table dimensions** parameter. However, during block diagram editing, you can enter an empty matrix (specified as `[]`) or an undefined workspace variable. This technique lets you postpone specifying a correctly dimensioned matrix for the table data and continue editing the block diagram. For information about how to construct multidimensional arrays in MATLAB software, see "Multidimensional Arrays" in the MATLAB online documentation.

In each **BP** row, enter the breakpoint set that corresponds to each dimension of table data. For each dimension, specify breakpoints as a `1-by-n` or `n-by-1` vector whose values are strictly monotonically increasing.

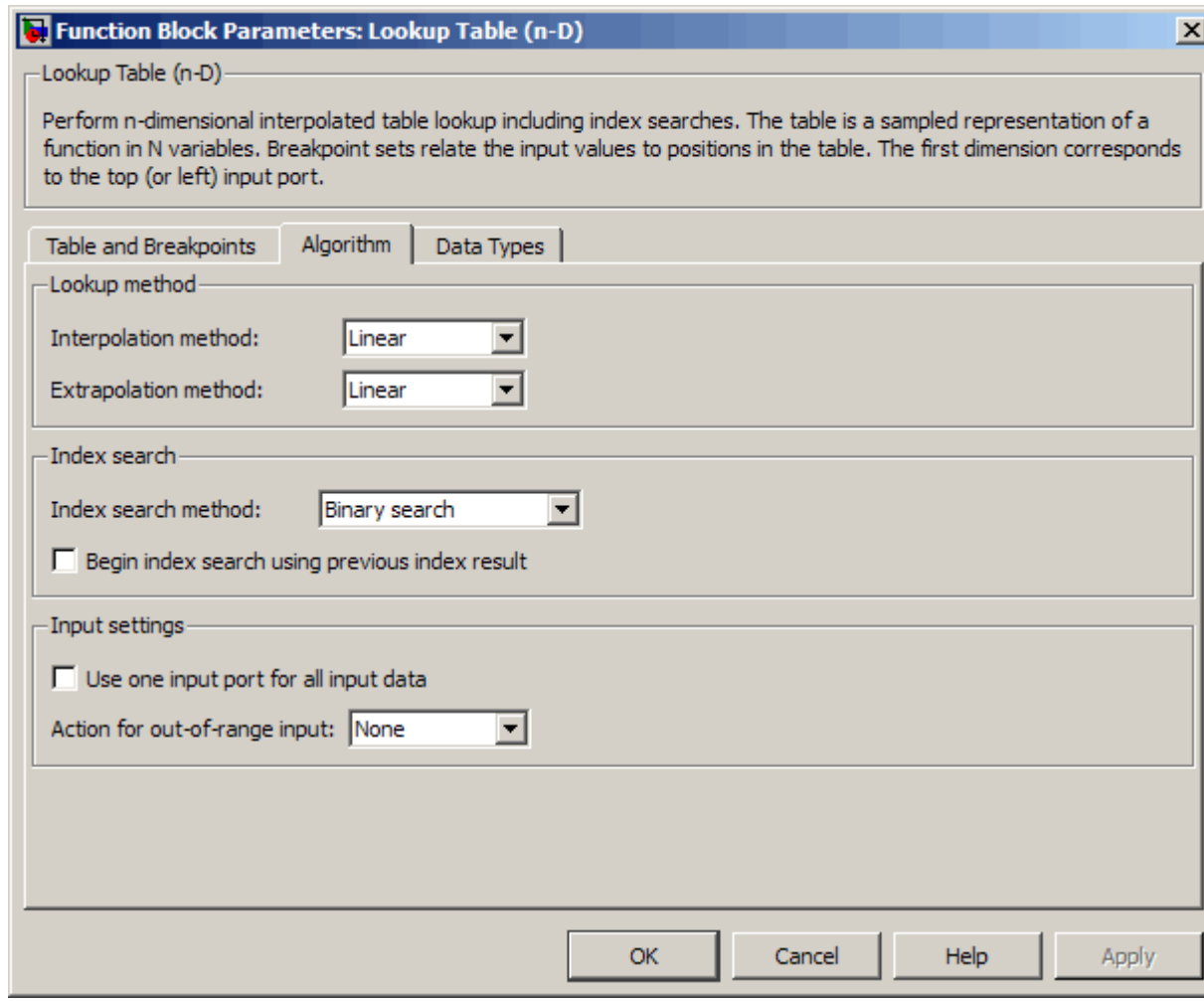
Edit table and breakpoints

Click this button to open the Lookup Table Editor. For more information, see "Lookup Table Editor" in the *Simulink User's Guide*.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in the *Simulink User's Guide* for more information.

The **Algorithm** pane of the Lookup Table (n-D) block dialog box appears as follows:



Interpolation method

Select `None - Flat`, `Linear` (the default), or `Cubic spline`. See "Interpolation Methods" in the *Simulink User's Guide* for more information.

Tip If you select `Cubic spline`, the block supports only scalar signals. The other interpolation methods support nonscalar signals.

Extrapolation method

Select `None - Clip`, `Linear` (the default), or `Cubic spline`. See "Extrapolation Methods" in the *Simulink User's Guide* for more information.

Tip To select `Cubic spline` for **Extrapolation method**, you must also select `Cubic spline` for **Interpolation method**.

Use last table value for inputs at or above last breakpoint

Specify the indexing convention that the block uses to address the last element of a breakpoint set and its corresponding table value.

Check Box Selection	Index That the Block Uses	Interval Fraction
Yes	The last element of a breakpoint set	0
No	The next-to-last element of a breakpoint set	1

This parameter is visible only when:

- **Interpolation method** is `Linear`.

- **Extrapolation method** is `None - Clip`.

Index search method

Select `Evenly spaced points`, `Linear search`, or `Binary search` (the default). Each search method has speed advantages in different circumstances:

- For evenly spaced breakpoint sets (for example, 10, 20, 30, and so on), you achieve optimal speed by selecting `Evenly spaced points` to calculate table indices.

  This algorithm uses only the first two breakpoints of a set to determine the offset and spacing of the remaining points.

- For unevenly spaced breakpoint sets, follow these guidelines:

  — If input signals do not vary much between time steps, selecting `Linear search` with **Begin index search using previous index result** produces the best performance.

  — If input signals jump more than one or two table intervals per time step, selecting `Binary search` produces the best performance.

A suboptimal choice of index search method can lead to slow performance of models that rely heavily on lookup tables.

Note Real-Time Workshop generated code stores only the first breakpoint, the spacing, and the number of breakpoints when:

- The breakpoint data is not tunable.
- The index search method is `Evenly spaced points`.

Begin index search using previous index result

Select this check box when you want the block to start its search using the index found at the previous time step. For inputs that change slowly with respect to the interval size, enabling this option can improve performance. Otherwise, the linear search

and binary search methods can take longer, especially for large breakpoint sets.

Use one input port for all input data

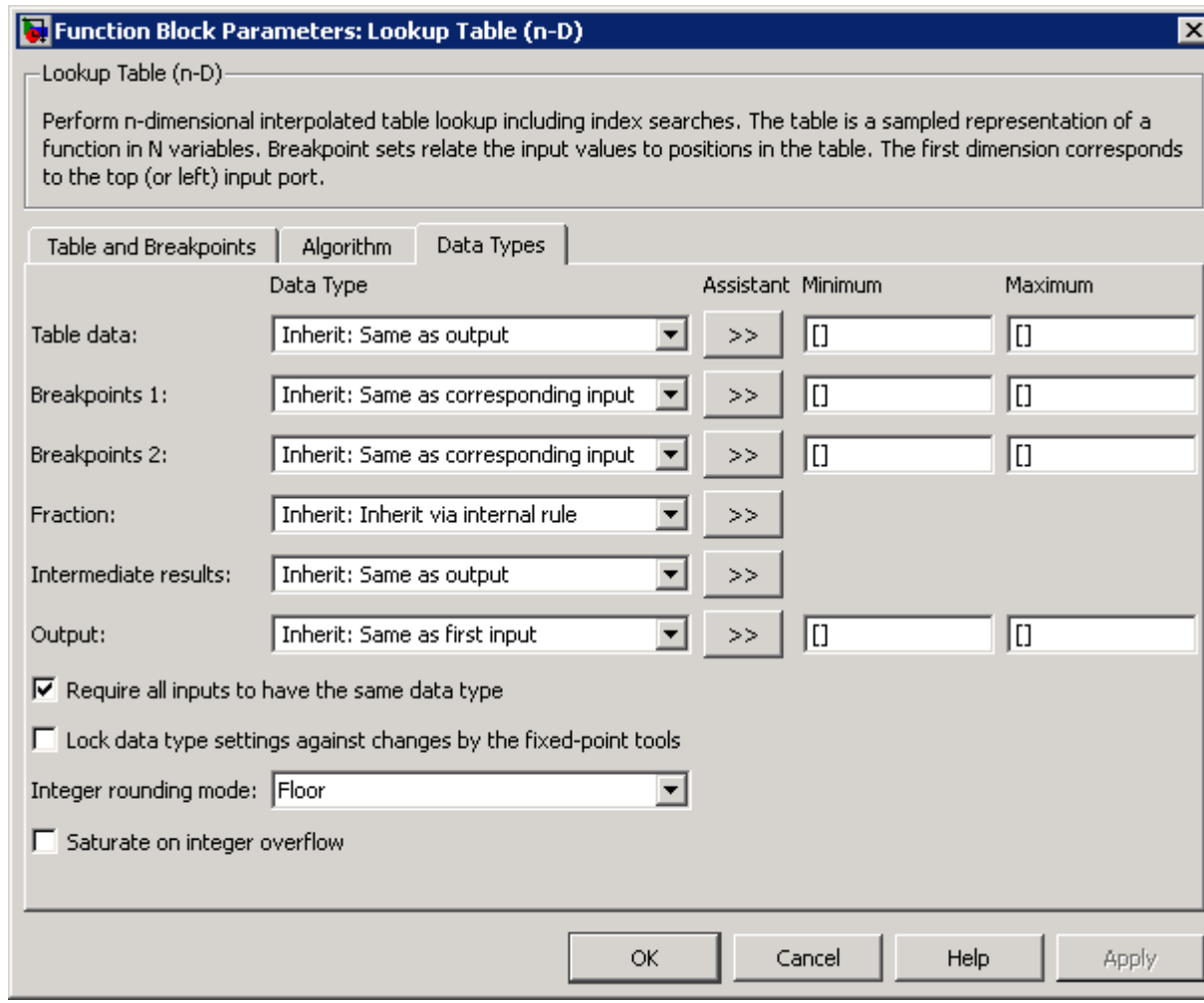
Instead of one input port per independent variable, use only one input port that expects a signal that is N elements wide for an N-dimensional table. This option is useful for removing line clutter on a block diagram with many lookup tables.

Action for out-of-range input

Specifies whether to produce a warning or error message when the input is out of range. Options include:

- `None` — the default, which means 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

The **Data Types** pane of the Lookup Table (n-D) block dialog box appears as follows:



Note If you have more than two sets of breakpoint data, the dialog box expands to show additional data type options. Up to 30 breakpoint data type specifications can appear.

Table data

Specify the table data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as output`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Table data** parameter.

Tip Specify a table data type different from the output data type for these cases:

- Lower memory requirement for storing table data that uses a smaller type than the output signal
- Sharing of prescaled table data between two Lookup Table (n-D) blocks with different output data types
- Sharing of custom storage table data in Real-Time Workshop generated code for blocks with different output data types

Minimum (for **Table data**)
Specify the minimum value for table data. The default value, `[]`, is equivalent to `-Inf`.

Maximum (for **Table data**)
Specify the maximum value for table data. The default value, `[]`, is equivalent to `Inf`.

Breakpoints 1
Specify the data type for the first set of breakpoint data. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as corresponding input`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Breakpoints 1** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Tip Specify a breakpoint data type different from the corresponding input data type for these cases:

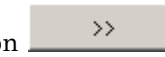
- Lower memory requirement for storing breakpoint data that uses a smaller type than the input signal
- Sharing of prescaled breakpoint data between two Lookup Table (n-D) blocks with different input data types
- Sharing of custom storage breakpoint data in Real-Time Workshop generated code for blocks with different input data types

Minimum (for **Breakpoints 1**)
Specify the minimum value that the first set of breakpoint data can have. The default value, `[]`, is equivalent to `-Inf`.

Maximum (for **Breakpoints 1**)
Specify the maximum value that the first set of breakpoint data can have. The default value, `[]`, is equivalent to `Inf`.

Breakpoints 2
Specify the data type for the second set of breakpoint data. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as corresponding input`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Breakpoints 2** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Tip Specify a breakpoint data type different from the corresponding input data type for these cases:

- Lower memory requirement for storing breakpoint data that uses a smaller type than the input signal
- Sharing of prescaled breakpoint data between two Lookup Table (n-D) blocks with different input data types
- Sharing of custom storage breakpoint data in Real-Time Workshop generated code for blocks with different input data types

---

Minimum (for **Breakpoints 2**)

Specify the minimum value that the second set of breakpoint data can have. The default value, `[]`, is equivalent to `-Inf`.

Maximum (for **Breakpoints 2**)

Specify the maximum value that the second set of breakpoint data can have. The default value, `[]`, is equivalent to `Inf`.

Fraction

Specify the fraction data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object

- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Fraction** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Intermediate results

Specify the intermediate results data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as output`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Intermediate results** parameter.

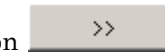
Tip Use this parameter to specify higher (or lower) precision for internal computations than for table data or output data.

Output

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`

- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Minimum (for **Output**)

Specify the minimum value that the block outputs. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Maximum (for **Output**)

Specify the maximum value that the block outputs. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Require all inputs to have the same data type

Select to require all inputs to have the same data type.

Lock data type settings against changes by the fixed-point tools
Select to lock all data type settings of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode
Select the rounding mode for fixed-point lookup table calculations that occur during simulation or execution of code generated from the model. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

This option does not affect rounding of values of block parameters. Simulink software rounds such values to the nearest representable integer value. To control the rounding of a block parameter, enter an expression using a MATLAB rounding function into the edit field on the block dialog box.

Saturate on integer overflow
Select to have overflows saturate. Otherwise, they wrap.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Tip If you save your model as version R2009a or earlier, this check box setting has no effect and no saturation code appears. This behavior preserves backward compatibility.

Examples

For an example of entering breakpoint and table data, see "Entering Data in a Block Parameter Dialog Box" in the *Simulink User's Guide*.

For an example that illustrates linear interpolation and extrapolation methods of this block, see "Example of a Logarithm Lookup Table" in the *Simulink User's Guide*.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes
Dimensionalized	Yes, if you do not select `Cubic spline` for **Interpolation method**
Zero-Crossing Detection	No

See Also

Lookup Table Dynamic

Lookup Table Dynamic

Purpose

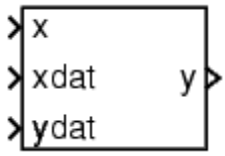
Approximate one-dimensional function using dynamic table

Library

Lookup Tables

Description

How This Block Differs from Other Lookup Table Blocks



The Lookup Table Dynamic block computes an approximation to a function `y = f(x)` using `xdat` and `ydat` vectors. The lookup method can use interpolation, extrapolation, or the original values of the input.

Using the Lookup Table Dynamic block, you can change the table data without stopping the simulation. For example, you can incorporate new table data if the physical system you are simulating changes.

Inputs for Breakpoint and Table Data

The `xdat` vector is the breakpoint data, which must be *strictly monotonically increasing*. The value of the next element in the vector must be greater than the value of the preceding element after conversion to a fixed-point data type. Due to quantization, the `xdat` vector can be strictly monotonic for a floating-point data type, but not after conversion to a fixed-point data type.

The `ydat` vector is the table data, which is an evaluation of the function at the breakpoint values.

You define the lookup table by feeding `xdat` and `ydat` as 1-by-n vectors to the block. To reduce ROM usage by the generated code for this block, you can use different data types for the `xdat` breakpoint data and the `ydat` table data. However, these restrictions apply:

- The `xdat` breakpoint data and the `x` input vector must have the same sign, the same bias, and the same fractional slope. Also, the precision and range for the `x` input vector must be greater than or equal to the precision and range for the `xdat` breakpoint data.
- The `ydat` table data and the `y` output vector must have the same sign, the same bias, and the same fractional slope.

Tip Evenly-spaced breakpoints can make Real-Time Workshop generated code division-free. For more information, see `fixpt_evenspace_cleanup` in the Simulink documentation and "Identify questionable fixed-point operations" in the Real-Time Workshop documentation.

How the Block Generates Output

The block uses the input values to generate output using the method you select for the **Lookup Method** parameter:

Lookup Method	Action by the Block
`Interpolation-Extrapolation`	Performs linear interpolation and extrapolation of the inputs. • If the input matches a breakpoint, the output is the corresponding element in the table data. • If the input does not match a breakpoint, the block performs linear interpolation between two elements of the table to determine the output. If the input falls outside the range of breakpoint values, the block extrapolates using the first two or last two points. **Note** If you select this lookup method, Real-Time Workshop software cannot generate code for this block.
`Interpolation-Use End Values`	Performs linear interpolation but does not extrapolate outside the end points of the breakpoint data. Instead, the block uses the end values.

Lookup Method	Action by the Block
`Use Input Nearest`	Finds the element in `xdat` nearest the current input. The corresponding element in `ydat` is the output.
`Use Input Below`	Finds the element in `xdat` nearest and below the current input. The corresponding element in `ydat` is the output. If there is no element in `xdat` below the current input, then the block finds the nearest element.
`Use Input Above`	Finds the element in `xdat` nearest and above the current input. The corresponding element in `ydat` is the output. If there is no element in `xdat` above the current input, then the block finds the nearest element.

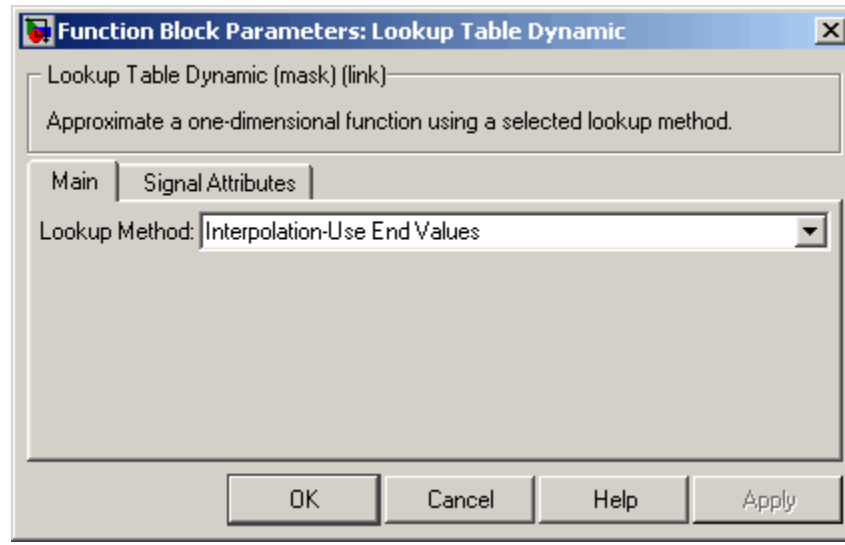
Note The `Use Input Nearest`, `Use Input Below`, and `Use Input Above` methods perform the same action when the input `x` matches a breakpoint value.

Data Type Support

The Lookup Table Dynamic block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box

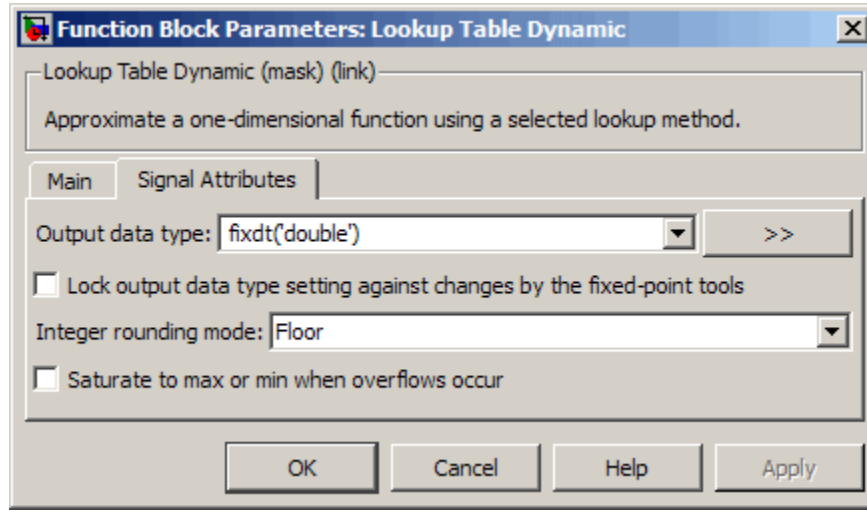
The **Main** pane of the Lookup Table Dynamic block dialog box appears as follows:



Lookup Method

Specify the lookup method. For details, see "How the Block Generates Output" on page 2-695.

The **Signal Attributes** pane of the Lookup Table Dynamic block dialog box appears as follows:



Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt('double')`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate. Otherwise, they wrap.

Examples

For an example of entering breakpoint and table data, see "Entering Data Using the Lookup Table Dynamic Block's Inports" in the *Simulink User's Guide*.

For an example that illustrates the lookup methods that this block supports, see "Example Output for Lookup Methods" in the *Simulink User's Guide*.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	No
Zero-Crossing Detection	No

See Also

Lookup Table (n-D)

Magnitude-Angle to Complex

Purpose

Convert magnitude and/or a phase angle signal to complex signal

Library

Math Operations

Description

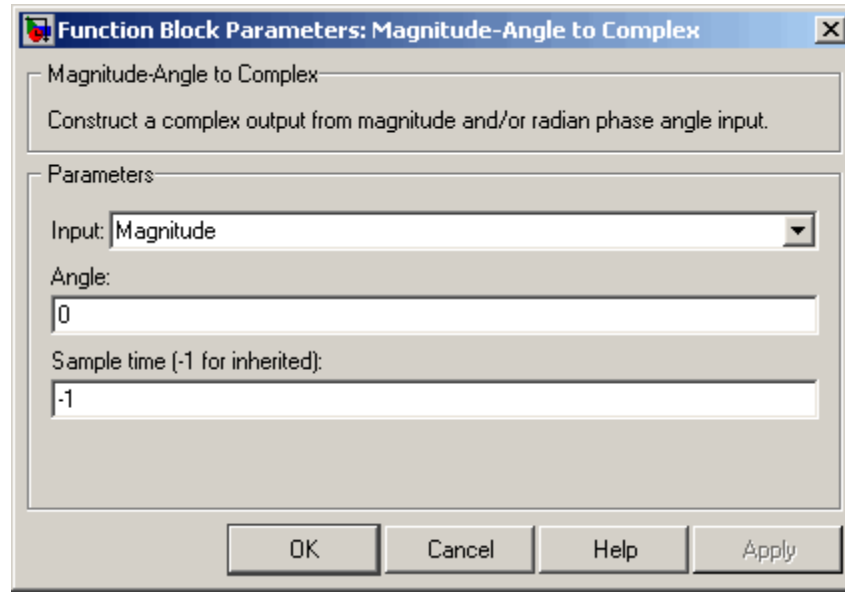
The Magnitude-Angle to Complex block converts magnitude and/or phase angle inputs to a complex-valued output signal. The inputs must be real-valued signals of type `double` or `single`. The angle input is assumed to be in radians. The complex output signal has the same data type as the block inputs.

The inputs can both be signals 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 an array of complex signals. The elements of a magnitude input vector are mapped to magnitudes of the corresponding complex output elements. An angle input vector is similarly mapped to the angles of the complex output signals. If one input is a scalar, it is mapped to the corresponding component (magnitude or angle) of all the complex output signals.

Data Type Support

See the preceding block description.

Parameters and Dialog Box



Input

Specifies the kind of input: a magnitude input, an angle input, or both.

Angle (Magnitude)

If the input is an angle signal, specifies the constant magnitude of the output signal. If the input is a magnitude, specifies the constant phase angle in radians of the output signal.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the 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
Multidimensionalized	Yes
Zero Crossing	No

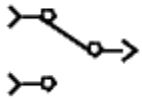
Purpose

Switch between two inputs

Library

Signal Routing

Description



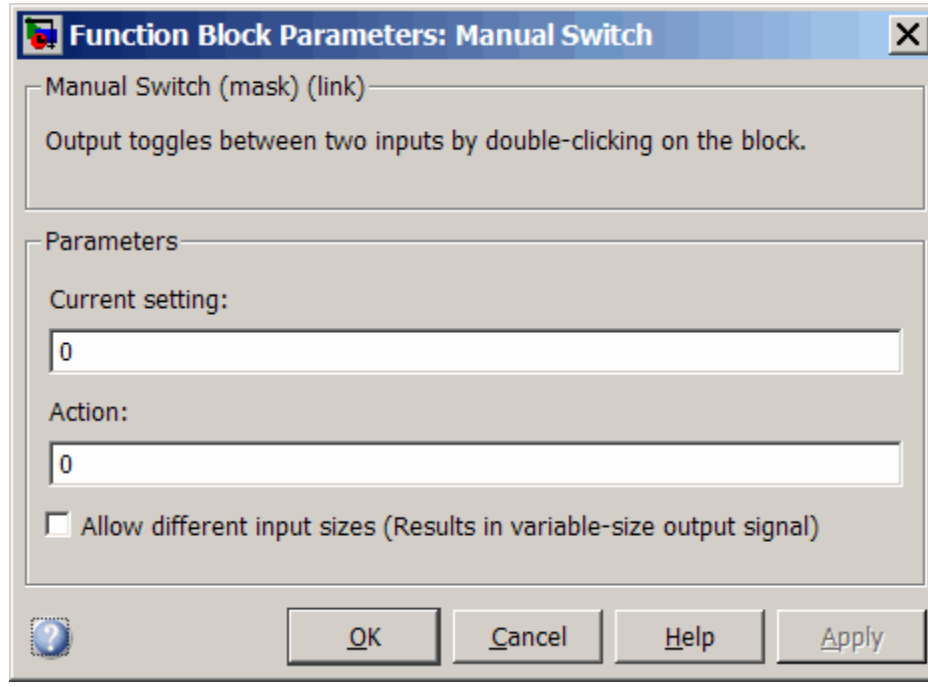
The Manual Switch block is a toggle switch that selects one of its two inputs to pass through to the output. To toggle between inputs, double-click the block (there is no dialog box). The block propagates the selected input to the output, while the block discards the unselected input. You can interactively control the signal flow by setting the switch before you start the simulation or by changing the switch while the simulation is executing. The Manual Switch block retains its current state when you save the model.

Data Type Support

The Manual Switch block accepts real or complex signals of any data type that Simulink supports, including fixed-point and enumerated data types. For more information, see “Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box

Double-clicking the Manual Switch block toggles the input. To open the Function Block Parameters dialog box, right-click the block, and then select `Mask Parameters`.



- "Current setting" on page 2-705
- "Action" on page 2-706
- "Allow different input sizes" on page 2-707

Current setting

This parameter tracks the current state of the Manual Switch block and cannot be modified by the user.

Action

This parameter tracks the current state of the Manual Switch block and cannot be modified by the user

Allow different input sizes

Select this check box to allow input signals with different sizes.

Settings

Default: Off

 On

Allows input signals with different sizes, and propagate the input signal size to the output signal.

 Off

Requires that all input signals be the same size.

Command-Line Information

Parameter: `AllowDiffInputSignals`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	N/A
Dimensionalized	Yes
Multidimensionalized	Yes
Zero Crossing	No

Math Function

Purpose

Perform mathematical function

Library

Math Operations

Description

The Math Function block performs numerous common mathematical functions.

You can select one of the following functions from the **Function** parameter list.

Function	Description	Mathematical Expression	MATLAB Equivalent
`exp`	Exponential	e^u	`exp`
`log`	Natural logarithm	$\ln u$	`log`
`10^u`	Power of base 10	10^u	`10.^u` (see `power`)
`log10`	Common (base 10) logarithm	$\log u$	`log10`
`magnitude^2`	Complex modulus	$\|u\|^2$	`(abs(u)).^2` (see `abs` and `power`)
`square`	Power 2	u^2	`u.^2` (see `power`)
`sqrt`	Square root	$u^{0.5}$	`sqrt`
`1/sqrt`	Reciprocal of a square root	$u^{-0.5}$	—
`pow`	Power	u^v	`power`
`conj`	Complex conjugate	$\bar{u}$	`conj`
`reciprocal`	Reciprocal	$1/u$	`1./u` (see `rdivide`)
`hypot`	Square root of sum squares	$(u^2+v^2)^{0.5}$	`hypot`

Function	Description	Mathematical Expression	MATLAB Equivalent
rem	Remainder after division	—	rem
mod	Modulus after division	—	mod
transpose	Transpose	u^T	u.' (see arithmetic operators)
hermitian	Complex conjugate transpose	u^H	u' (see arithmetic operators)

The block output is the result of the operation of the function on the input or inputs. The functions support the following types of operations.

Function	Scalar Operations	Element-Wise Vector and Matrix Operations	Vector and Matrix Operations
exp	yes	yes	—
log	yes	yes	—
10^u	yes	yes	—
log10	yes	yes	—
magnitude^2	yes	yes	—
square	yes	yes	—
sqrt	yes	yes	—
1/sqrt	yes	yes	—
pow	yes	yes	—
conj	yes	yes	—
reciprocal	yes	yes	—

Function	Scalar Operations	Element-Wise Vector and Matrix Operations	Vector and Matrix Operations
`hypot`	yes, on two inputs	yes, on two inputs (two vectors or two matrices of the same size, a scalar and a vector, or a scalar and a matrix)	—
`rem`	yes, on two inputs	yes, on two inputs (two vectors or two matrices of the same size, a scalar and a vector, or a scalar and a matrix)	—
`mod`	yes, on two inputs	yes, on two inputs (two vectors or two matrices of the same size, a scalar and a vector, or a scalar and a matrix)	—
`transpose`	yes	—	yes
`hermitian`	yes	—	yes

The name of the function appears on the block. The appropriate number of input ports appears automatically.

Tip Use the Math Function block instead of the Fcn block when you want vector or matrix output, because the Fcn block produces only scalar output.

Data Type Support

The following table shows the input data types that each function of the block can support.

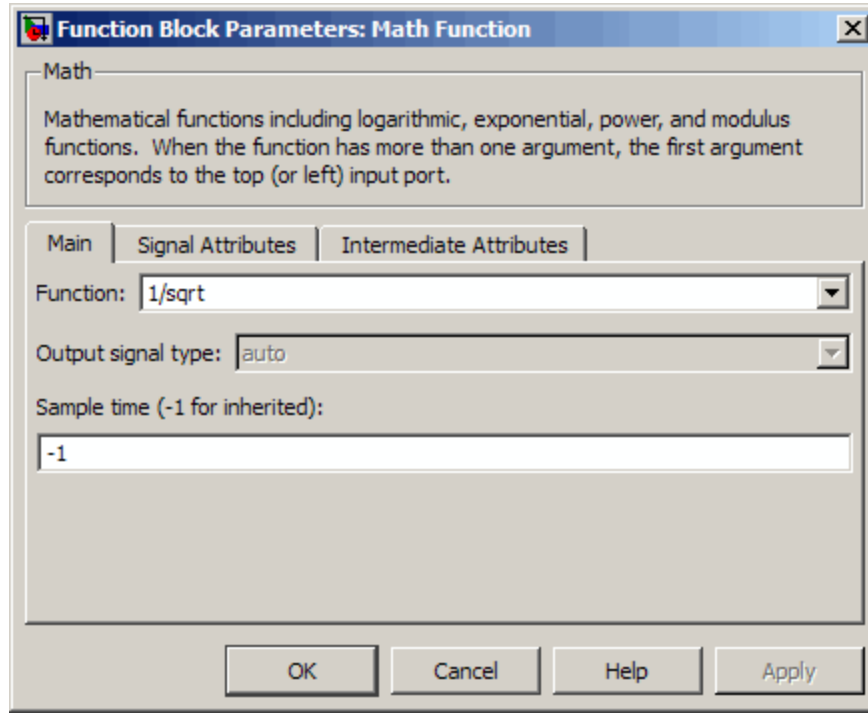
Function	single	double	boolean	built-in integer	fixed point
`exp`	yes	yes	—	—	—
`log`	yes	yes	—	—	—
`10^u`	yes	yes	—	—	—
`log10`	yes	yes	—	—	—
`magnitude^2`	yes	yes	—	yes	yes
`square`	yes	yes	—	yes	yes
`sqrt`	yes	yes	—	yes	yes
`1/sqrt`	yes	yes	—	yes	yes
`pow`	yes	yes	—	—	—
`conj`	yes	yes	—	yes	yes
`reciprocal`	yes	yes	—	yes	yes
`hypot`	yes	yes	—	—	—
`rem`	yes	yes	—	yes	—
`mod`	yes	yes	—	yes	—
`transpose`	yes	yes	yes	yes	yes
`hermitian`	yes	yes	—	yes	yes

All supported modes accept both real and complex inputs, except for `reciprocal`, `sqrt`, and `1/sqrt`, which do not accept complex fixed-point inputs. Also, `sqrt` and `1/sqrt` do not accept fixed-point inputs that are negative or have nontrivial slope and nonzero bias.

The block output is real or complex, depending on what you select for **Output signal type**. For `1/sqrt`, the output is always real and you cannot change the setting for **Output signal type**.

Parameters and Dialog Box

The **Main** pane of the Math Function block dialog box appears as follows:



Function

Specify the mathematical function. See Description for more information about the options for this parameter.

Output signal type

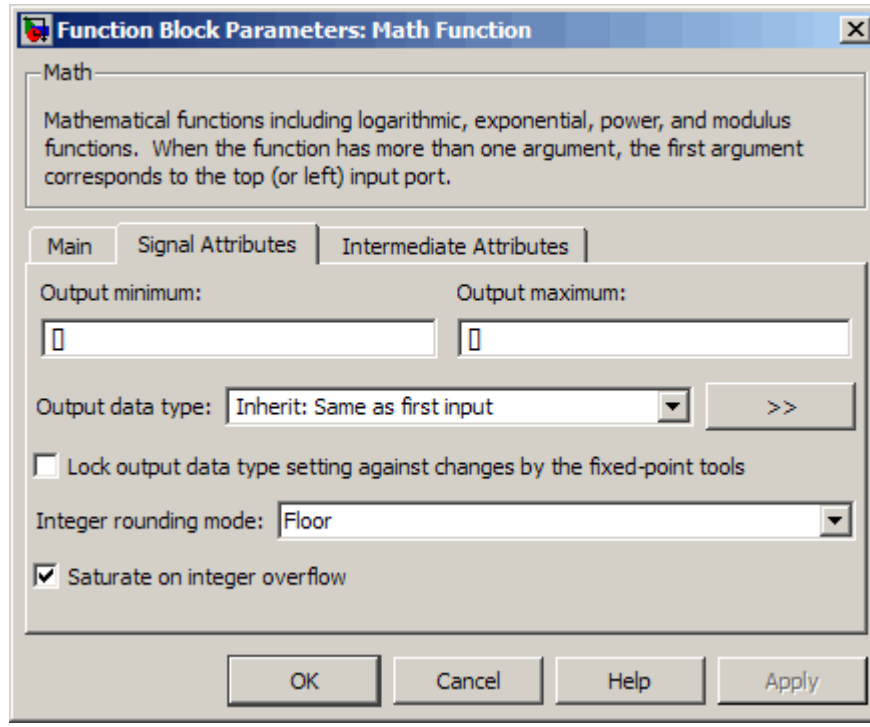
Specify the output signal type of the Math Function block as `real`, `complex`, or `auto`.

	Input	Output Signal Type		
Function	**Signal**	**Auto**	**Real**	**Complex**
exp, log, 10u, log10, square, sqrt, pow, reciprocal, conjugate, transpose, hermitian	real complex	real complex	real error	complex complex
magnitude squared	real complex	real real	real real	complex complex
hypot, rem, mod	real complex	real error	real error	complex error
1/sqrt	real complex	real error	Not available	Not available

Sample time (-1 for inherited)

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

The **Signal Attributes** pane of the Math Function block dialog box appears as follows:



Note Some parameters on this pane are available only when the function you select in the **Function** parameter supports fixed-point data types.

Output minimum

Specify the minimum value that the block can output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block can output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode

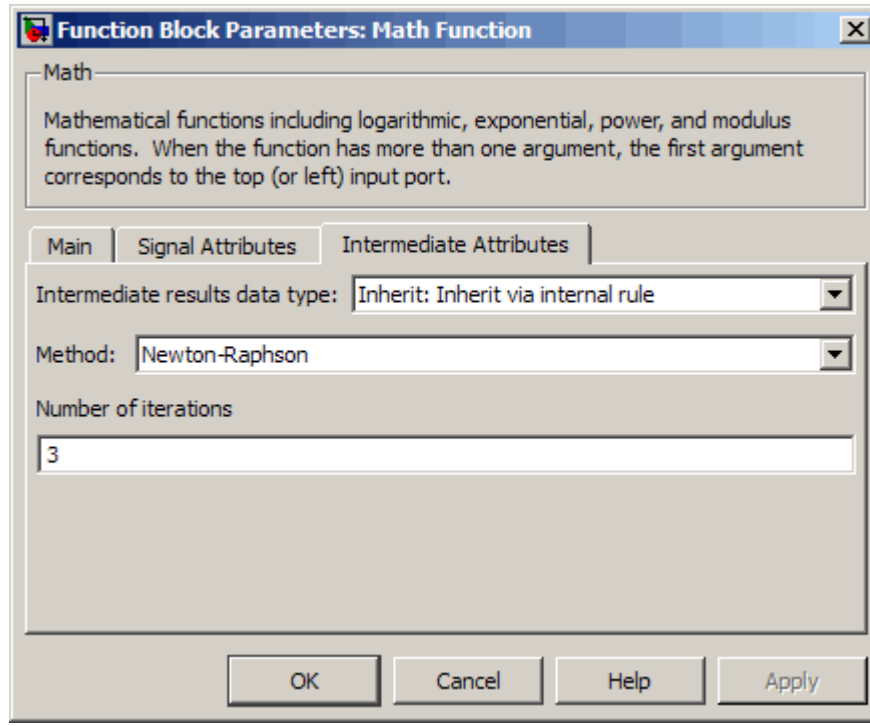
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate on integer overflow

If selected, fixed-point overflows saturate. Otherwise, overflows wrap.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

The **Intermediate Attributes** pane of the Math Function block dialog box appears as follows:



Note The following parameters are available only if you select `1/sqrt` for the **Function** parameter.

Intermediate results data type

Specify the data type for intermediate results. You can set it to:

- `Inherit: Inherit via internal rule` (default)
- `Inherit: Inherit from input`
- `Inherit: Inherit from output`

Tip Do not select `Inherit: Inherit from output` under these conditions:

- You select `Newton-Raphson` to compute the reciprocal of a square root.
- The input data type is floating-point.
- The output data type is fixed-point.

Under these conditions, selecting `Inherit: Inherit from output` yields suboptimal performance and an error results.

To avoid this error, convert the input signal from a floating-point to fixed-point data type. For example, insert a Data Type Conversion block in front of the Math Function block to perform the conversion.

Method

Select the method for computing the reciprocal of a square root.

Method	Data Types Supported	When to Use This Method
`Newton-Raphson` (default)	Floating-point, fixed-point, and built-in integer types	You want a fast, approximate calculation.
`Exact`	Floating-point If you use a fixed-point or built-in integer type, an upcast to a floating-point type occurs.	You want an exact calculation. **Note** The input or output must be floating-point.

Number of iterations

Specify the number of iterations to perform the `Newton-Raphson` algorithm. The default value is 3.

This parameter is not available when you select `Exact` for the **Method**.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of the input when the function requires two inputs
Dimensionalized	Yes
Multidimensionalized	Yes, for all functions except `hermitian` and `transpose`
Zero-Crossing Detection	No

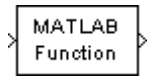
Purpose

Apply MATLAB function or expression to input

Library

User-Defined Functions

Description



The MATLAB Fcn block applies the specified MATLAB function or expression to the input. The output of the function must match the output dimensions of the block or an error occurs.

Here are some sample valid expressions for this block.

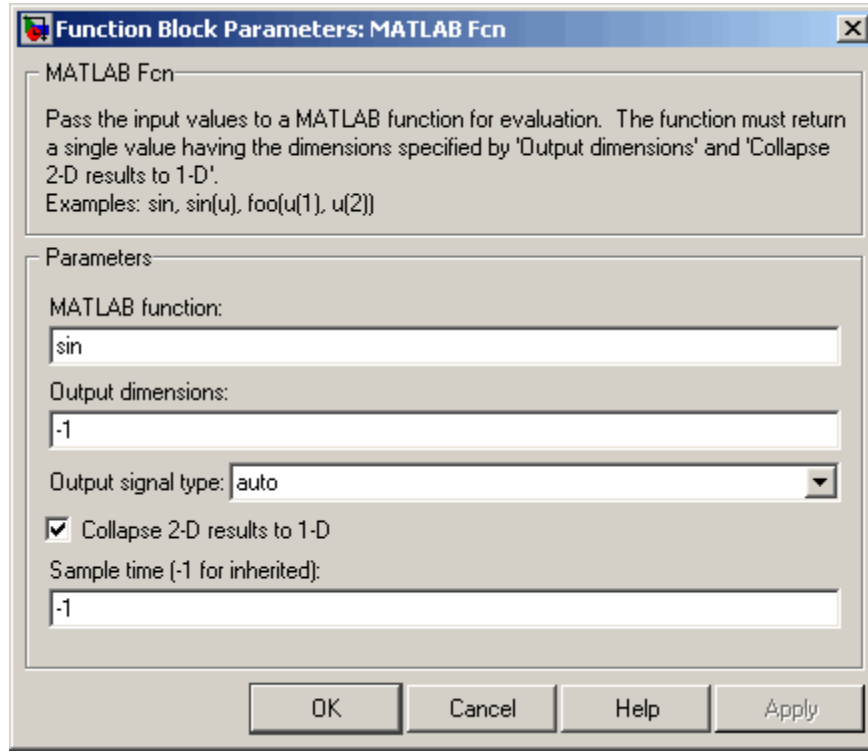
```
sin
atan2(u(1), u(2))
u(1)^u(2)
```

Note This block is slower than the Fcn block because it calls the MATLAB parser during each integration step. Consider using built-in blocks (such as the Fcn block or the Math Function block) instead, or writing the function as an M-file or MEX-file S-function, then accessing it using the S-Function block.

Data Type Support

The MATLAB Fcn block accepts one complex or real input of type `double` and generates real or complex output of type `double`, depending on the setting of the **Output signal type** parameter.

Parameters and Dialog Box



MATLAB function

The function or expression. If you specify a function only, it is not necessary to include the input argument in parentheses.

Output dimensions

Dimensions of the signal output by this block. If the output dimensions are to be the same as the dimensions of the input signal, specify -1. Otherwise, enter the dimensions of the output signal, e.g., 2 for a two-element vector. In either case, the output dimensions must match the dimensions of the value returned by the function or expression in the **MATLAB function** field.

Output signal type

The dialog allows you to select the output signal type of the MATLAB Fcn as `real`, `complex`, or `auto`. A value of `auto` sets the block's output type to be the same as the type of the input signal.

Collapse 2-D results to 1-D

Outputs a 2-D array as a 1-D array containing the 2-D array's elements in column-major order.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the 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

Matrix Concatenate, Vector Concatenate

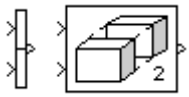
Purpose

Concatenate input signals of same data type to create contiguous output signal

Library

Math Operations

Description



The Concatenate block concatenates the signals at its inputs to create an output signal whose elements reside in contiguous locations in memory. This block operates in either vector or multidimensional array concatenation mode, depending on the setting of its **Mode** parameter. In either case, the block concatenates the inputs from the top to bottom, or left to right, input ports.

Vector Mode

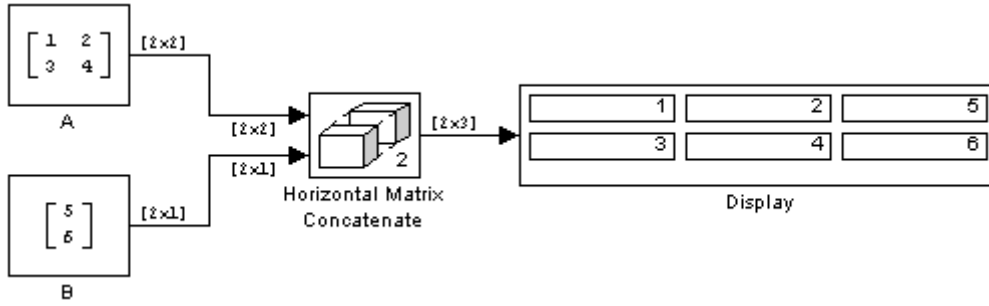
In vector mode, all input signals must be either vectors or row vectors [1xM matrices] or column vectors [Mx1 matrices] or a combination of vectors and either row or column vectors. The output is a vector if all inputs are vectors.

The output is a row or column vector if any of the inputs are row or column vectors, respectively.

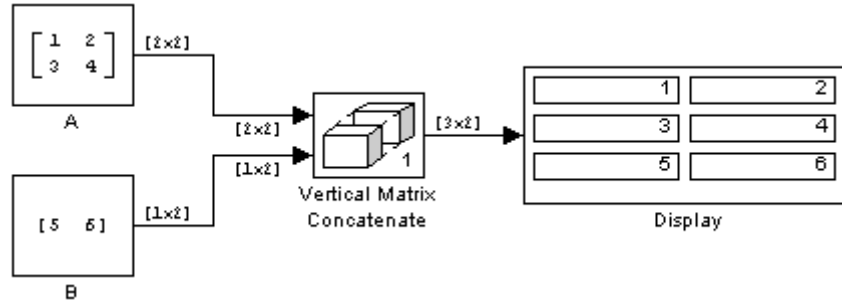
Multidimensional Array Mode

Multidimensional array mode accepts vectors and arrays of any size. It assumes that the trailing dimensions are all ones for input signals with lower dimensionality. For example, if the output is 4-D and the input is `[2x3]` (2-D) this block treats the input as `[2x3x1x1]`. The output is always an array. The block's **Concatenate dimension** parameter allows you to specify the output dimension along which the block concatenates its input arrays.

If you set the **Concatenate dimension** parameter to `2` and inputs are 2-D matrices, the block performs horizontal matrix concatenation and places the input matrices side-by-side to create the output matrix, for example:

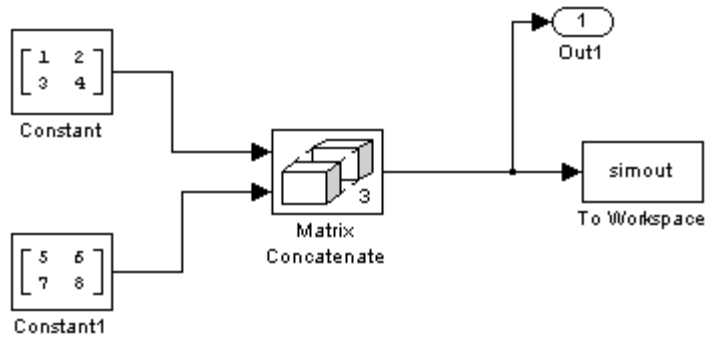


If you set the **Concatenate dimension** parameter to `1` and inputs are 2-D matrices, the block performs vertical matrix concatenation and stacks the input matrices on top of each other to create the output matrix, for example:



For horizontal concatenation, the input matrices must have the same column dimension. For vertical concatenation, the input matrices must have the same row dimension. All input signals must have the same dimension for all dimensions other than the concatenation dimensions.

If you set the **Mode** parameter to `Multidimensional array`, the **Concatenate dimension** parameter to `3`, and the inputs are 2-D matrices, the block performs multidimensional matrix concatenation, for example:



Data Type Support

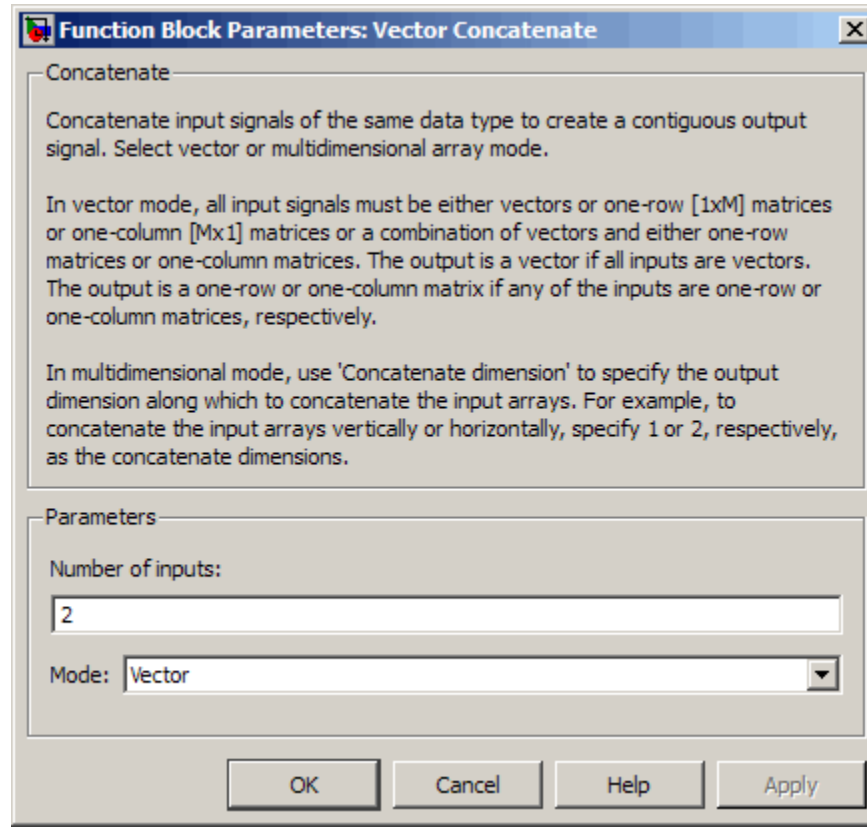
Accepts signals of any data type supported by Simulink software. All inputs must be of the same data type. Outputs have the same data type as the input.

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the "Working with Data" chapter of the Simulink documentation.

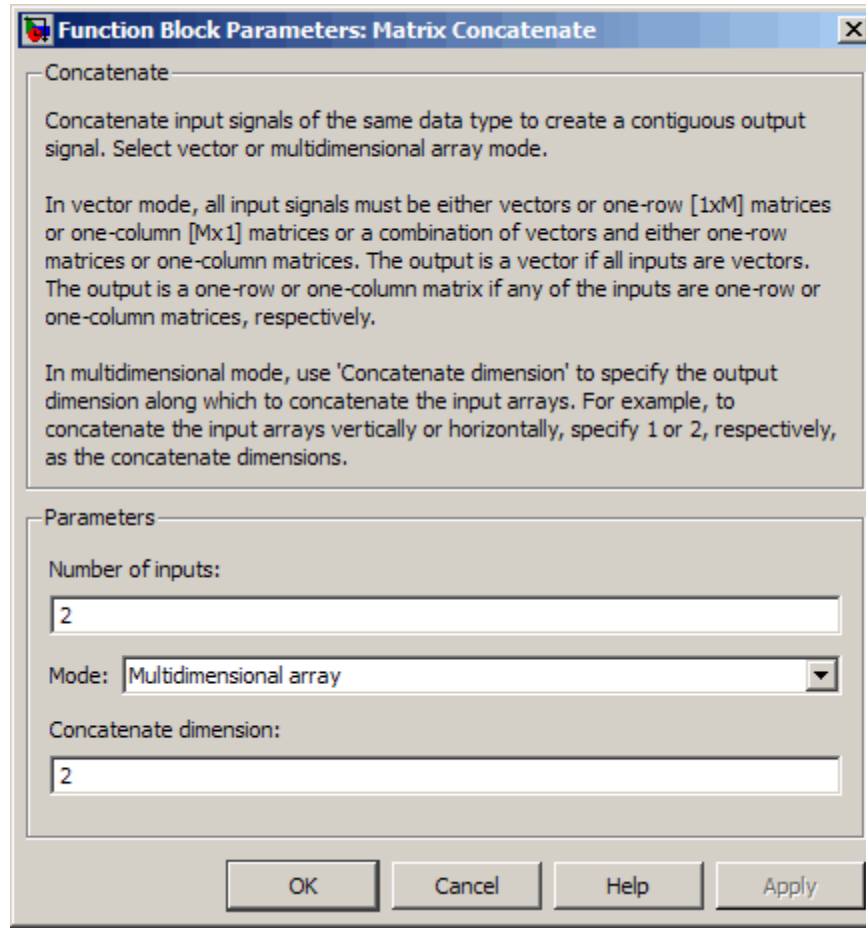
Parameters and Dialog Box

The parameters and dialog box differ, based on the mode in which the block is operating: vector or matrix. Most parameters exist in both modes.

The dialog box for the Vector Concatenate block appears as follows.



The dialog box for the Matrix Concatenate block appears as follows.


Function Block Parameters: Matrix Concatenate
Concatenate
Concatenate input signals of the same data type to create a contiguous output signal. Select vector or multidimensional array mode.
In vector mode, all input signals must be either vectors or one-row [1xM] matrices or one-column [Mx1] matrices or a combination of vectors and either one-row matrices or one-column matrices. The output is a vector if all inputs are vectors. The output is a one-row or one-column matrix if any of the inputs are one-row or one-column matrices, respectively.
In multidimensional mode, use 'Concatenate dimension' to specify the output dimension along which to concatenate the input arrays. For example, to concatenate the input arrays vertically or horizontally, specify 1 or 2, respectively, as the concatenate dimensions.
Parameters
Number of inputs:
2
Mode:
Multidimensional array
Concatenate dimension:
2
OK
Cancel
Help
Apply

Number of inputs

Specifies the number of inputs for the block.

Settings

Default: 2

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Mode

Select the type of concatenation that this block performs.

Settings

Default: `Vector` (for the Vector Concatenate block), `Multidimensional array` (for the Matrix Concatenate block)

`Vector`
: Perform vector concatenation (see "Vector Mode" on page 2-722 for details).

`Multidimensional array`
: Perform matrix concatenation (see "Multidimensional Array Mode" on page 2-722 for details).

Dependency

This parameter enables **Concatenate dimension**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Concatenate dimension

Specifies the output dimension along which to concatenate the input arrays.

Settings

Default: `2`

- Enter `1` to concatenate input arrays vertically.
- Enter `2` to concatenate input arrays horizontally.
- Enter a higher dimension to perform multidimensional concatenation on the inputs.

Dependency

Selecting `Multidimensional array` for **Mode** enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

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

See Also

`cat` in the MATLAB reference documentation

Memory

Purpose

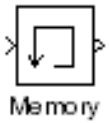
Output input from previous time step

Library

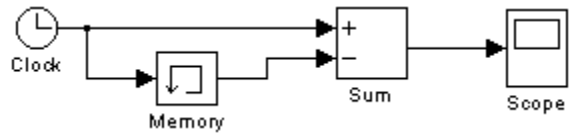
Discrete

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.



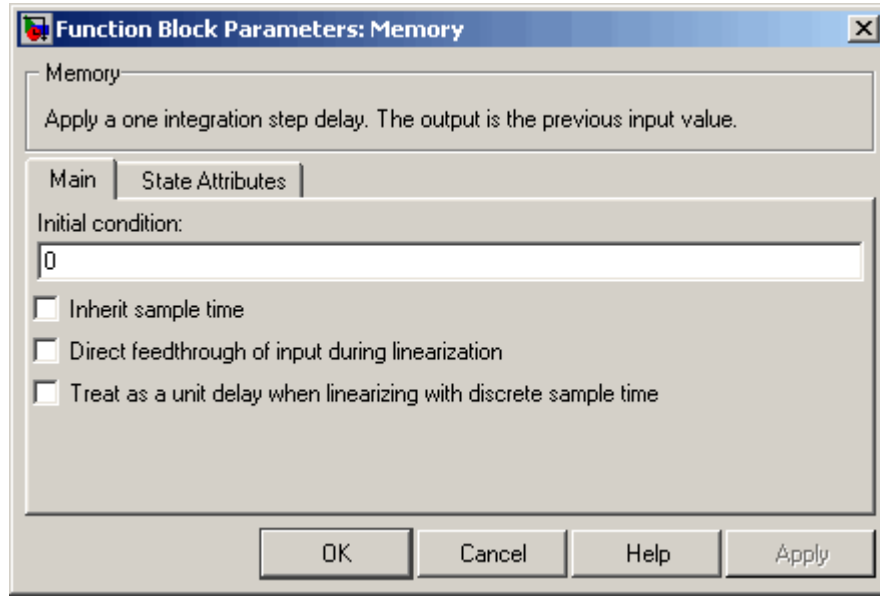
Note Avoid using the Memory block when integrating with `ode15s` or `ode113`, unless the input to the block does not change.

Data Type Support

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

For a discussion on the data types supported by Simulink software, 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 software 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 continuous but fixed in minor time step (`[0, 1]`). If the solver is a fixed-step solver, this `[0, 1]` sample time is converted to the solver's step size after sample time propagation.

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');
```

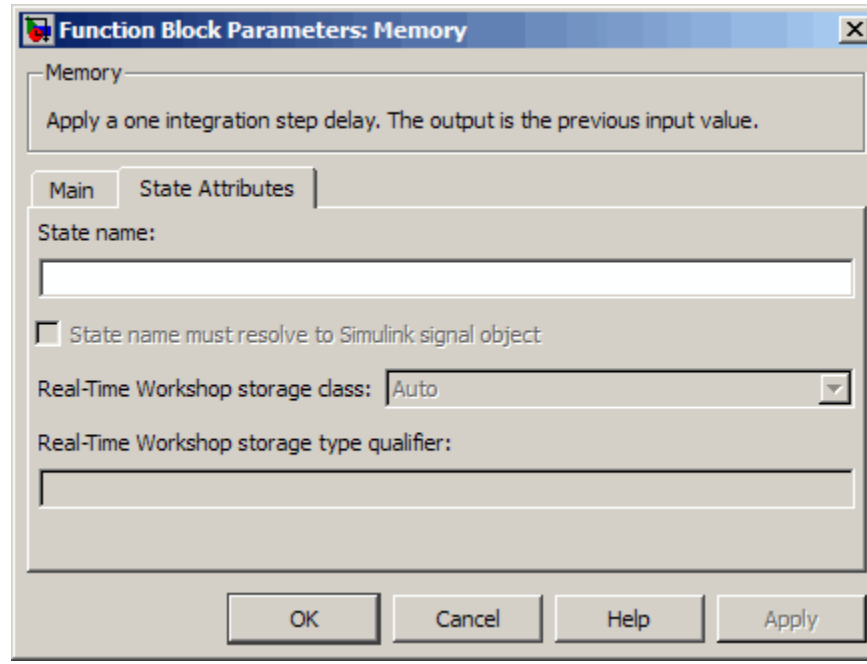
Next, terminate the compilation with the following command.

```
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.

Treat as a unit delay when linearizing with discrete sample time
Select this check box to linearize the Memory block to a unit delay when the Memory block is driven by a signal with a discrete sample time.

The **State Attributes** pane of the Memory block dialog box appears as follows:



State name

Use this parameter to assign a unique name to each state. The default is `' '`. If left blank, no name is assigned. Consider the following when using this parameter:

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'` .
- The state names apply only to the selected block.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces. For example, `{'a', 'b', 'c'}` . Each name must be unique.
- The number of states must be evenly divided by the number of state names. There can be fewer names than states, but there cannot be more names than states.

- For example, you can specify two names in a system with four states. Simulink software will assign the first name to the first two states and the second name to the last two.
- To assign state names with a variable that has been defined in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell, or structure.

This parameter enables **State name must resolve to Simulink signal object** when you click the **Apply** button.

State name must resolve to Simulink signal object

Select this checkbox to require that state name resolve to Simulink signal object. This check box is cleared by default.

This parameter is enabled by **State name**.

Selecting this check box enables **Real-Time Workshop storage class**.

Real-Time Workshop storage class

From the list, select state storage class.

`Auto`

Auto is the appropriate storage class for states that you do not need to interface to external code.

`ExportedGlobal`

State is stored in a global variable

`ImportedExtern`

`model``_private.h` declares the state as an extern variable.

`ImportedExternPointer`

`model``_private.h` declares the state as an extern pointer.

This parameter is enabled by **State name**.

Setting this parameter to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables **Real-Time Workshop storage type qualifier**.

During simulation, the block uses the following values:

- The initial value of the signal object to which the state name is resolved
- Min and Max values of the signal object

See "Block State Storage and Interfacing Considerations" in the Real-Time Workshop Workshop documentation for more information.

Bus Support

The Memory block is a bus-capable block. The input can be a virtual or nonvirtual bus signal subject to the following restrictions:

- **Initial condition** must be zero or a nonzero scalar.
- If **Initial condition** is zero and a **State name** is specified, the input cannot be a virtual bus.
- If **Initial condition** is a nonzero scalar, no **State name** can be specified.

All signals in a nonvirtual bus input to a Memory block must have the same sample time, even if the elements of the associated bus object specify inherited sample times. You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus. See "Using Composite Signals" and Bus-Capable Blocks for more information.

Characteristics

Bus-capable	Yes, with restrictions as noted above
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
Multidimensionalized	Yes
Zero Crossing	No

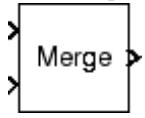
Purpose

Combine multiple signals into 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.

Use Merge blocks only to interleave input signals that update at different times into a combined signal in which the interleaved values retain their separate identities and times. To combine signals that update at the same time into an array or matrix signal, use a Concatenate block.

Merge blocks assume that all driving signals share the same signal memory. The shared signal memory should be accessed only in mutually exclusive fashion. Therefore, always use alternately executing subsystems to drive Merge blocks. See "Creating Alternately Executing Subsystems" for an example.

All signals that connect to a Merge block, or exist anywhere in a network of Merge blocks, are functionally the same signal, and are therefore subject to the restriction that a given signal can have at most one associated signal object. See `Simulink.Signal` and "Multiple Signal Objects" on page 7-148 for more information.

Guidelines for Using the Merge Block

When you use the Merge block, follow these guidelines:

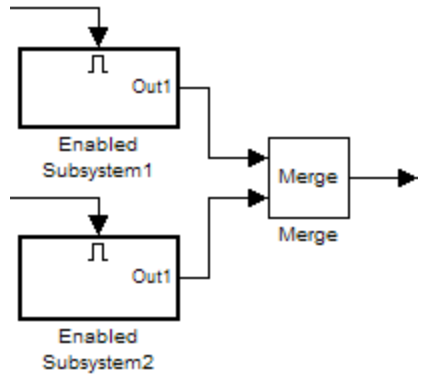
- Always use conditionally-executed subsystems to drive Merge blocks.
- Write your control logic to ensure that at most one of the driving conditionally-executed subsystems executes at any time step.
- Do not connect more than one input of a Merge block to the same conditionally-executed subsystem.
- Do not use a signal that inputs to a Merge block as an input to any other block.

- Always connect a Merge block to at least two input signals.
- Ensure that all input signals have the same sample time.
- Always set the **Initial output** parameter of the Merge block, unless the output port of the Merge block connects to another Merge block.
- For all conditionally-executed subsystem Outport blocks that drive Merge blocks, set the **Output when disabled** parameter to `held`.

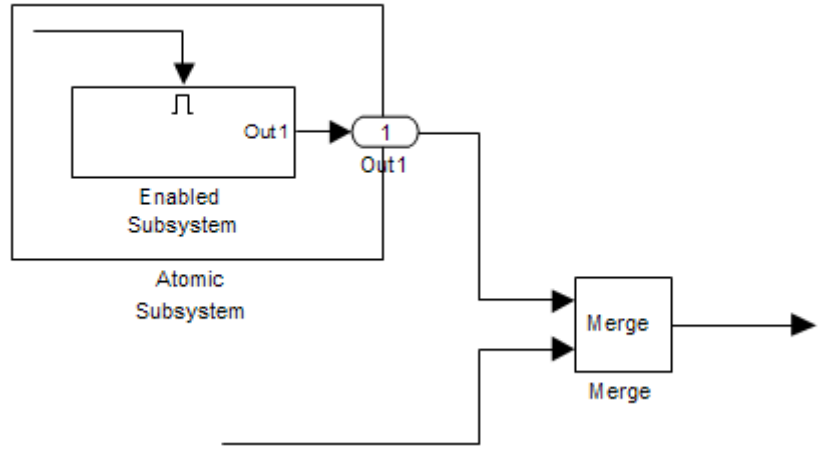
Proper Merge Block Usage

For each input of a Merge block, the topmost non-atomic and non-virtual source must be a conditionally-executed subsystem that is not an Iterator subsystem.

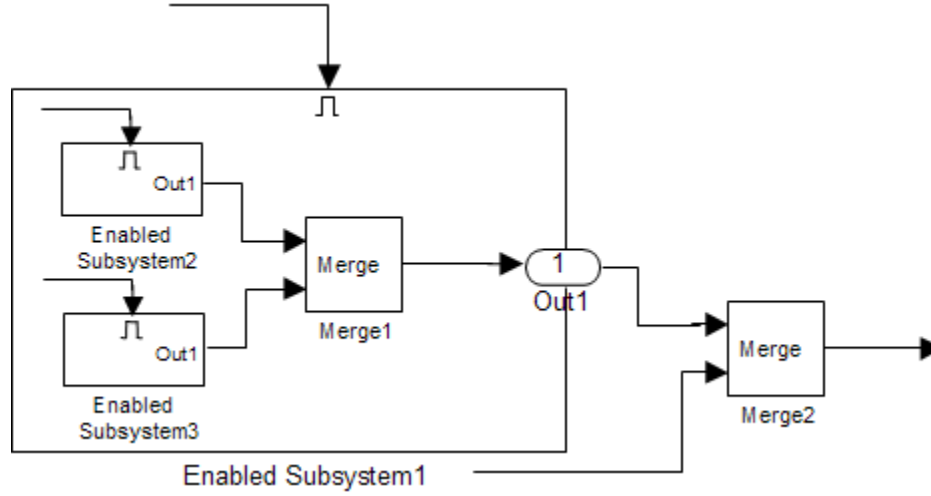
The following example shows proper Merge block usage, merging signals from two conditionally-executed subsystems.



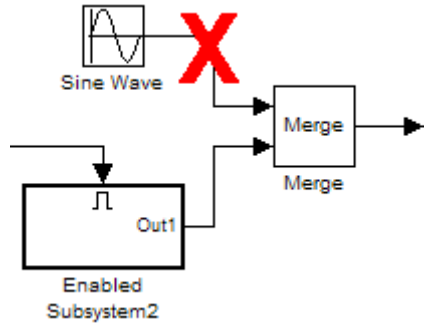
The following figure shows another valid example of Merge block usage, with the topmost non-atomic, non-virtual source being a conditionally executed subsystem.



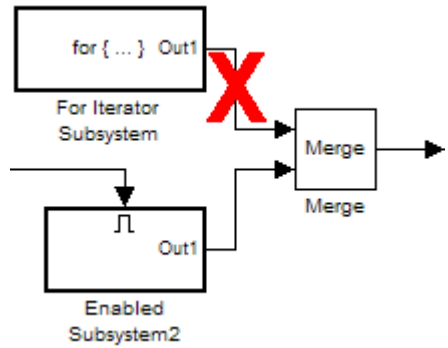
You can also use multiple merge blocks, as shown in the following example.



The following example is *not allowed*, because one input of the Merge block is connected to a Sine Wave block (not a conditionally-executed subsystem).



The following example is also *not allowed*, because one input of the Merge block is connected to an Iterator subsystem.



You can use the Model Advisor to check for proper Merge block usage in your model. For more information, see “Check for proper Merge block usage” on page 10-23.

Initial Output Value

You can specify an initial output value for the Merge block by setting the **Initial output** parameter. If you do not specify an initial output value

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.

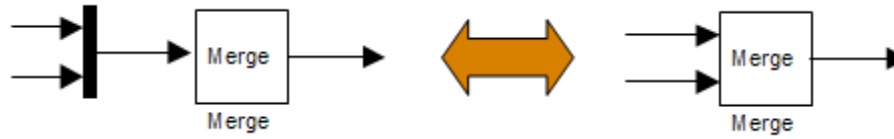
Note If you use simplified initialization mode, you *must* specify the **Initial output** value for all *root* Merge blocks. A root Merge block is any Merge block with an output port that does not connect to another Merge block.

For more information on simplified initialization mode, see "Underspecified initialization detection".

Single-Input Merge

Single-input merge is not supported. Each Merge block must have at least two inputs.

Use Merge blocks only for signals that require merging. If you were previously connecting a Merge block input to a Mux block, use a multi-input Merge block instead.



Input Dimensions and Merge Offsets

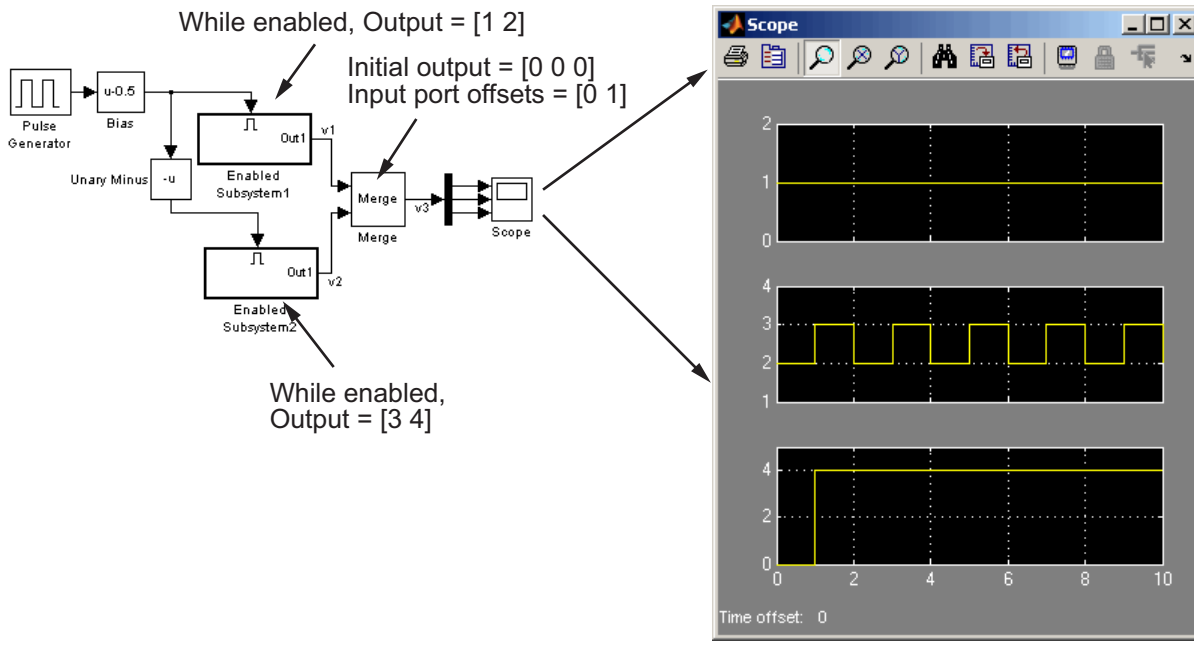
The Merge block accepts only inputs of equal dimensions and outputs a signal of the same dimensions as the inputs, unless you select the **Allow unequal port widths** parameter.

If you select **Allow unequal port widths**, 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 has a Merge block width of

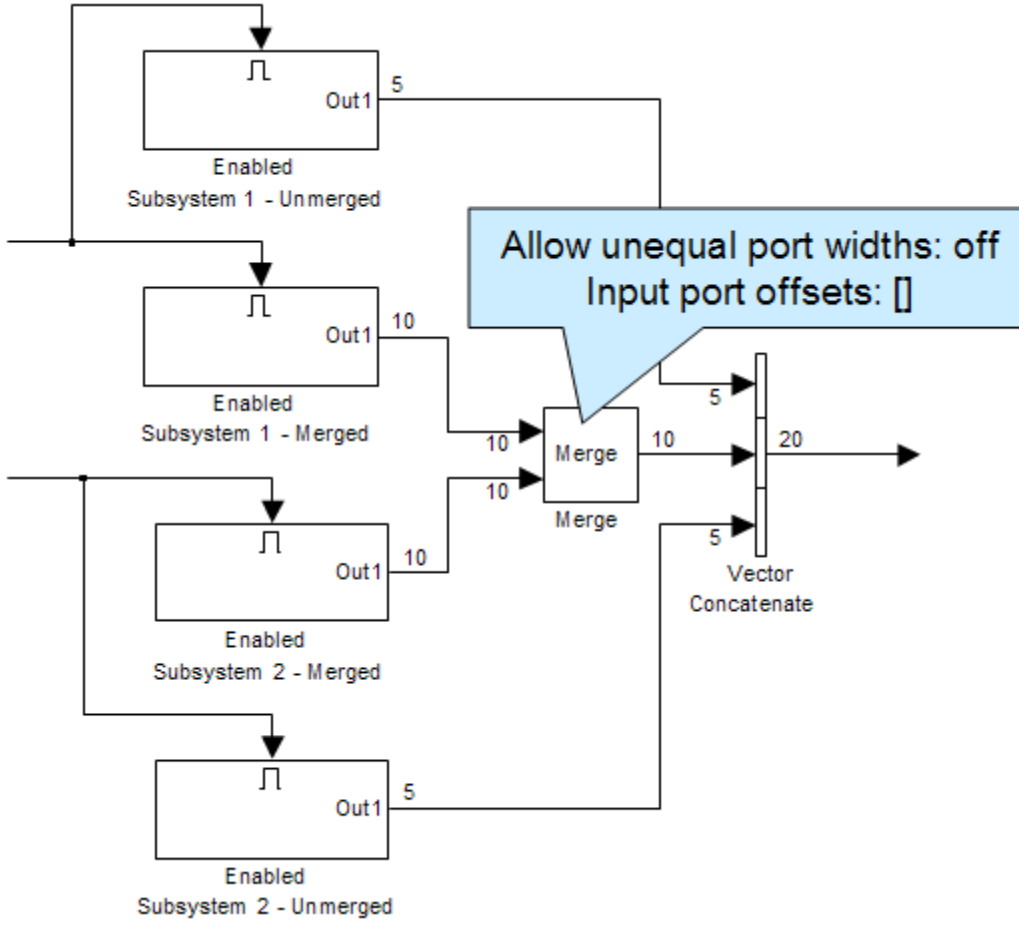
```
max(2+0,2+1)=3
```



In this example, the offset of `v1` is `0` and the offset of `v2` is `1`. The Merge block maps the elements of `v1` to the first two elements of `v3` and the elements of `v2` to the last two elements of `v3`. Only the second element of `v3` is effectively merged, as seen from the scopes output.

If you use Simplified Initialization Mode, you must clear the **Allow unequal port widths** check box. The input port offsets for all input signals must be zero.

Consider using Merge blocks only for signal elements that require true merging. Other elements can be combined with merged elements using the Concatenate block, as shown in the following example.

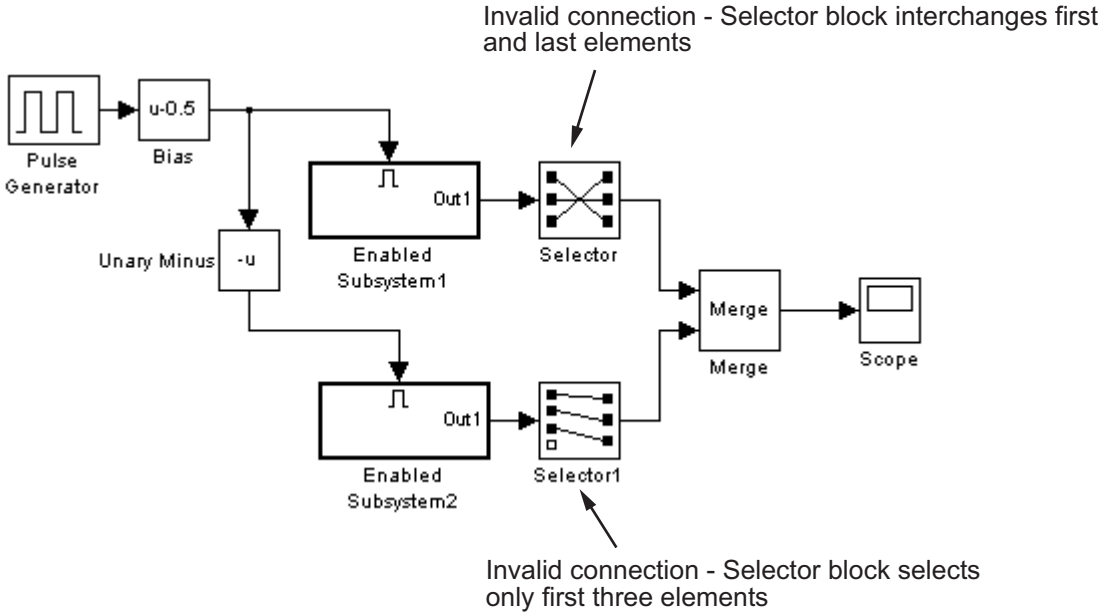


For more information on simplified initialization mode, see "Underspecified initialization detection".

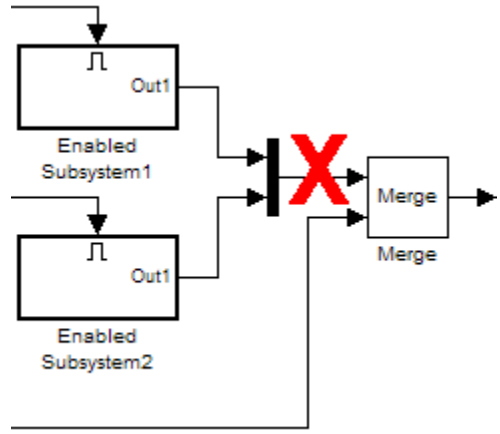
Combining or Reordering of Input Signals

A Merge block does not accept input signals whose elements have been reordered or partially selected. In addition, you should not connect input signals to the Merge block that have been combined outside of a conditionally-executed subsystem.

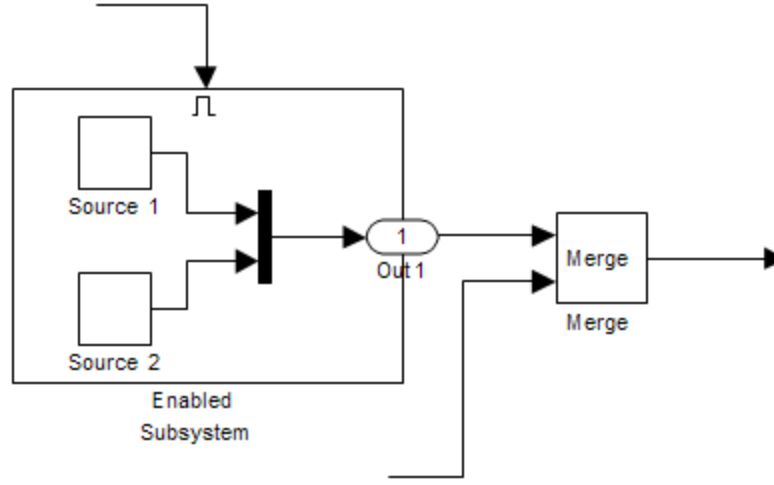
For example, in the following diagram, the Merge block does not accept the output of the first Selector block because the Selector block interchanges the first and last elements of the vector signal. The Merge block does not accept the output of the second Selector block because the Selector block selects only the first three elements.



If you use simplified initialization mode, the following example is *not allowed*, because two signals are being combined outside of a conditionally-executed subsystem.



You can, however, combine or reorder Merge block input signals within a conditionally-executed subsystem. For example, the following diagram is valid.



For more information on simplified initialization mode, see "Underspecified initialization detection".

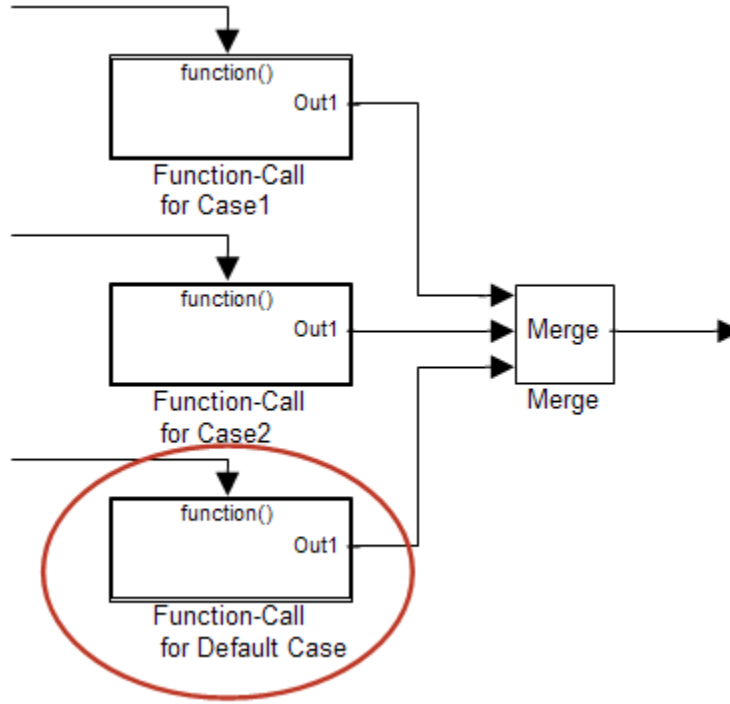
Conditionally-Executed Subsystem Outport Reset

The Outports of conditionally-executed subsystems being merged should not reset when disabled. This action can cause multiple subsystems to update the Merge block at the same time. Specifically, the disabled subsystem updates the Merge block by resetting its output, while the enabled subsystem updates the Merge block by computing its output.

To prevent this behavior, set the Outport block parameter **Output when disabled** to `held` for each conditionally-executed subsystem being merged.

Note If you are using Simplified Initialization Mode, you *must* set the Outport block parameter **Output when disabled** to `held`.

Instead of resetting the subsystem output when it is disabled, add an additional subsystem for the default case, and use control logic to run this subsystem if nothing else runs. For example, see the following:



For more information on simplified initialization mode, see "Underspecified initialization detection".

Merging S-Function Outputs

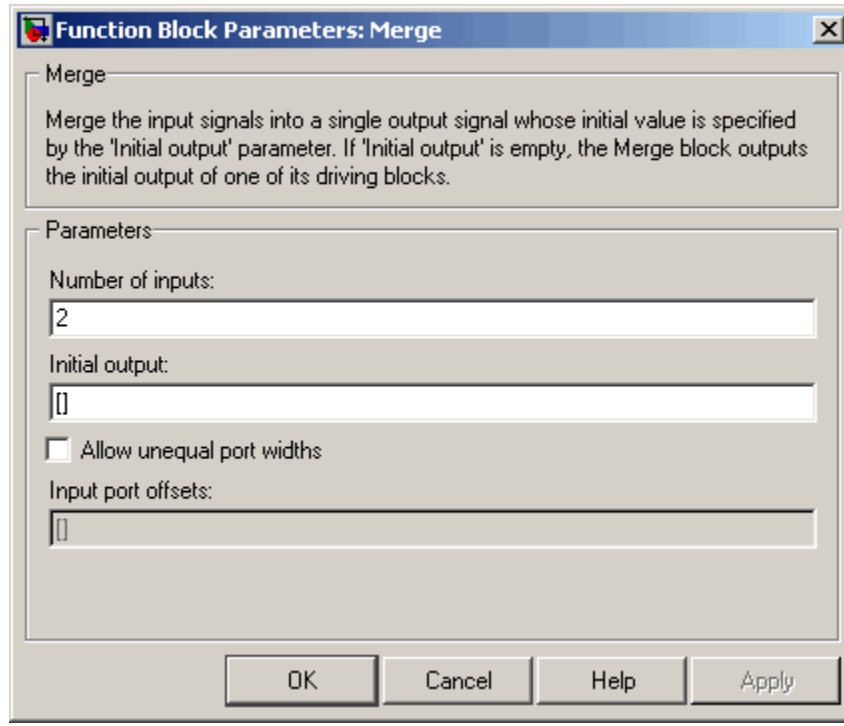
The Merge block can merge a signal from an S-Function block only if the memory used to store the S-Function block's output is reusable. Simulink software displays an error message if you attempt to update or simulate a model that connects a nonreusable port of an S-Function block to a Merge block. See `ssSetOutputPortOptimOpts` for more information.

Data Type Support

The Merge block accepts real or complex signals of any data type supported by Simulink software, including fixed-point and enumerated data types. All inputs must be of the same data type and numeric type.

For a discussion on the data types supported by Simulink software, 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 software 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.

Bus Support

The Merge block is a bus-capable block. The inputs can be virtual or nonvirtual bus signals subject to the following restrictions:

- The number of inputs must be greater than one.
- **Initial output** must be zero or a nonzero scalar.
- **Allow unequal port widths** must be disabled.
- All inputs to the merge must be buses and must be equivalent (same hierarchy with identical names and attributes for all elements).

All signals in a nonvirtual bus input to a Merge block must have the same sample time, even if the elements of the associated bus object specify inherited sample times. You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus. See "Using Composite Signals" and Bus-Capable Blocks for more information.

Characteristics

Bus-capable	Yes, with restrictions as noted above
Direct Feedthrough	Yes
Sample Time	Inherited from the driving block
Scalar Expansion	No
Dimensionalized	Yes
Multidimensionalized	Yes
Zero Crossing	No

MinMax

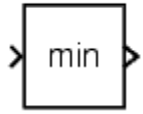
Purpose

Output 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, all nonscalar inputs must 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.

The MinMax block ignores any input value that is `NaN`, except when every input value is `NaN`. When all input values are `NaN`, the output is `NaN`, either as a scalar or the value of each output vector element.

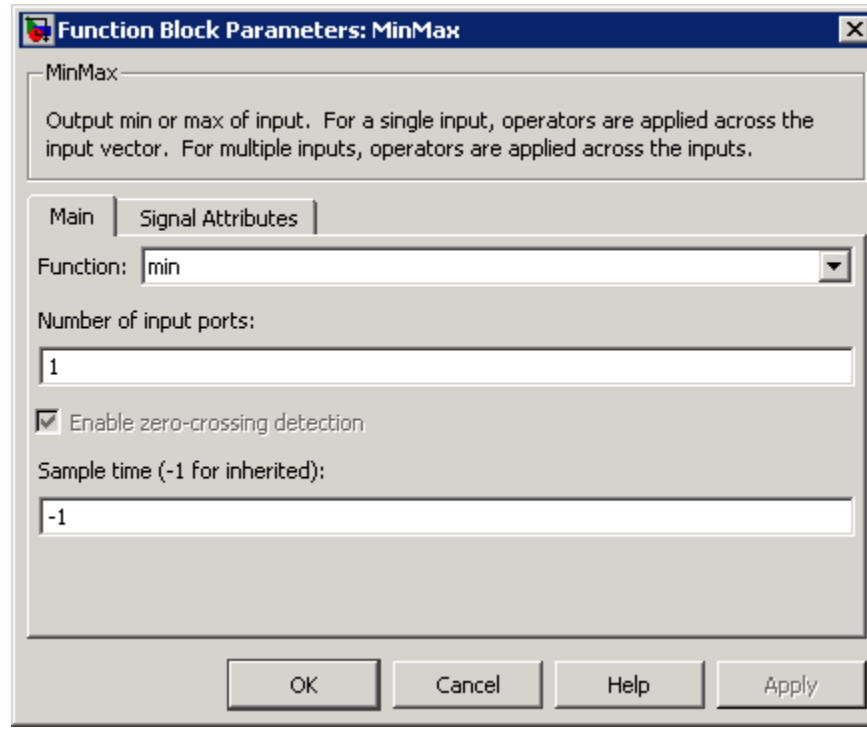
Data Type Support

The MinMax block accepts and outputs real signals of any numeric data type supported by Simulink software, except `Boolean`. The MinMax block supports fixed-point data types.

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

Parameters and Dialog Box

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



Function
Specify whether to apply the function `min` or `max` to the input.

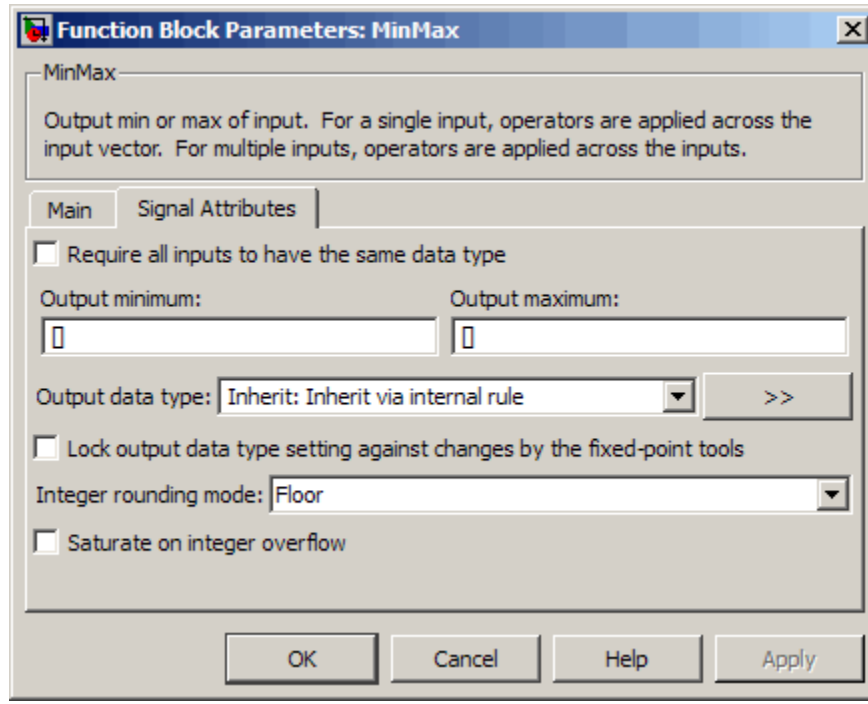
Number of input ports
Specify the number of inputs to the block.

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

Sample time (-1 for inherited)

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

The **Signal Attributes** pane of the MinMax block dialog box appears as follows:



Require all inputs to have the same data type

Select this parameter to require that all inputs must have the same data type.

Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate on integer overflow

Select to have overflows saturate.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of the inputs
Dimensionalized	Yes
Multidimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled

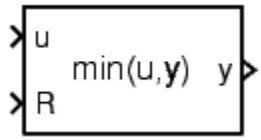
Purpose

Determine minimum or maximum of 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.

The input can be a scalar, vector, or matrix signal. If you specify a scalar **Initial condition** parameter, the block expands the parameter to have the same dimensions as a nonscalar input. The block outputs a signal having the same dimensions as the input. Each output element equals the running minimum or maximum of the corresponding input elements.

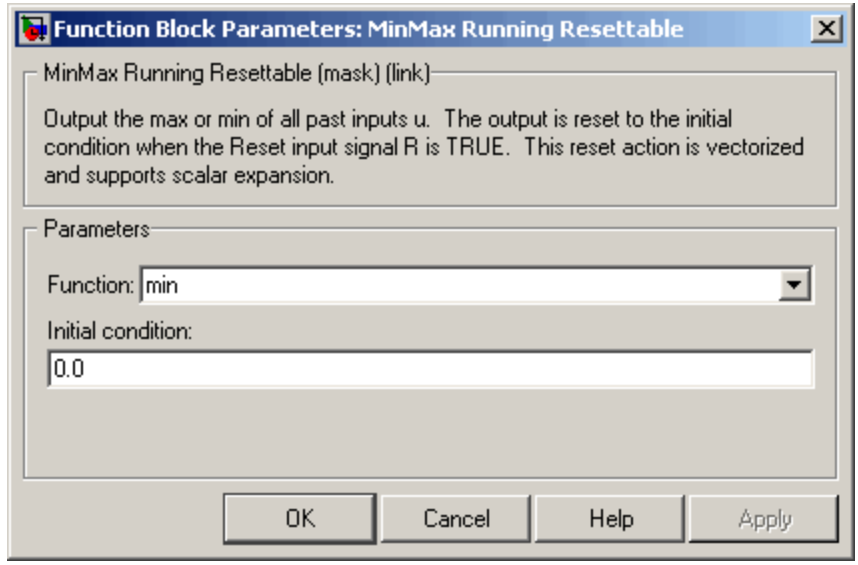
Data Type Support

The MinMax Running Resettable block accepts and outputs real signals of any numeric data type supported by Simulink software, except `Boolean`. The MinMax Running Resettable block supports fixed-point data types.

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

MinMax Running Resettable

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

Purpose

Include model as block in another model

Library

Ports & Subsystems

Description

The Model block allows you to include a model as a block in another model. The included model is called a *referenced model*, and the model containing it (via the Model block) is called the *parent model*.

The Model block displays input ports and output ports corresponding to the referenced model's top-level input and output ports. These ports allow you to connect the referenced model to other blocks in the parent model. See "Referencing a Model" for more information.

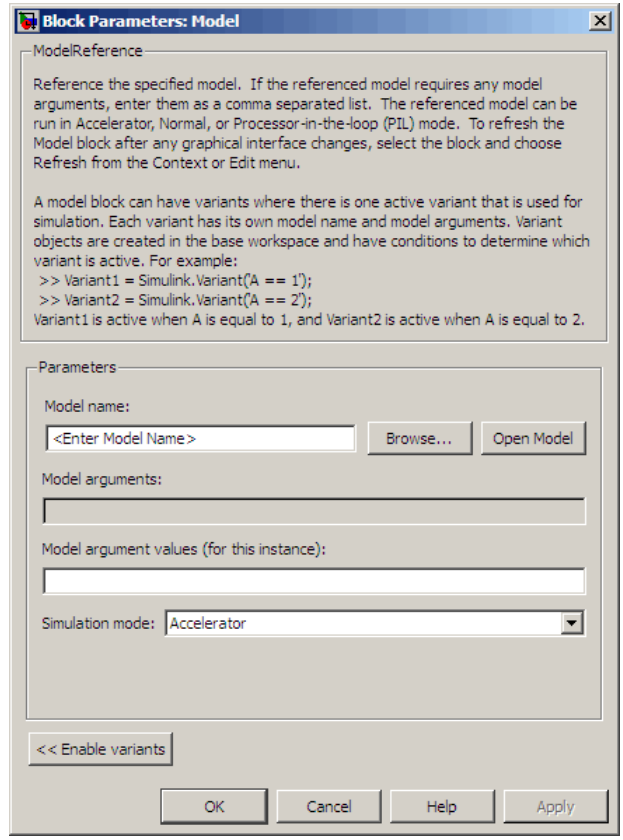
A Model block can specify the referenced model statically as a Model block parameter value, which must name the model literally, or dynamically depending on base workspace values, as described in "Using Model Reference Variants". By default, the contents of a referenced model are user-visible, but you can hide the contents as described in "Protecting Referenced Models".

A signal that connects to a Model block is functionally the same signal outside and inside the block. A given signal can have at most one associated signal object, so the signal connected to the Model block cannot have a signal object in both the parent and the referenced models. See `Simulink.Signal` and "Multiple Signal Objects" on page 7-148 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 extension is optional, and if provided can be `.mdl` or `.mdlp`. See "Creating a Model Reference" and "Protecting Referenced Models" for details.

Model arguments

Names of model arguments accepted by the model referenced by this block. See "Using Model Arguments" for details.

Model argument values

Values to be passed as model 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 **Model arguments** field. See "Using Model Arguments" for details.

Simulation mode

The simulation mode for the model referenced by this block. See "Referenced Model Simulation Modes" for details.

`Accelerator`

Simulink software creates a MEX-file for the submodel, then executes the submodel by running the S-function. See "Accelerator Mode".

`Normal`

Simulink software executes the submodel interpretively, as if the submodel were an atomic subsystem implemented directly within the parent model. See "Normal Mode" and "Systems and Subsystems".

`Processor-in-the-loop (PIL)`

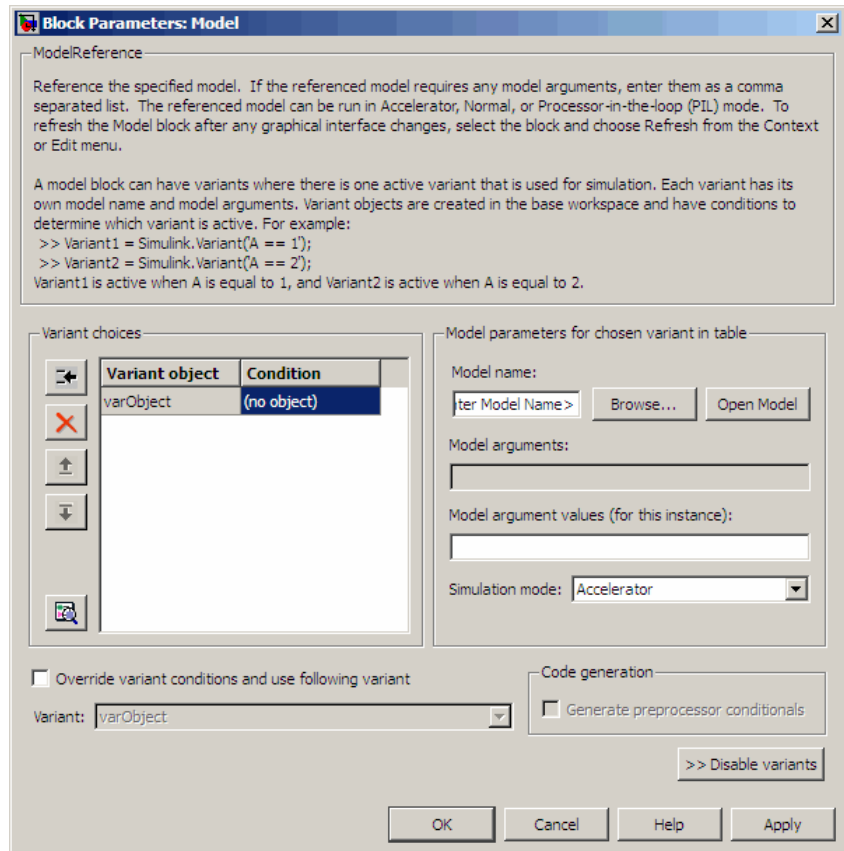
This feature requires Real-Time Workshop Embedded Coder software. Simulink creates both a MEX-file and a model reference target for the submodel. The MEX-file executes the cross-compiled object code on the target processor (or equivalent instruction set simulator). See "Processor-in-the-loop (PIL) mode" and "Verifying Compiled Object Code with Processor-in-the-Loop Simulation".

Enable Variants

Enables model reference variants and opens the Model Variants Section, which is hidden by default. See "Model Variants Section" on page 2-760 and "Enabling Model Variants".

Model Variants Section

When you click **Enable Variants**, the Model block dialog box expands to show the Model variants Section. The dialog then looks like this:



The standard Model block capabilities remain available on the right, and have the same capabilities, although some labels have changed slightly to accommodate model variants. See “Using Model Reference Variants” for information about using the Model Reference Variants section. The parameters specific to model reference variants are:

Disable Variants
Disables model reference variants and hides the Model Variants Section. The block retains any information you have entered and approved by clicking **Apply** or **OK**. See "Disabling Model Variants".

Variant objects
A list (shown as a table) of `Simulink.Variant` objects whose Boolean expressions determine which is the active variant. See "Example Variant Objects" and "Implementing Variant Objects".

Override variant conditions and use the following variant
Whether to override the variant conditions and make a specified variant the active variant. See "Overriding Variant Conditions".

Variant
The name of the variant to use if **Override variant conditions and use the following variant** is selected. See "Overriding Variant Conditions".

Generate preprocessor conditionals
Controls whether generated code contains preprocessor conditionals. This checkbox is relevant only to code generation, and has no effect on the behavior of a model in Simulink. The checkbox is available only for ERT targets when **Inline parameters** is `'on'`. See "Generating Code Variants for Variant Models" for more information.

Navigating a Model Block

Model blocks behave differently from other blocks when double-clicked. This customized behavior provides the results most likely to be useful given the current status of the Model block, as follows:

- Double-clicking the prototype Model block in the Ports & Subsystems library opens its Block Parameters dialog box for inspection, but does not allow you to specify parameter values.
- Double-clicking an unresolved Model block opens its Block Parameters dialog box. You can then resolve the block by specifying a **Model name**.

- Double-clicking a resolved Model block opens the model that the block references. You can also open the model by choosing **Open Model** from the **Context** or **Edit** menu.

To display the Block Parameters dialog box for a resolved Model block, choose **Model Reference Parameters** from the **Context** or **Edit** menu.

Model Blocks and Direct Feedthrough

When a Model block is part of a cycle, and the block is a direct feedthrough block, an algebraic loop can result. An algebraic loop in a model is not necessarily an error, but it may not give the expected results. See:

- "Algebraic Loops" for information about direct feedthrough and algebraic loops.
- "Highlighting Algebraic Loops" for information about seeing algebraic loops graphically
- "Displaying Algebraic Loop Information" for information about tracing algebraic loops in the debugger.
- The "Diagnostics Pane: Solver" pane "Algebraic loop" option for information about detecting algebraic loops automatically.

Direct Model Block Feedthrough Caused by Submodel Structure

A Model block may be a direct feedthrough block due to the structure of the referenced model. Where direct feedthrough results from submodel structure, and causes an unwanted algebraic loop, you can:

- Automatically eliminate the algebraic loop using techniques described in:
  - "Minimize algebraic loop"
  - "Minimize algebraic loop occurrences"
  - "Eliminating Algebraic Loops"

- Manually insert one or more Unit Delay blocks as needed to break the algebraic loop.

Direct Model Block Feedthrough Caused by Model Configuration

ERT-based targets provide the option **Configuration Parameters > Real-Time Workshop Pane > Interface > Single output/update function**. This option controls whether generated code has separate output and update functions, or a combined output/update function. See:

- "Embedded Model Functions" for information about separate and combined output and update functions.
- "Single output/update function" for information about specifying whether code has separate or combined functions.

When **Single output/update function** is enabled (the default) a Model block has a combined output/update function, which makes the block a direct feedthrough block for all inports regardless of the structure of the referenced model. Where an unwanted algebraic loop results, you can:

- Disable **Single output/update function**. The code for the Model block then has separate output and update functions, eliminating the direct feedthrough and hence the algebraic loop.
- Automatically eliminate the algebraic loop using techniques described in:
  - "Minimize algebraic loop"
  - "Minimize algebraic loop occurrences"
  - "Eliminating Algebraic Loops"
- Manually insert one or more Unit Delay blocks as needed to break the algebraic loop.

Characteristics

Direct Feedthrough	If “Single output/update function” is enabled (the default), a Model block is a direct feedthrough block regardless of the structure of the referenced model. If “Single output/update function” is disabled, a Model block may or may not be a direct feedthrough block, depending on the structure of the referenced model.
Scalar Expansion	Depends on model referenced by this block.
Multidimensionalized	Yes

See Also

- “Referencing a Model”

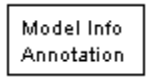
Purpose

Display revision control information in model

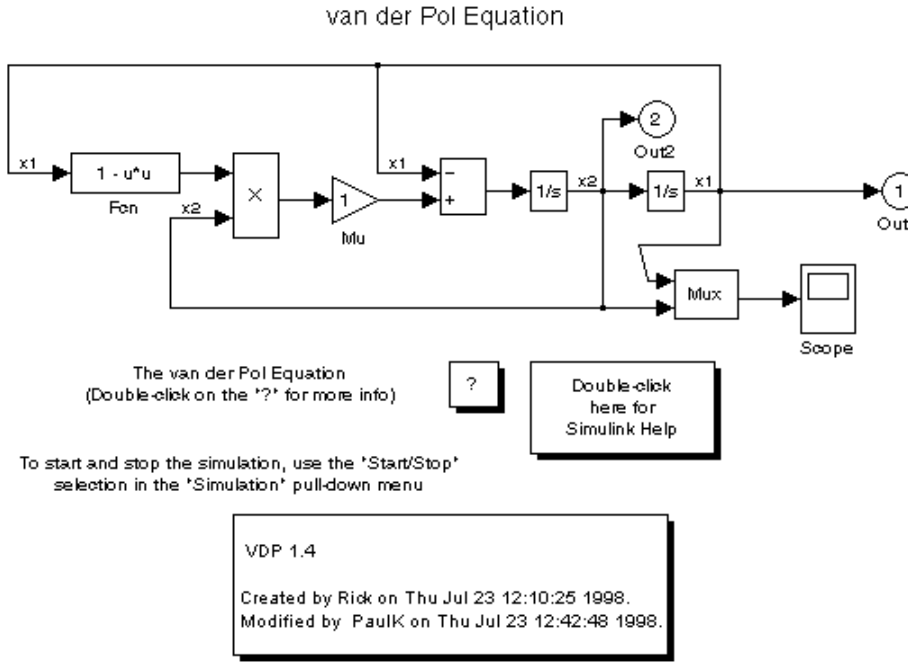
Library

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.

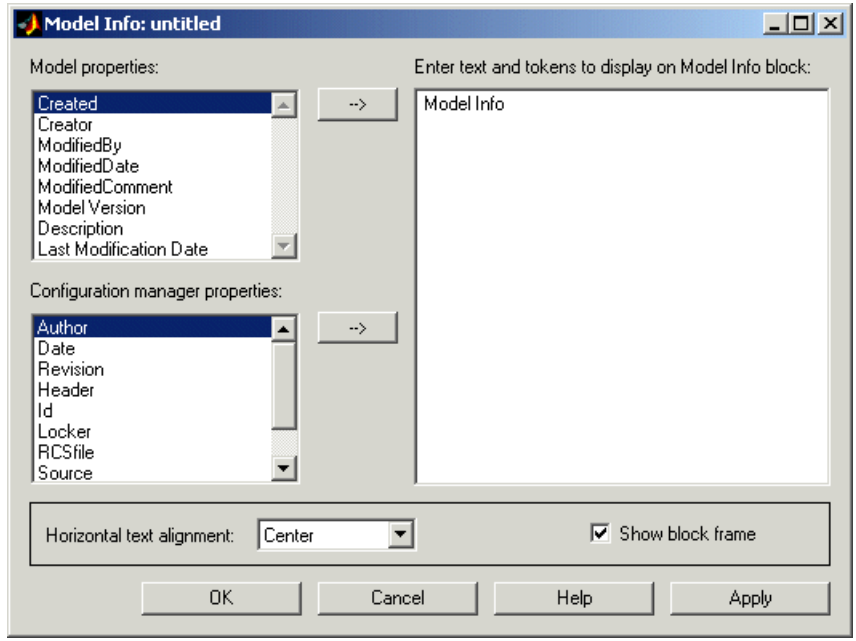


A Model Info block can show revision control information embedded in the model itself and/or information maintained by an external revision control or configuration management system. A Model Info block's dialog allows you to specify the content and format of the text displayed by the block.

Data Type Support

Not applicable.

Parameters and Dialog Box



The Model Info block dialog box includes the following fields:

Editable text

Enter the text to be displayed by the Model Info block in this field. You can freely embed variables of the form `%<propname>`, where `propname` is the name of a model or revision control system property, in the entered text. The value of the property replaces the variable in the displayed text. For example, suppose that the current version of the model is 1.1. Then the entered text

```
Version %<ModelVersion>
```

appears as

```
Version 1.1
```

in the displayed text. The model and revision control system properties that you can reference in this way are listed in the **Model properties** and **Configuration manager properties** fields.

Model properties

Lists revision control properties stored in the model. Selecting a property and then selecting the adjacent arrow button enters the corresponding variable in the **Editable text** field. For example, selecting `CreatedBy` enters `%<CreatedBy%>` in the **Editable text** field. See "Version Control Properties" for a description of the usage of the properties specified in this field.

Configuration manager properties

This field appears only if you previously specified an external configuration manager for this model on the MATLAB **Preferences** dialog box for the model (see "Specifying the Source Control System on UNIX Platforms" in the online MATLAB documentation) or by setting the model's `ConfigurationManager` property. The field lists version control information maintained by the external system that you can include in the Model Info block. To include an item from the list, select it and then click the adjacent arrow button.

Note The selected item does not appear in the Model Info block until you check the model in or out of the repository maintained by the configuration manager and you have closed and reopened the model.

Multiport Switch

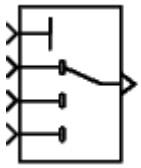
Purpose

Choose between multiple block inputs

Library

Signal Routing

Description



Types of Block Inputs

The Multiport Switch block chooses among several inputs. The first input is the *control input*, while the rest of the inputs are *data inputs*.

The value of the control input determines which data input passes through to the output port. See "How to Rotate a Block" in the Simulink documentation for a description of the port order for various block orientations.

Rules That Determine the Block Output

You specify the number of data inputs with the **Number of inputs** parameter. These data inputs can be scalar or vector. The following rules determine the block output:

- If you specify only one data input and that input is a vector, the block behaves as an "index selector," and not as a multiport switch. The block output is the vector element that corresponds to the value of the control input.
- If you specify more than one data input, the block behaves like a multiport switch. The block output is the data input that corresponds to the value of the control input. If at least one of the data inputs is a vector, the block output is a vector. The block expands any scalar inputs to vectors.
- If the data inputs are scalar, the output is a scalar.

How the Block Interprets the Control Input

The following table summarizes how the block interprets the control input and passes data to the output.

Control Input	Truncation	Setting for Zero-Based Indexing	Block Behavior During Simulation
Integer value	None	On	The specified data input passes to the output, based on zero-based indexing. If the control input is less than `0` or greater than the number of data inputs minus one, an out-of-range error appears.
		Off	The specified data input passes to the output, based on one-based indexing. If the control input is less than `1` or greater than the number of data inputs, an out-of-range error appears.
Not an integer value	The block truncates the value to an integer by rounding to floor.	On	The specified data input passes to the output, based on zero-based indexing. If the truncated control input is less than `0` or greater than the number of data inputs minus one, an out-of-range error appears.
		Off	The specified data input passes to the output, based on one-based indexing. If the truncated control input is less than `1` or greater than the number of data inputs, an out-of-range error appears.

How the Block Handles an Out-of-Range Control Input

The following table summarizes how the block defines the default output for an out-of-range control input.

Block Usage	How the Block Defines the Default Output
Simulation	None, because an out-of-range control input results in an error.
Real-Time Workshop generated code	Undefined
Accelerator modes	Undefined

Tip Ensure that you use an in-range control input for the Multiport Switch block. Avoid relying on undefined behavior for Real-Time Workshop generated code or Accelerator modes.

Related Blocks

The Index Vector block, also in the Signal Routing library, is another implementation of the Multiport Switch block that has different default parameter settings.

Data Type Support

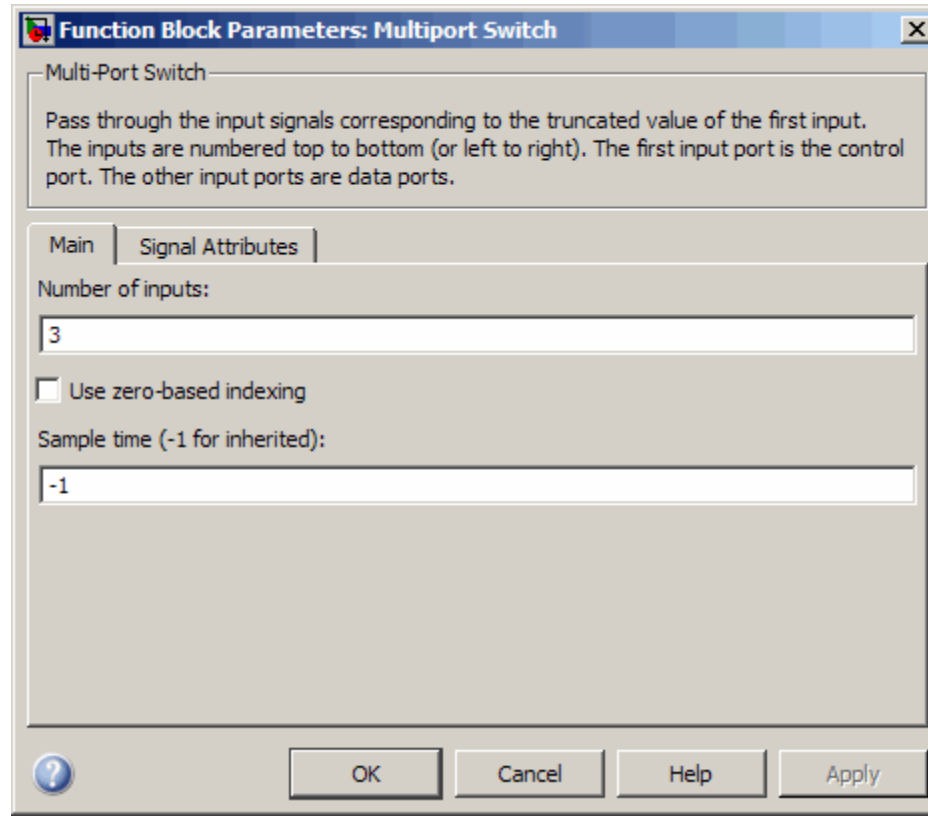
The control signal can be of any data type that Simulink supports, including fixed-point and enumerated types, except `Boolean`. If the control signal is numeric, it cannot be complex. If the control signal is an enumerated signal, the block uses the value of the underlying integer to select a data port. If the underlying integer does not correspond to a data port, an error occurs.

The data signals can be of any data type that Simulink supports. If any data signal is of an enumerated type, all others must be of the same enumerated type.

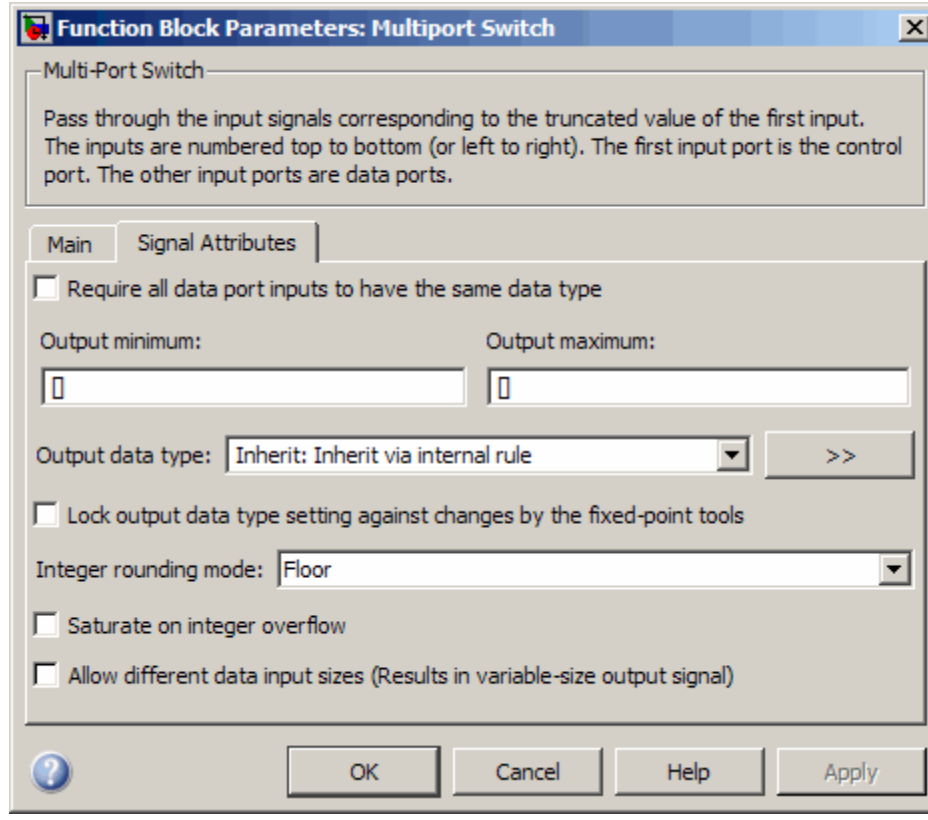
For more informaiton, see “Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Multiport Switch block dialog box appears as follows:



The **Signal Attributes** pane of the Multiport Switch block dialog box appears as follows:



- "Number of inputs" on page 2-774
- "Use zero-based indexing" on page 2-775
- "Sample time (-1 for inherited)" on page 2-776
- "Require all data port inputs to have the same data type" on page 2-777
- "Lock output data type setting against changes by the fixed-point tools" on page 2-1304
- "Integer rounding mode" on page 2-779

Number of inputs

Default: 1

Specify the number of data inputs to the block.

Use zero-based indexing

Specify block indexing.

Settings

Default: Off

 On

Uses zero-based indexing.

 Off

Uses one-based indexing.

Sample time (-1 for inherited)

Specify the time interval between samples.

Settings

Default: `-1`

This setting indicates the block inherits the sample time.

Require all data port inputs to have the same data type

Specify allowed data types.

Settings

Default: Off

 On

Requires all data port inputs to have the same data type.

 Off

Allows data port inputs to have different data types.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Floor`

`Ceiling`
: Rounds toward positive infinity and is equivalent to the MATLAB `ceil` function.

`Convergent`
: Rounds toward the nearest representable number, with ties rounding to the nearest even integer. Convergent rounding is equivalent to the Fixed-Point Toolbox `convergent` function.

`Floor`
: Rounds toward negative infinity and is equivalent to the MATLAB `floor` function.

`Nearest`
: Rounds toward the nearest representable number, with the exact midpoint rounded toward positive infinity. Rounding toward nearest is equivalent to the Fixed-Point Toolbox `nearest` function.

`Round`
: Rounds to the nearest representable number. Ties for positive numbers round in the direction of positive infinity and ties for negative numbers round in the direction of negative infinity. This mode is equivalent to the Fixed-Point Toolbox `round` function.

`Simplest`
: Automatically chooses between round toward floor and round toward zero to produce generated code that is as efficient as possible

`Zero`
: Rounds toward zero and is equivalent to the MATLAB `fix` function.

Saturate on integer overflow

Specify what to do with overflows.

Settings

Default: Off

 On

Allows overflows to saturate. Saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when an overflow is not possible. In this case, the code generator does not produce saturation code.

 Off

Does not allow overflow to saturate.

Allow different data input sizes

Select this check box to allow input signals with different sizes.

Settings

Default: Off

 On

Allows input signals with different sizes, and propagate the input signal size to the output signal.

 Off

Requires that input signals be the same size.

Command-Line Information

Parameter: `AllowDiffInputSignals`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

Output minimum

Specify the minimum value that the block outputs.

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value the block outputs.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `Inherit: Inherit via internal rule`

`Inherit: Inherit via internal rule`
: This option appears only for blocks (Discrete-Time Integrator, Gain, Product, Sum, Switch blocks). Simulink chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the calculated 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 you set the **Device type** parameter on the **Hardware Implementation** configuration parameters pane 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 you specify `ASIC/FPGA` as the targeted hardware type, the output data type is `sfix24`. If you select `Unspecified (assume 32-bit Generic)` for a generic 32-bit microprocessor as the target hardware, the output data type is `int32`. If the word lengths provided by the target microprocessor cannot accommodate the output range, Simulink displays an error message in the Simulation Diagnostics Viewer.

`Inherit: Inherit from 'Constant value'`
: Use data type of the **Constant value** parameter. This option appears only for the Constant block.

`Inherit: Inherit via back propagation`
: Uses the data type of the driving block (Data Type Conversion block).

`Inherit: Same as input`
: Uses the data type of sole input signal. This option appears only for the Saturation block.

`Inherit: Same as first input`
: Uses the data type of the first input signal. This option appears only for some blocks.

`Inherit: Same as accumulator`
: Uses the accumulator data type for the output data type. This option appears only for some blocks.

`double`
: Specifies output data type `double`.

`single`
: Specifies output data type `single`.

`int8`
: Specifies output data type `int8`.

`uint8`
: Specifies output data type `uint8`.

`int16`
: Specifies output data type `int16`.

`uint16`
: Specifies output data type `uint16`.

`int32`
: Specifies output data type `int32`.

`uint32`
: Specifies outputs data type `unt32`.

`fixdt(1,16,0)`
: Specifies output data type fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Specifies output data type fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
: Uses an enumerated data type, for example, `Enum: BasicColors`. This option appears only for some blocks.

`<data type expression>`
: Uses a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Mode

Select the category of data to specify.

Settings
Default:

- `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch)
- `Built in` (Logical Operator, Relational Operator)

`Inherit`
: Specifies inheritance rules for data types. Selecting `Inherit` enables a list of possible values, which can vary by block:

  - `Inherit from 'Constant value'` (Constant block default)
  - `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
  - `Inherit via back propogation` (Data Type Conversion block default)
  - `auto` (Inport, Outport block default)
  - `Logical (see Configuration Parameters: Optimization)`
  - `Same as first input`
  - `Same as input` (Saturation block default)
  - `Same as accumulator`

`Built in`
: Specifies built-in data types. Selecting `Built in` enables list of possible values, which can vary by block:

  - `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
  - `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. Following is a list of possible values, which can vary by block:

  - `<class name>`

`Expression`
: Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. Following is a list of possible values, which can vary by block:

  - `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specifies the fixed-point data as signed.

`Unsigned`
: Specifies the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings
Default:

- `Best precision` (Constant),
- `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch)
- `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specifies binary point location.

`Slope and bias`
: Enters slope and bias.

`Best precision`
: Specifies best-precision values. This option appears for some blocks.

`Integer`
: Specifies integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**

- **Bias**
- **Calculate Best-Precision Scaling**

See Also

See “Specifying a Fixed-Point Data Type” in the *Simulink User’s Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bus Support

The Multiport Switch block is a bus-capable block. The data inputs can be virtual or nonvirtual bus signals subject to the following restrictions:

- All the buses must be equivalent (same hierarchy with identical names and attributes for all elements).
- All signals in a nonvirtual bus input to a Multiport Switch block must have the same sample time. This requirement holds even if the elements of the associated bus object specify inherited sample times.

You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus. See "Using Composite Signals" and"Bus-Capable Blocks"for more information.

Characteristics

Bus-capable	Yes, with restrictions
Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter

Scalar Expansion	Yes
Dimensionalized	Yes
Multidimensionalized	Yes
Zero Crossing	No

Mux

Purpose

Combine several input signals into vector

Library

Signal Routing

Description

The Mux block combines its inputs into a single vector output. An input can be a scalar or vector signal. All inputs must be of the same data type and numeric type. The elements of the vector output signal take their order from the top to bottom, or left to right, input port signals. See "How to Rotate a Block" for a description of the port order for various block orientations. To avoid adding clutter to a model, Simulink software hides the name of a Demux block when you copy it from the Simulink library to a model. See "Mux Signals" for information about creating and decomposing vectors.

Note The Mux block allows you to connect signals of differing data and numeric types and matrix signals to its inputs. In this case, the Mux block acts like a Bus Creator block and outputs a bus signal rather than a vector. The MathWorks discourages using mux blocks to create bus signals, and may not support this practice in future releases. See "Avoiding Mux/Bus Mixtures" for more information.

The Mux block's **Number of Inputs** parameter allows you to specify input signal names and sizes as well as the number of inputs. You can use any of the following formats to specify this parameter:

- Scalar

  Specifies the number of inputs to the Mux block. When this format is used, the block accepts scalar or vector signals of any size. Simulink software assigns each input the name `signalN`, where `N` is the input port number.

- Vector

  The length of the vector specifies the number of inputs. Each element specifies the size of the corresponding input. A positive value specifies

that the corresponding port can accept only vectors of that size. For example, `[2 3]` specifies two input ports of sizes `2` and `3`, respectively. If an input signal width does not match the expected width, Simulink software displays an error message. A value of `-1` specifies that the corresponding port can accept scalars or vectors of any size.

- Cell array

  The length of the cell array specifies the number of inputs. The value of each cell specifies the size of the corresponding input. A scalar value `N` specifies a vector of size `N`. A value of `-1` means that the corresponding port can accept scalar or vector signals of any size.

- Signal name list

  You can enter a list of signal names separated by commas. Simulink software assigns each name to the corresponding port and signal. For example, if you enter `position,velocity`, the Mux block will have two inputs, named `position` and `velocity`.

The MathWorks encourages using Vector Concatenate blocks rather than Mux blocks to combine vectors. The primary exception is the creation of a vector of function calls, which requires a Mux block. In future releases, Mux blocks may have no unique capabilities and may be deprecated.

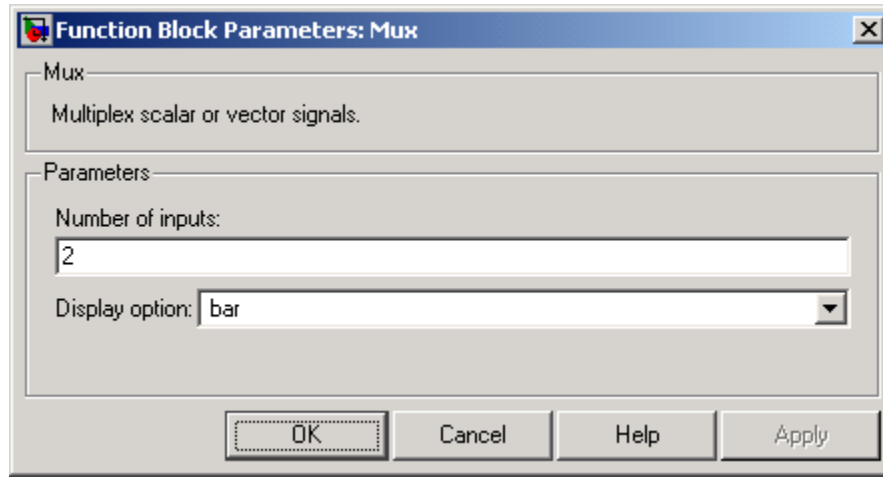
If you want to create a composite signal, in which the constituent signals retain their identities and can have different data types, use a Bus Creator block rather than a Mux block. Although you can use a Mux block to create a composite signal, The MathWorks discourages this practice. See “Avoiding Mux/Bus Mixtures” for more information.

Data Type Support

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

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

Parameters and Dialog Box



Number of inputs

Specify number and size of inputs.

Settings

Default: 2

You can enter a comma-separated list of signal names for this parameter field.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Display option

Specify the appearance of the block in the model.

Settings

Default: `bar`

`bar`
: Displays the block in a solid foreground color

`none`
: `Mux` appears inside the block

`signals`
: Displays signal names next to each port

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Virtual	Yes For more information, see "Virtual Blocks" in the Simulink documentation.

See Also

Demux

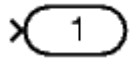
Purpose

Create output port for subsystem or external output

Library

Ports & Subsystems, Sinks

Description



Outport blocks are the links from a system to a destination outside the system.

Simulink software assigns Outport block port numbers according to these rules:

- It automatically numbers the Outport blocks within a top-level system or subsystem sequentially, starting with 1.
- If you add an Outport block, it is assigned the next available number.
- If you delete an Outport block, other port numbers are automatically renumbered to ensure that the Outport blocks are in sequence and that no numbers are omitted.

Outport Blocks in a Subsystem

Outport blocks in a subsystem represent outputs from the subsystem. A signal arriving at an Outport block in a subsystem flows out of the associated output port on that Subsystem block. The Outport block associated with an output port on a Subsystem block is the block whose **Port number** parameter matches the relative position of the output port on the Subsystem block. For example, the Outport block whose **Port number** parameter is 1 sends its signal to the block connected to the topmost output port on the Subsystem block.

If you renumber the **Port number** of an Outport block, the block becomes connected to a different output port, although the block continues to send the signal to the same block outside the subsystem.

When you create a subsystem by selecting existing blocks, if more than one Outport block is included in the grouped blocks, Simulink software automatically renumbers the ports on the blocks.

The Outport block name appears in the Subsystem icon as a port label. To suppress display of the label, select the Outport block and choose **Hide Name** from the **Format** menu.

Outport Blocks in Conditionally Executed Subsystems

When an Outport block is in an enabled subsystem, its initial output value can be either explicitly specified, or inherited from its input signal.

Specifying Initial Conditions

To explicitly specify an initial output value:

1. Select `Dialog` in the **Source of initial output value** drop-down list.
2. Specify the **Initial output** parameter.

If you select `Dialog`, you can also specify what happens to the output when the subsystem is disabled, using the **Output when disabled** drop-down menu. Select either:

- `reset` – The output value is reset to the **Initial output** value when the subsystem is disabled.
- `held` – The output value remains at its most recent value when the subsystem is disabled.

Note If you are connecting the output of the conditionally executed subsystem to a Merge block, set **Output when disabled** to `held` to ensure consistent simulation results.

If you are using simplified initialization mode, you must select `held` when connecting a conditionally executed subsystem to a Merge block. For more information, see "Underspecified initialization detection".

Inheriting Initial Conditions

The initial output value of the outport block can be inherited from the following sources:

- Output port of another conditionally executed subsystem
- Merge block (with Initial output specified)

- Function-Call Model Reference block
- Constant block (simplified initialization mode only)
- IC block (simplified initialization mode only)

The procedure you use to inherit the initial conditions of the Outport block differs depending on whether you are using classic initialization mode or simplified initialization mode.

To inherit initial conditions in classic initialization mode:

1. Select `Dialog` in the **Source of initial output value** drop-down list.
2. Set the **Initial output** parameter to `[]` (empty matrix).
3. Click **OK**.

Note For all other driving blocks, specify an explicit initial output value.

To inherit initial conditions in simplified initialization mode:

1. Select `Input signal` in the **Source of initial output value** drop-down list.
2. Click **OK**.

   The **Initial output** and **Output when disabled** parameters are disabled, and the values are both inherited from the input signal.

For more information on classic and simplified initialization mode, see "Underspecified initialization detection".

Outport Blocks in a Top-Level System

Outport blocks in a top-level system have two uses: to supply external outputs to the workspace, which you can do by using either the

Configuration Parameters dialog box or the `sim` command, and to provide a means for analysis functions to obtain output from the system.

- To supply external outputs to the workspace, use the **Configuration Parameters** dialog box (see Exporting Output Data to the MATLAB Workspace) or the `sim` command (see `sim`). For example, if a system has more than one Outport block and the save format is array, the following command

  ```
  [t,x,y] = sim(...);
  ```

  writes `y` as a matrix, with each column containing data for a different Outport block. The column order matches the order of the port numbers for the Outport blocks.

  If you specify more than one variable name after the second (state) argument, data from each Outport block is written to a different variable. For example, if the system has two Outport blocks, to save data from Outport block 1 to `speed` and the data from Outport block 2 to `dist`, you could specify this command:

  ```
  [t,x,speed,dist] = sim(...);
  ```

- To provide a means for the `linmod` and `trim` analysis functions to obtain output from the system (see “Linearizing Models”)

Connecting Buses to Root Level Outports

A root level Outport of a model can accept a virtual bus only if all elements of the bus have the same data type. The Outport block automatically unifies the bus to a vector having the same number of elements as the bus, and outputs that vector.

If you want a root level Outport of a model to accept a bus signal that contains mixed types, you must select the Outport's **Specify properties via bus object** parameter and set the Outport's **Bus object for validating input bus** parameter to the name of a bus object that defines the type of bus that the Outport produces. If the bus signal is virtual, it will be converted to nonvirtual, as described in “Automatic Bus Conversion”. See “Using Bus Objects” more information.

An Outport in a conditionally executed subsystem that is connected to a bus that contains mixed data types cannot be configured to reset and have initial values specified.

Data Type Support

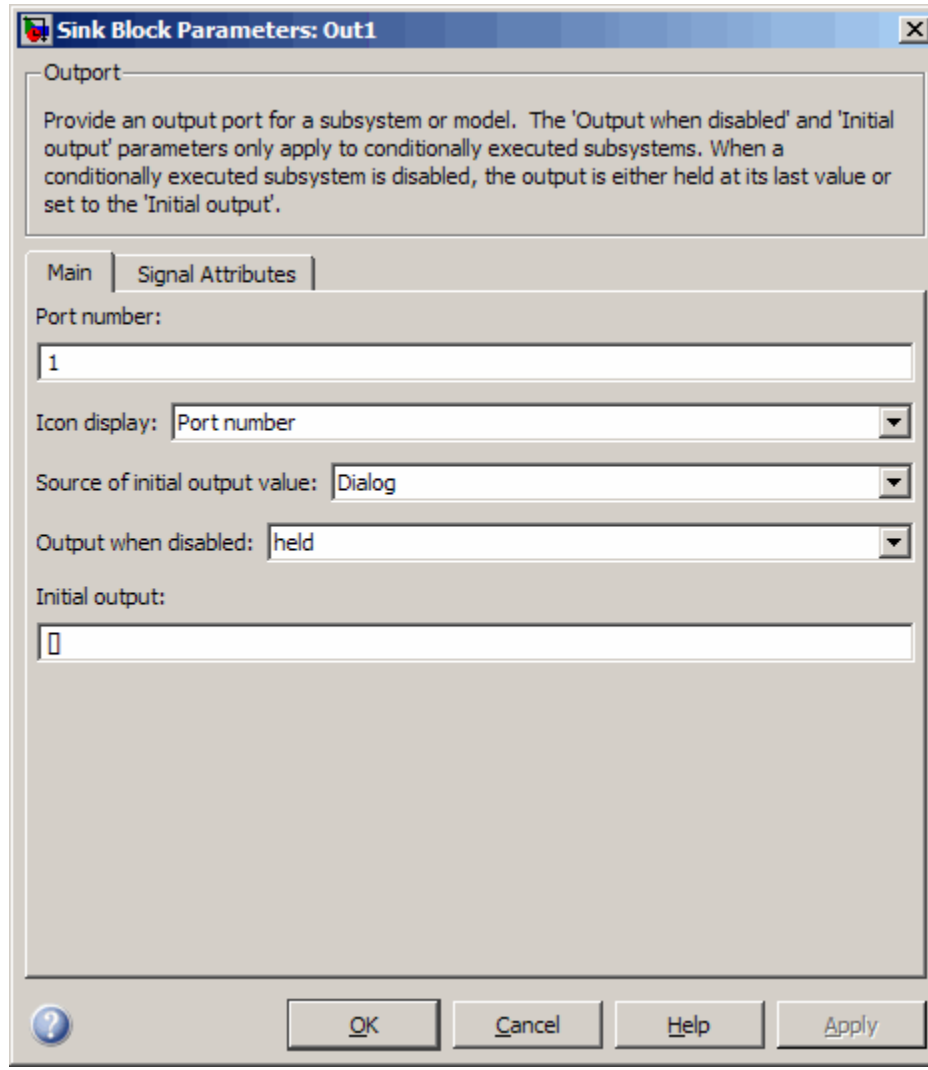
The Outport block accepts complex or real signals of any data type supported by Simulink software. An Outport block can also accept fixed-point and enumerated data types if the block is not a root-level output port. The complexity and data type of the block's output are the same as those of its input. For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the Simulink documentation.

The elements of a signal array connected to an Outport block can be of differing complexity and data types except in the following circumstance: If the output port is in a conditionally executed subsystem and the initial output is specified, all elements of an input array must be of the same complexity and data types.

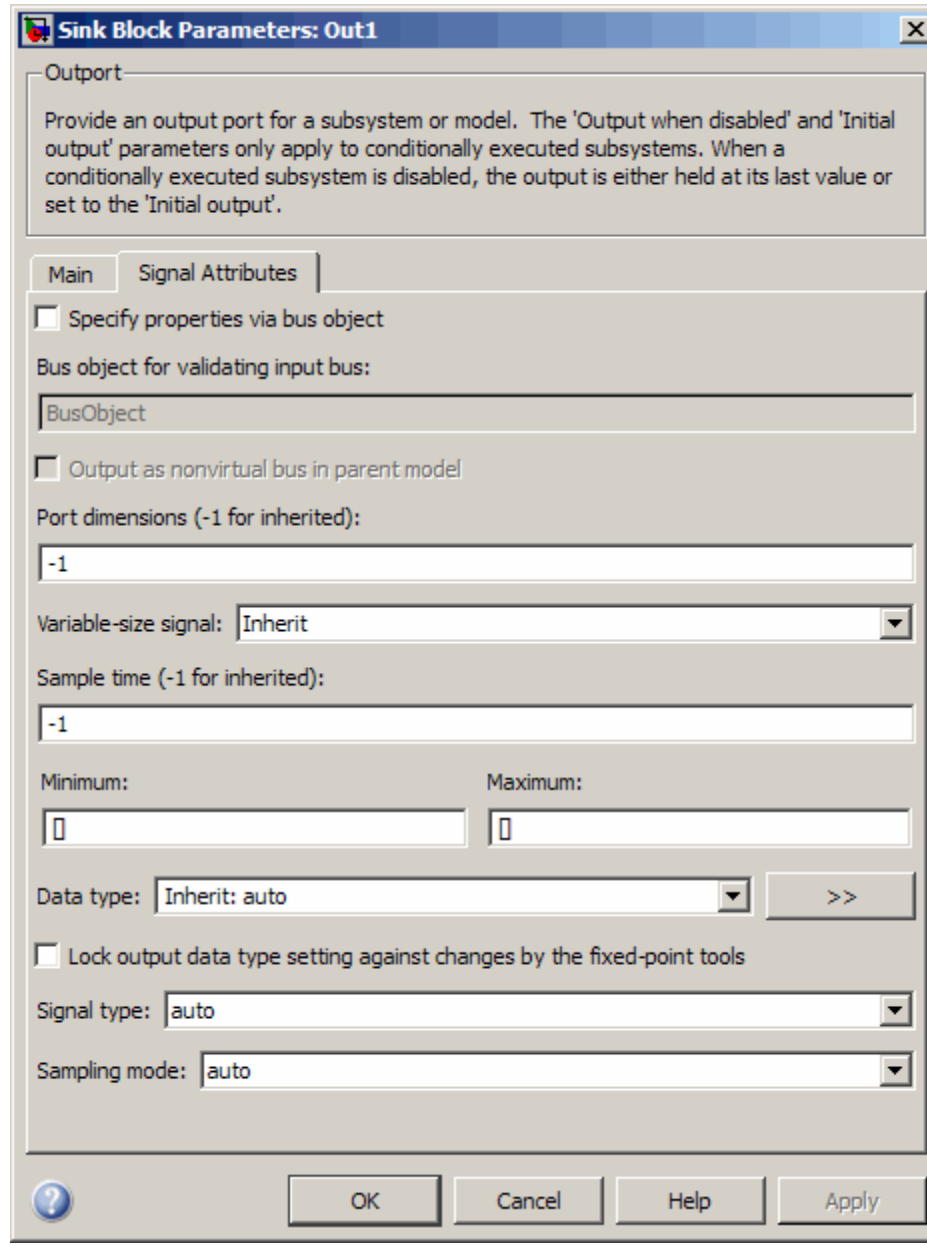
Typical Simulink data type conversion rules apply to an output port's **Initial output** parameter. If the initial output value is in the range of the block's output data type, Simulink software converts the initial output to the output data type. If the specified initial output is out of the range of the output data type, Simulink software halts the simulation and signals an error.

Parameters and Dialog Box

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



The **Signal Attributes** pane of the Outport block dialog box appears as follows:


Sink Block Parameters: Out1
Outport
Provide an output port for a subsystem or model. The 'Output when disabled' and 'Initial output' parameters only apply to conditionally executed subsystems. When a conditionally executed subsystem is disabled, the output is either held at its last value or set to the 'Initial output'.
Main
Signal Attributes
Specify properties via bus object
Bus object for validating input bus:
BusObject
Output as nonvirtual bus in parent model
Port dimensions (-1 for inherited):
-1
Variable-size signal:
Inherit
Sample time (-1 for inherited):
-1
Minimum:
Maximum:
Data type:
Inherit: auto
>>
Lock output data type setting against changes by the fixed-point tools
Signal type:
auto
Sampling mode:
auto
OK
Cancel
Help
Apply

- “Bias” on page 2-1321

Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User's Guide* for more information.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Port number

Specify the port number of the block.

Settings

Default: 1

This parameter controls the order in which the port that corresponds to the block appears on the parent subsystem or model block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Icon display

Specify the information to be displayed on the icon of this input port.

Settings

Default: `Port number`

`Signal name`
: Display the name of the signal connected to this port (or signals if the input is a bus).

`Port number`
: Display port number of this port.

`Port number and signal name`
: Display both the port number and the names of the signals connected to this port.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Source of initial output value

Select the source of the initial output value of the Outport block.

Settings

Default: `Dialog`

`Dialog`
: The initial output value is specified by the **Initial output** parameter on the dialog.

`Input signal`
: The initial output value is inherited from the input signal.

Tips

- If you are using classic initialization mode, you *must* select `Dialog`. Selecting `Input signal` will cause an error.
- If you are using classic initialization mode and want to inherit the initial output value, set this parameter to `Dialog`, and then specify `[]` (empty matrix) for the **Initial output** value. For more information, see Inheriting Initial Conditions on page 814.

Dependencies

This parameter applies only if the Outport resides in an Enabled Subsystem.

Selecting `Dialog` enables the following parameters:

- **Output when disabled**
- **Initial output**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Output when disabled

Specify what happens to the block output when the subsystem is disabled.

Settings

Default: `held`

`held`
: Output is held when the subsystem is disabled.

`reset`
: Output is reset when the subsystem is disabled.

Tips

- If you are connecting the output of a conditionally executed subsystem to a Merge block, set **Output when disabled** to `held` to ensure consistent simulation results.
- If you are using simplified initialization mode, you must select `held` when connecting a conditionally executed subsystem to a Merge block.

Dependencies

- Selecting `Dialog` in **Source of initial output value** enables this parameter.
- This parameter applies only if the Outport resides in an Enabled Subsystem.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Initial output

For conditionally executed subsystems, specify the block output before the subsystem executes and while it is disabled.

Settings

Default: [ ]

Simulink software does not allow the initial output of this block to be `inf` or `NaN`.

Tips

- If you are using classic initialization mode, specify `[]` (empty matrix) to inherit the initial output value from the input signal. For more information, see Inheriting Initial Conditions on page 814.
- You can also specify `[]` if you are using classic initialization mode, and your model does not depend on the initial output of the conditionally executed subsystem.
- If you are using simplified initialization mode, you cannot specify `[]`, you must specify an explicit value. If you do not want to specify an initial output value for this block, set **Source of initial output value** to `Input signal`.

Dependencies

- Selecting `Dialog` in **Source of initial output value** enables this parameter.
- This parameter applies only if the Outport resides in an Enabled Subsystem.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Specify properties via bus object

Use a bus object to define the structure of the bus that is input or output by this block.

Settings

Default: Off

 On

Use a bus object to define the structure of the bus that is input or output by this block.

 Off

Do not use a bus object to define the structure of the bus that is input or output by this block.

Tips

Selecting this parameter is required if the bus is nonvirtual (including a nonvirtual bus that was converted from a virtual bus as described in "Automatic Bus Conversion") and is optional otherwise.

Dependencies

This parameter enables **Bus object for validating input bus**.

This parameter enables **Output as nonvirtual bus in parent model**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Bus object for validating input bus

Specify the name of the bus object that defines the structure that a bus must have to connect to this port.

Settings

Default: `BusObject`

A bus object is an instance of class `Simulink.Bus` that is defined in the base workspace.

Tips

At the beginning of a simulation or when you update the model's diagram, Simulink software checks whether the bus connected to this port has the specified structure. If not, Simulink software displays an error message.

Dependencies

Specify properties via bus object enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output as nonvirtual bus in parent model

Select this parameter if you want the bus emerging in the parent model to be nonvirtual. The bus that is input to the port can be virtual or nonvirtual, regardless of the setting of **Output as nonvirtual bus in parent model**.

Settings

Default: Off

 On

Select this parameter if you want the bus emerging in the parent model to be nonvirtual.

 Off

Clear this parameter if you want the bus emerging in the parent model to be virtual.

Tips

All signals in a nonvirtual bus must have the same sample time, even if the elements of the associated bus object specify inherited sample times. Any bus operation that would result in a nonvirtual bus that violates this requirement generates an error. Therefore, if you select this option all signals in the bus must have the same sample time. You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus, to allow the signal or bus to be included in a nonvirtual bus.

Dependency

Specify properties via bus object enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Port dimensions (-1 for inherited)

Specify the dimensions that a signal must have in order to be connected to this Outport block.

Settings

Default: `-1`

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.

Dependency

Clearing **Specify properties via bus object** enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Variable-size signal

Specify the type of signals allowed out of this port.

Settings

Default: `Inherit`

`Inherit`
: Allow variable-size and fixed-size signals.

`No`
: Do not allow variable-size signals.

`Yes`
: Allow only variable-size signals.

Dependencies

When the signal at this port is a variable-size signal, the **Port dimensions** parameter specifies the maximum dimensions of the signal.

Command-Line Information

Parameter: `VarSizeSig`
Type: string
Value: `'Inherit'| 'No' | 'Yes'`
Default: `'Inherit'`

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits or specify another appropriate sample time such as continuous or inherited.

Settings

Default: -1

By default, the block inherits its sample time based upon the context of the block within the model. To set a different sample time, enter a valid sample time based upon the table in "Types of Sample Time".

See also "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Signal type

Specify the numeric type of the signal output by this block.

Settings

Default: `auto`

`auto`
: Output the numeric type of the signal that is connected to its input.

`real`
: Output a real-valued signal. The signal connected to this block must be real. If it is not, Simulink software displays an error if you try to update the diagram or simulate the model that contains this block.

`complex`
: Output a complex signal. The signal connected to this block must be complex. If it is not, Simulink software displays an error if you try to update the diagram or simulate the model that contains this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Sampling mode

Specify the sampling mode (`Sample based` or `Frame based`) that the input signal must match.

Settings

Default: `auto`

`auto`
: Accept any sampling mode.

`Sample based`
: The output signal is sample-based.

`Frame based`
: The output signal is frame-based.

Tips

To generate frame-based signals, you must have the Signal Processing Blockset product installed.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Data type

Specify the output data type of the external input.

Settings

Default: `Inherit: auto`

`Inherit: auto`
: A rule that inherits a data type

`double`
: Data type is `double`.

`single`
: Data type is `single`.

`int8`
: Data type is `int8`.

`uint8`
: Data type is `uint8`.

`int16`
: Data type is `int16`.

`uint16`
: Data type is `uint16`.

`int32`
: Data type is `int32`.

`uint32`
: Data type is `uint32`.

`boolean`
: Data type is `boolean`.

`fixdt(1,16,0)`
: Data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
Data type is enumerated.

`<data type expression>`
The name of a data type object, for example `Simulink.NumericType`

Tips

This parameter can also be an expression that evaluates to a data type, for example, `float('single')`

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<class name>`

`Expression`
: Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

See “Specifying a Fixed-Point Data Type” in the *Simulink User’s Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Characteristics

Sample Time	Inherited from the driving block
Dimensionalized	Yes
Multidimensionalized	Yes
Virtual	Yes, when the block resides in a subsystem block and not at the root level of a model For more information, see "Virtual Blocks" in the Simulink documentation.

See Also

Inport

Purpose

Rearrange dimensions of multidimensional array dimensions

Library

Math Operations

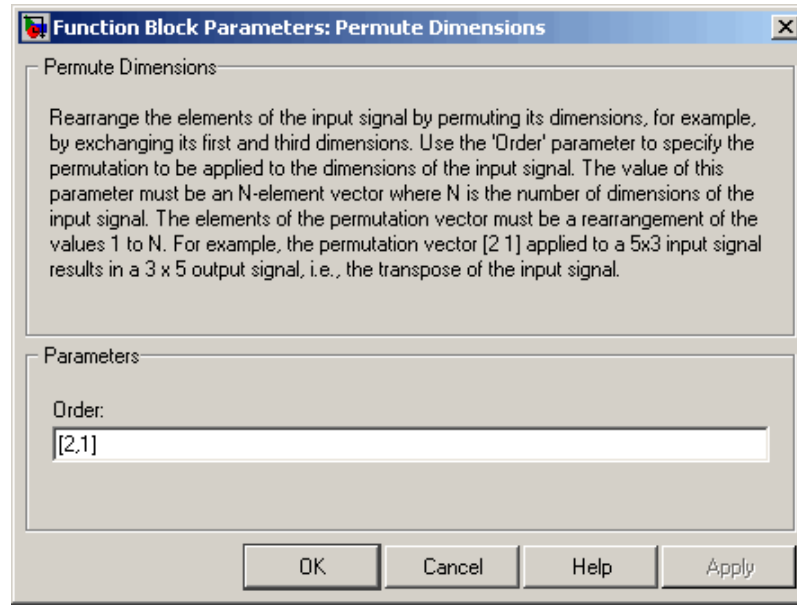
Description

The block reorders the elements of the input signal so that they are in the order that you specify in the **Order** parameter.

Data Type Support

Accepts signals of any data type supported by Simulink software, including fixed-point and enumerated data types. Output must be the same data type as the input.

Parameters and Dialog Box



Order

Specify the permutation order to apply to the dimensions of the input signal. This parameter is a vector of elements, where the number of elements in the vector is the number of dimensions of the input signal.

Characteristics

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

See Also

Math Function (`transpose`), `permute` (in the MATLAB reference documentation)

Purpose

Simulate continuous- or discrete-time PID controllers

Library

Continuous, Discrete

Description

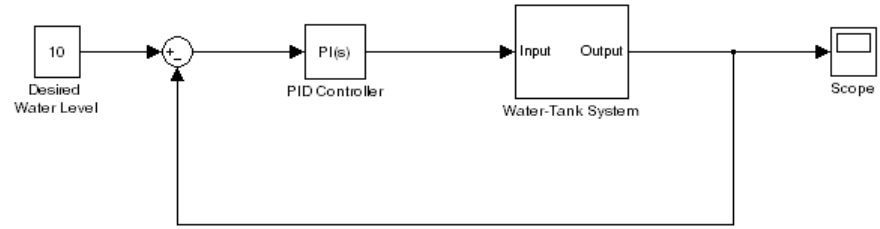
Implement a continuous- or discrete-time controller (PID, PI, PD, P, or I) in your Simulink model. PID controller gains are tunable either manually or automatically. Automatic tuning requires Simulink® Control Design™ software (PID Tuner or SISO Design Tool).

The PID Controller block output is a weighted sum of the input signal, the integral of the input signal, and the derivative of the input signal. The weights are the proportional, integral, and derivative gain parameters. A first-order pole filters the derivative action.

Configurable options in the PID Controller block include:

- Controller type and form
- Time domain (continuous or discrete)
- Initial conditions and reset trigger
- Output saturation limits and built-in anti-windup mechanism
- Signal tracking for bumpless control transfer and multiloop control

In one common implementation, the PID Controller block operates in the feedforward path of the feedback loop:



The input of the block is typically an error signal, which is the difference between a reference signal and the system output. For a two-input block that permits setpoint weighting, see the PID Controller (2 DOF) block reference page.

You can generate code to implement your controller using any Simulink data type, including fixed-point data types. (Code generation requires Real-Time Workshop software; fixed-point implementation requires Fixed-Point Toolbox.)

Data Type Support

The PID Controller block accepts real signals of any numeric data type that Simulink software supports, including fixed-point data types. See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Parameters

The following table summarizes the PID Controller block parameters, accessible via the block parameter dialog box.

Task	Parameters
Choose controller form and type.	• **Controller Form** in Main tab • **Controller**
Choose discrete or continuous time.	• **Time-domain** • **Sample time**
Choose an integration method (discrete time).	• **Integrator method** • **Filter method**

Task	Parameters
Set and tune controller gains.	• **Proportional (P)** in Main tab • **Integral (I)** in Main tab • **Derivative (D)** in Main tab • **Filter coefficient (N)** in Main tab
Set integrator and filter initial conditions.	• **Initial conditions Source** in Main tab • **Integrator Initial condition** in Main tab • **Filter Initial condition** in Main tab • **External reset** in Main tab • **Ignore reset when linearizing** in Main tab
Limit block output.	• **Limit output** in PID Advanced tab • **Lower saturation limit** in PID Advanced tab • **Upper saturation limit** in PID Advanced tab • **Ignore saturation when linearizing** in PID Advanced tab
Configure anti-windup mechanism (when you limit block output).	• **Anti-windup method** in PID Advanced tab • **Back-calculation gain (Kb)** in PID Advanced tab
Enable signal tracking.	• **Enable tracking mode** in PID Advanced tab • **Tracking gain (Kt)** in PID Advanced tab

Task	Parameters
Configure data types.	• **Parameter data type** in Data Type Attributes tab • **Product output data type** in Data Type Attributes tab • **Summation output data type** in Data Type Attributes tab • **Accumulator data type** in Data Type Attributes tab • **Integrator output data type** in Data Type Attributes tab • **Filter output data type** in Data Type Attributes tab • **Saturation output data type** in Data Type Attributes tab • **Lock output data type setting against changes by the fixed-point tools** in Data Type Attributes tab • **Saturate on integer overflow** in Data Type Attributes tab • **Integer rounding mode** in Data Type Attributes tab
Configure block for code generation.	• **State name** in State Attributes tab • **State name must resolve to Simulink signal object** in State Attributes tab • **Real-Time Workshop storage class** in State Attributes tab • **Real-Time Workshop storage type qualifier** in State Attributes tab

Controller form

Select the controller form.

Settings

`Parallel` (Default)

Selects a controller form in which the output is the sum of the proportional, integral, and derivative actions, weighted according to the independent gain parameters **P**, **I**, and **D**. The filter coefficient **N** sets the location of the pole in the derivative filter. For a continuous-time parallel PID controller, the transfer function is:

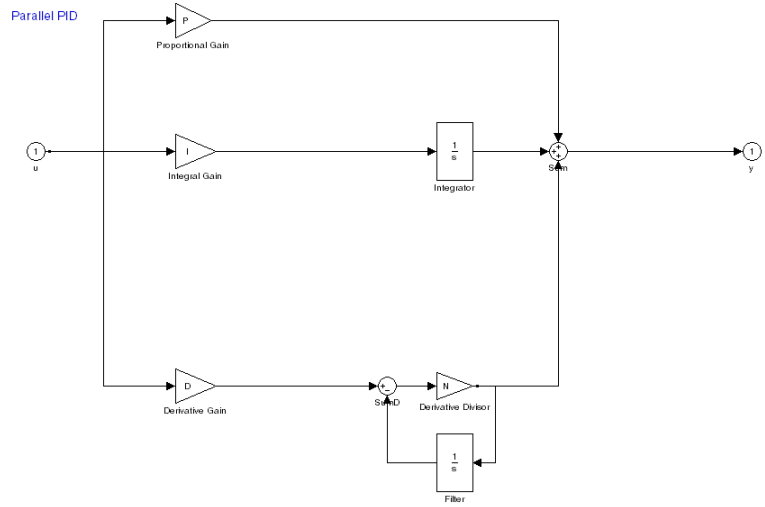
$$C_{par}(s) = \left[P + I\left(\frac{1}{s}\right) + D\left(\frac{Ns}{s+N}\right) \right]$$

For a discrete-time parallel PID controller, the transfer function takes the form:

$$C_{par}(z) = P + Ia(z) + D\left[\frac{N}{1+Nb(z)}\right]$$

where the **Integrator method** determines `a(z)` and the **Filter method** determines `b(z)` (for sampling time T_s):

	Forward Euler method	Backward Euler method	Trapezoidal method
$a(z)$ (determined by Integrator method)	$\frac{T_s}{z-1}$	$\frac{T_s z}{z-1}$	$\frac{T_s}{2}\frac{z+1}{z-1}$
$b(z)$ (determined by Filter method)	$\frac{T_s}{z-1}$	$\frac{T_s z}{z-1}$	$\frac{T_s}{2}\frac{z+1}{z-1}$



Parallel PID Controller

`Ideal`

Selects a controller form in which the proportional gain `P` acts on the sum of all actions. The transfer functions are the same as

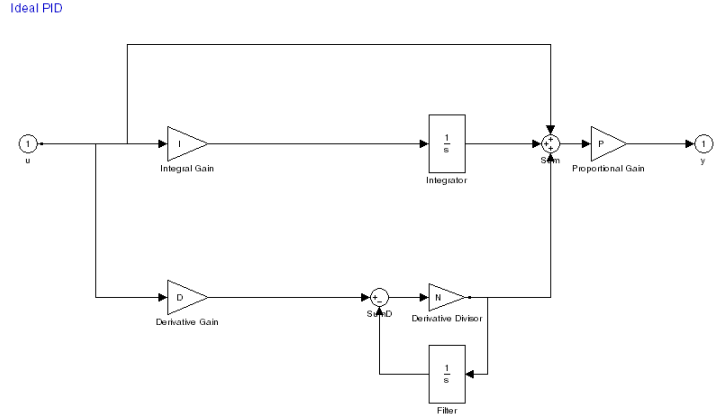
for the parallel form, except that `P` multiplies all terms. For a continuous-time ideal PID controller, the transfer function is:

$$C_{id}(s) = P\left[1 + I\left(\frac{1}{s}\right) + D\left(\frac{Ns}{s+N}\right)\right]$$

For a discrete-time ideal PID controller the transfer function is:

$$C_{id}(z) = P\left[1 + Ia(z) + D\frac{N}{1+Nb(z)}\right]$$

where the **Integrator method** determines `a(z)` and the **Filter method** determines `b(z)` as described previously for the parallel controller form.



Ideal PID Controller

Controller

Specify the controller type.

Settings

`PID` (Default)

Implements a controller with proportional, integral, and derivative action.

`PI`

Implements a controller with proportional and integral action.

`PD`

Implements a controller with proportional and derivative action.

`P`

Implements a controller with proportional action.

`I`

Implements a controller with integral action.

Time-domain

Select continuous or discrete time domain. The appearance of the block changes to reflect your selection.

Settings

`Continuous-time` (Default)

Selects the continuous-time representation.

`Discrete-time`

Selects the discrete-time representation. Selecting `Discrete-time` also allows you to specify the:

- **Sample time**, which is the discrete interval between samples.
- Discrete integration methods for the integrator and the derivative filter using the **Integrator method** and **Filter method** menus.

Integrator method

(Available only when you set **Time-domain** to `Discrete-time`.) Specify the method used to compute the integrator output. For more information about discrete-time integration methods, see the Discrete-Time Integrator block reference page.

Settings

`Forward Euler` (Default)

Selects the Forward Rectangular (left-hand) approximation.

- This method is best for small sampling times, where the Nyquist limit is large compared to the bandwidth of the controller. For larger sampling times, the `Forward Euler` method can result in instability, even when discretizing a system that is stable in continuous time.

`Backward Euler`

Selects the Backward Rectangular (right-hand) approximation.

- If you are generating code using Real-Time Workshop or using the Fixed-Point Toolbox and you activate the `Back-calculation` **Anti-windup method**, this integration method can cause algebraic loops in your controller. Algebraic loops can lead to slower performance of generated code. For more information about algebraic loops in Simulink models, see "Algebraic Loops" in the Simulink documentation.
- An advantage of the `Backward Euler` method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result.

`Trapezoidal`

Selects the Bilinear approximation.

- If you are generating code using Real-Time Workshop or using the Fixed-Point Toolbox and you activate the `Back-calculation` **Anti-windup method**, this integration method can cause algebraic loops in your controller. Algebraic loops can lead to slower performance of generated code. For

more information about algebraic loops in Simulink models, see "Algebraic Loops" in the Simulink documentation.

- An advantage of the `Trapezoidal` method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result. Of all available integration methods, the `Trapezoidal` method yields the closest match between frequency-domain properties of the discretized system and the corresponding continuous-time system.

Filter method

(Available only when you set **Time-domain** to `Discrete-time`.) Specify the method used to compute the derivative filter output. For more information about discrete-time integration methods, see the Discrete-Time Integrator block reference page.

Settings

`Forward Euler` (Default)

Selects the Forward Rectangular (left-hand) approximation.

- This method is best for small sampling times, where the Nyquist limit is large compared to the bandwidth of the controller. For larger sampling times, the `Forward Euler` method can result in instability, even when discretizing a system that is stable in continuous time.

`Backward Euler`

Selects the Backward Rectangular (right-hand) approximation.

- If you are generating code using Real-Time Workshop or using the Fixed-Point Toolbox, this filter method can cause algebraic loops in your controller. Algebraic loops can lead to slower performance of generated code. For more information about algebraic loops in Simulink models, see “Algebraic Loops” in the Simulink documentation.
- An advantage of the `Backward Euler` method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result. Any filter parameter value `N > 0` yields a stable result with this method.

`Trapezoidal`

Selects the Bilinear approximation.

- If you are generating code using Real-Time Workshop or using the Fixed-Point Toolbox, this filter method can cause algebraic loops in your controller. Algebraic loops can lead to slower performance of generated code. For more information about algebraic loops in Simulink models, see “Algebraic Loops” in the Simulink documentation.

- An advantage of the `Trapezoidal` method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result. Any filter parameter value `N > 0` yields a stable result with this method. Of all available filter methods, the `Trapezoidal` method yields the closest match between frequency-domain properties of the discretized system and the corresponding continuous-time system.

Sample time (-1 for inherited)

(Available only when you set **Time-domain** to `Discrete-time`.) Specify the discrete interval between samples.

Settings

Default: 1

By default, the block uses a discrete sample time of 1. To specify a different sample time, enter another discrete value, such as 0.1.

If you specify a value of -1, the PID Controller block inherits the sample time from the upstream block. Do not enter a value of 0; to implement a continuous-time controller, select the **Time-domain** `Continuous-time`.

See "How to Specify the Sample Time" in the online documentation for more information.

Proportional (P)

(Available for PID, PD, PI, and P controllers.) Specify the proportional gain `P`.

Default: 1

Enter a finite, real gain value into the **Proportional (P)** field. Use either scalar or vector gain values. For a `Parallel` PID **Controller form**, the proportional action is independent of the integral and derivative actions. For an `Ideal` PID **Controller form**, the proportional action acts on the integral and derivative actions. See **Controller form** for more information about the role of `P` in the controller transfer function.

When you have Simulink Control Design software installed, you can automatically tune the controller gains using the PID Tuner or the SISO Design Tool. See “Designing Compensators” in the Simulink Control Design documentation.

Integral (I)

(Available for PID, PI, and I controllers.) Specify the integral gain `I`.

Default: 1

Enter a finite, real gain value into the **Integral (I)** field. Use either scalar or vector gain values.

When you have Simulink Control Design software installed, you can automatically tune the controller gains using the PID Tuner or the SISO Design Tool. See “Designing Compensators” in the Simulink Control Design documentation.

Derivative (D)

(Available for PID and PD controllers.) Specify the derivative gain `D`.

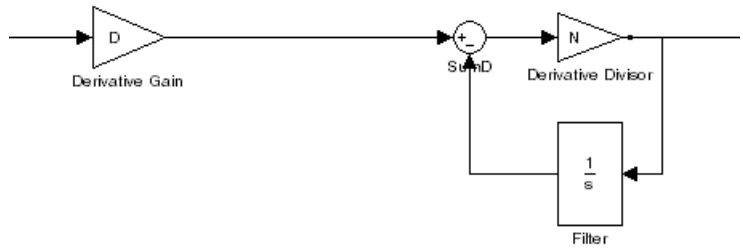
Default: 0

Enter a finite, real gain value into the **Derivative (D)** field. Use either scalar or vector gain values.

When you have Simulink Control Design software installed, you can automatically tune the controller gains using the PID Tuner or the SISO Design Tool. See “Designing Compensators” in the Simulink Control Design documentation.

Filter coefficient (N)

(Available for PID and PD controllers.) Specify the filter coefficient `N`, which determines the pole location of the filter in the derivative action:



The filter pole falls at `s = -N` in the `Continuous-time` **Time-domain**. For `Discrete-time`, the location of the pole depends on which **Filter method** you select (for sampling time T_s):

- `Forward Euler`:

  $$z_{pole} = 1 - NT_s$$

- `Backward Euler`:

  $$z_{pole} = \frac{1}{1 + NT_s}$$

- `Trapezoidal`:

  $$z_{pole} = \frac{1 - NT_s/2}{1 + NT_s/2}$$

Default: 100.

Enter a finite, real gain value into the **Filter Coefficient (N)** field. Use either scalar or vector gain values. Note that the PID controller block does not support `N = inf` (ideal unfiltered derivative).

When you have Simulink Control Design software installed, you can automatically tune the controller gains using the PID Tuner or the SISO Design Tool. See “Designing Compensators” in the Simulink Control Design documentation. Automatic tuning requires `N > 0`.

Initial conditions Source

(Only available for controllers with integral or derivative action.) Select the source of the integrator and filter initial conditions. Simulink uses initial conditions to initialize the integrator and filter output at the start of a simulation or at a specified trigger event (See **External Reset**). The integrator and filter initial conditions in turn determine the initial block output.

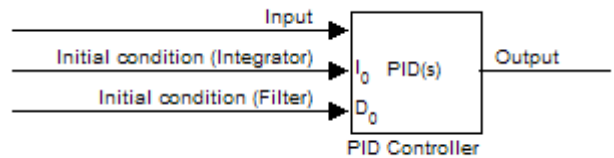
Settings

`internal` (Default)

Specifies the integrator and filter initial conditions explicitly using the **Integrator Initial condition** and **Filter Initial condition** parameters.

`external`

Specifies the integrator and filter initial conditions externally. An additional input port appears under the block input for each initial condition: I_0 for the integrator and D_0 for the filter:



Integrator Initial condition

(Available only when **Initial conditions Source** is `internal` and the controller includes integral action.) Specify the integrator initial value. Simulink uses the initial condition to initialize the integrator output at the start of a simulation or at a specified trigger event (see **External Reset**). The integrator initial condition, together with the filter initial condition, determines the initial output of the PID controller block.

Default: 0

Simulink does not permit the integrator initial condition to be `inf` or `NaN`.

Filter Initial condition

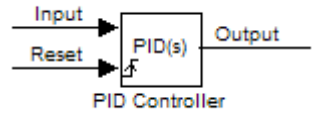
(Available only when **Initial conditions Source** is `internal` and the controller includes integral action.) Specify the filter initial value. Simulink uses the initial condition to initialize the filter output at the start of a simulation or at a specified trigger event (see **External Reset**). The filter initial condition, together with the integrator initial condition, determines the initial output of the PID controller block.

Default: 0

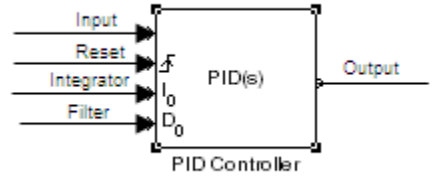
Simulink does not permit the filter initial condition to be `inf` or `NaN`.

External reset

Select the trigger event that resets the integrator and filter outputs to the initial conditions you specify in the **Integrator Initial condition** and **Filter Initial condition** fields. Selecting any option other than `none` enables a reset input on the block for the external reset signal, as shown:



Or, if the **Initial conditions Source** is `External`,



Settings

`none` (Default)
: Does not reset the integrator and filter outputs to initial conditions.

`rising`
: Resets the outputs when the reset signal has a rising edge.

`falling`
: Resets the outputs when the reset signal has a falling edge.

`either`
: Resets the outputs when the reset signal either rises or falls.

`level`

Resets and holds the outputs to the initial conditions while the reset signal is nonzero.

Note To be compliant with the Motor Industry Software Reliability Association (MISRA) software standard, your model must use Boolean signals to drive the external reset ports of the PID controller block.

Ignore reset when linearizing

Force Simulink linearization commands to ignore any reset mechanism that you have chosen with the **External reset** menu. Ignoring reset states allows you to linearize a model around an operating point even if that operating point causes the PID Controller block to reset.

Settings

 `Off` (Default)

Simulink linearization commands do not ignore states corresponding to the reset mechanism.

 `On`

Simulink linearization commands ignore states corresponding to the reset mechanism.

Enable zero-crossing detection

Enable zero-crossing detection in continuous-time models upon reset and upon entering or leaving a saturation state.

Zero-crossing detection can accurately locate signal discontinuities without resorting to excessively small time steps that can lead to lengthy simulation times. If you select **Limit output** or activate an **External reset** in your PID Controller block, activating zero-crossing detection can reduce computation time in your simulation. For more information, see "Zero-Crossing Detection" in the Simulink documentation.

Settings


`On` (Default)

Uses zero-crossing detection at any of the following events: reset; entering or leaving an upper saturation state; and entering or leaving a lower saturation state.


`Off`

Does not use zero-crossing detection.

Enabling zero-crossing detection for the PID Controller block also enables zero-crossing detection for all under-mask blocks that include the zero-crossing detection feature.

Limit output

Limit the block output to values you specify as the **Lower saturation limit** and **Upper saturation limit** parameters.

Activating this option limits the block output internally to the block, obviating the need for a separate Saturation block after the controller in your Simulink model. It also allows you to activate the block's built-in anti-windup mechanism (see **Anti-windup method**).

Settings


`Off` (Default)

Does not limit the block output, which equals the weighted sum of the proportional, integral, and derivative actions.


`On`

Limits the block output to the **Lower saturation limit** or the **Upper saturation limit** whenever the weighted sum exceeds those limits. Allows you to select an **Anti-windup method**.

Lower saturation limit

(Available only when you select the **Limit Output** box.) Specify the lower limit for the block output. The block output is held at the **Lower saturation limit** whenever the weighted sum of the proportional, integral, and derivative actions goes below that value.

Default: `-inf`

Upper saturation limit

(Available only when you select the **Limit Output** box.) Specify the upper limit for the block output. The block output is held at the **Upper saturation limit** whenever the weighted sum of the proportional, integral, and derivative actions exceeds that value.

Default: `inf`

Anti-windup method

(Available only when you select the **Limit Output** option and the controller includes integral action.) Select an anti-windup mechanism to discharge the integrator when the block is saturated, which occurs when the sum of the block components exceeds the output limits.

When you select the **Limit output** check box and the weighted sum of the controller components exceeds the specified output limits, the block output holds at the specified limit. However, the integrator output can continue to grow (integrator wind-up), increasing the difference between the block output and the sum of the block components. Without a mechanism to prevent integrator wind-up, two results are possible:

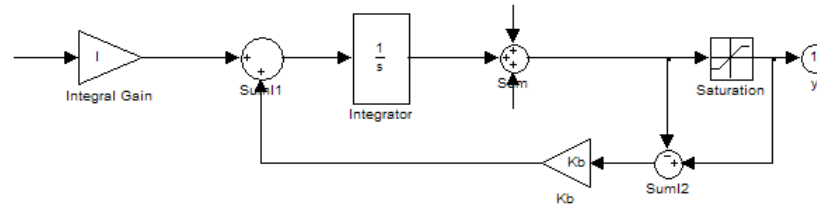
- If the sign of the input signal never changes, the integrator continues to integrate until it overflows. The overflow value is the maximum or minimum value for the data type of the integrator output.
- If the sign of the input signal changes once the weighted sum has grown beyond the output limits, it can take a long time to discharge the integrator and return the weighted sum within the block saturation limit.

In both cases, controller performance can suffer. To combat the effects of wind-up without an anti-windup mechanism, it may be necessary to detune the controller (for example, by reducing the controller gains), resulting in a sluggish controller. Activating an anti-windup mechanism can improve controller performance.

Settings

`none` (Default)
: Does not use an anti-windup mechanism. This setting may cause the block's internal signals to be unbounded even if the output appears to be bounded by the saturation limits. This can result in slow recovery from saturation or unexpected overflows.

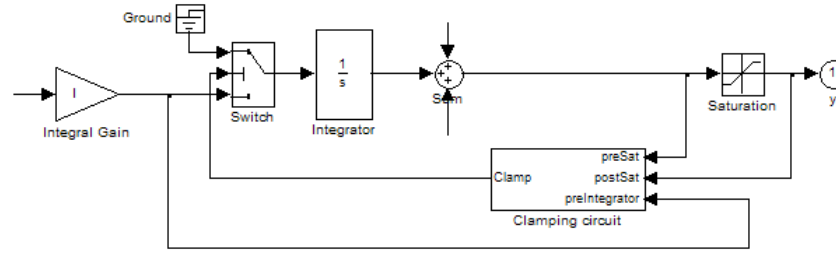
`back-calculation`
: Discharges the integrator when the block output saturates using this feedback loop:



You can also specify a value for the **Back-calculation coefficient (Kb)**.

`clamping`

Stops integration when the sum of the block components exceeds the output limits and the integrator output and block input have the same sign. Resumes integration when the sum of the block components exceeds the output limits and the integrator output and block input have opposite sign. The integrator portion of the block is:



where the clamping circuit implements the logic necessary to determine whether integration continues.

Back-calculation gain (Kb)

(Available only when the `back-calculation` **Anti-windup method** is active.) Specify the gain coefficient of the anti-windup feedback loop.

The `back-calculation` anti-windup method discharges the integrator on block saturation using a feedback loop having gain coefficient `Kb`.

Default: 1

Ignore saturation when linearizing

Force Simulink linearization commands ignore PID Controller block output limits. Ignoring output limits allows you to linearize a model around an operating point even if that operating point causes the PID Controller block to exceed the output limits.

Settings

 `Off` (Default)

Simulink linearization commands do not ignore states corresponding to saturation.

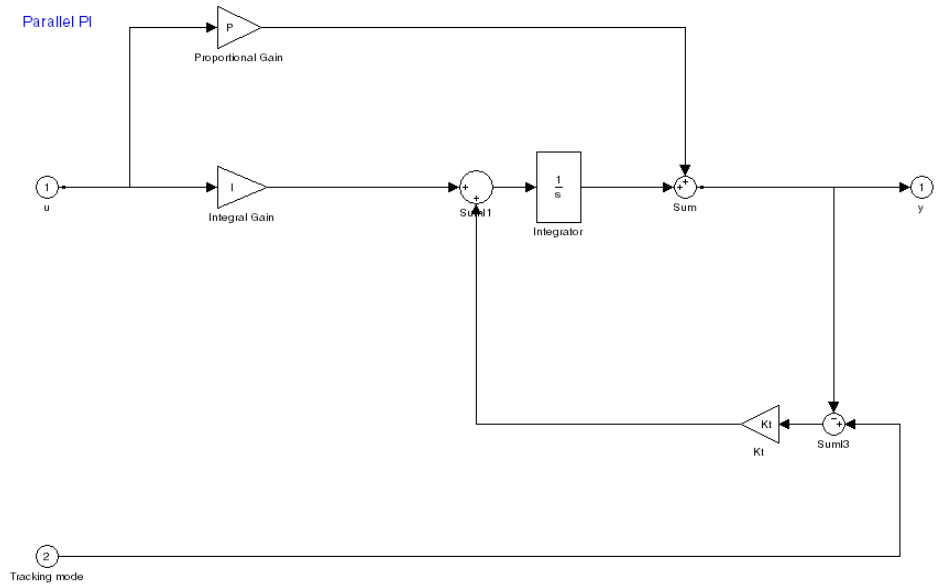
 `On`

Simulink linearization commands ignore states corresponding to saturation.

Enable tracking mode

(Available for any controller with integral action.) Activate signal tracking, which lets the output of the PID Controller block follow a tracking signal. Provide the tracking signal to the block at the `TR` port, which becomes active when you select **Enable tracking mode**.

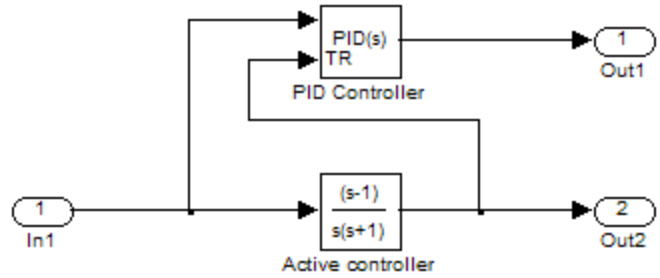
When signal tracking is active, the difference between the tracked signal and the block output is fed back to the integrator input with a gain `Kt`. The structure is illustrated for a PI controller:



You can also specify the **Tracking coefficient (Kt)**.

Bumpless control transfer

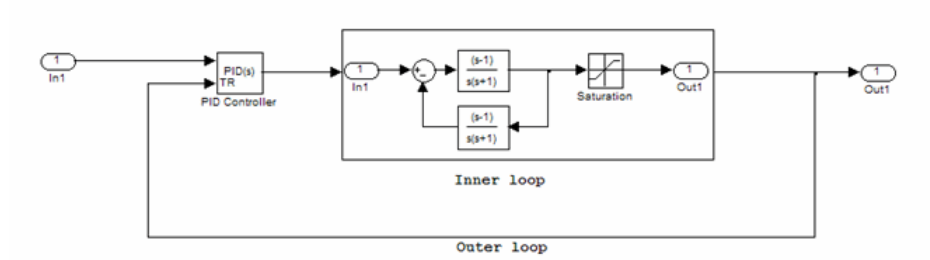
Use signal tracking, for example, to achieve bumpless control transfer in systems that switch between two controllers. You can make one controller track the output of the other controller by connecting `TR` port to the signal to be tracked. For example:



In this example, the outputs `Out1` and `Out2` can drive a controlled system (not shown) through a switch that transfers control between the "Active controller" block and the PID Controller block. The signal tracking feature of the PID Controller block provides smooth operation upon transfer of control from one controller to another, ensuring that the two controllers have the same output at the time of transfer.

Multiloop control

Use signal tracking to prevent block wind-up in multiloop control approaches, as this example illustrates:



In this example, inner loop has an effective gain of 1 when it is not saturated. Without signal tracking, inner loop winds up in saturation. Signal tracking ensures that the PID Controller output does not exceed the saturated output of the inner loop.

Settings

 `Off` (Default)

Disables signal tracking and removes `TR` block input.

 `On`

Enables signal tracking and activates `TR` input.

Tracking gain (Kt)

(Available only when you select **Enable tracking mode**.) Specify `Kt`, which is the gain of the signal tracking feedback loop.

Default: 1

Parameter data type

Select the data type of the gain parameters **P**, **I**, **D**, **N**, **Kb**, and **Kt**.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Inherit via back propagation`

Use data type of the driving block.

`Inherit: Same as input`

Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Product output data type

Select the product output data type of the gain parameters **P**, **I**, **D**, **N**, **Kb**, and **Kt** .

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Inherit via back propagation`

Use data type of the driving block.

`Inherit: Same as input`

Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Summation output data type

Select the summation output data type of the sums **Sum**, **Sum D**, **Sum I1** , **SumI2** ,and **SumI3**, which are sums computed internally within the block. To see where Simulink computes each of these sums , right-click the PID Controller block in your model and select `Look Under Mask`:

- **Sum** is the weighted sum of the proportional, derivative, and integral signals.
- **SumD** is the sum in the derivative filter feedback loop.
- **SumI1** is the sum of the block input signal (weighted by the integral gain `I`) and **SumI2**. **SumI1** is computed only when **Limit output** and **Anti-windup method** `back-calculation` are active.
- **SumI2** is the difference between the weighted sum **Sum** and the limited block output. **SumI2** is computed only when **Limit output** and **Anti-windup method** `back-calculation` are active.
- **SumI3** is the difference between the block output and the signal at the block's tracking input. **SumI3** is computed only when you select the **Enable tracking mode** box.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Same as first input`
Use data type of first input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
Name of a data type object. For example, `Simulink.NumericType`.

Accumulator data type

Specify the accumulator data type.

Settings

Default: `Inherit:  Inherit via internal rule`

`Inherit:  Inherit via internal rule`
: Use internal rule to determine accumulator data type.

`Inherit:  Same as first input`
: Use data type of first input signal.

`double`
: Accumulator data type is `double`.

`single`
: Accumulator data type is `single`.

`int8`
: Accumulator data type is `int8`.

`uint8`
: Accumulator data type is `uint8`.

`int16`
: Accumulator data type is `int16`.

`uint16`
: Accumulator data type is `uint16`.

`int32`
: Accumulator data type is `int32`.

`uint32`
: Accumulator data type is `uint32`.

`fixdt(1,16,0)`
: Accumulator data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Accumulator data type is fixed point `fixdt(1,16,2^0,0)`.

`<data type expression>`
: The name of a data type object, for example `Simulink.NumericType`

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Integrator output data type

Select the data type of the integrator output.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Same as input`

Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Filter output data type

Select the data type of the filter output.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Same as input`

Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Saturation output data type

Select the saturation output data type.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Same as input` (Default)
: Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Saturate on integer overflow

Specify whether overflows saturate.

Settings

Default: Off

 On

Overflows saturate.

 Off

Overflows do not saturate.

Tips

- When you select this check box, saturation applies to every internal operation on the block, not just the output or result.
- In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Floor`

`Ceiling`
: Round both positive and negative numbers toward positive infinity.

`Convergent`
: Round number to the nearest representable value. If a tie occurs, round to the nearest even stored value.

`Floor`
: Round both positive and negative numbers toward negative infinity.

`Nearest`
: Round number to the nearest representable value. If a tie occurs, round toward positive infinity.

`Round`
: Round number to the nearest representable value. If a tie occurs, round positive numbers toward positive infinity and round negative numbers toward negative infinity.

`Simplest`
: This option provides for an optimization of the rounding code for several blocks.

`Zero`
: Round number toward zero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Rounding” in the *Simulink Fixed Point User’s Guide*.

State name

Assign unique name to each state. The state names apply only to the selected block.

To assign a name to a single state, enter the name between quotes; for example, `'velocity'`.

To assign names to multiple states, enter a comma-delimited list surrounded by braces; for example, `{'a', 'b', 'c'}`. Each name must be unique. To assign state names with a variable that has been defined in the MATLAB workspace, enter the variable without quotes. The variable can be a string, cell, or structure.

Settings

Default: `' '` (no name)

State name must resolve to Simulink signal object

Require that state name resolve to Simulink signal object.

Settings

Default: Off

 On

Require that state name resolve to Simulink signal object.

 Off

Do not require that state name resolve to Simulink signal object.

Dependencies

State name enables this parameter.

This parameter enables **Real-Time Workshop storage class**.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Real-Time Workshop storage class

Select state storage class.

Settings

Default: `Auto`

`Auto`
: Auto is the appropriate storage class for states that you do not need to interface to external code.

`ExportedGlobal`
: State is stored in a global variable

`ImportedExtern`
: *`model`*`_private.h` declares the state as an extern variable.

`ImportedExternPointer`
: *`model`*`_private.h` declares the state as an extern pointer.

Dependencies

State name enables this parameter.

Setting this parameter to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables **Real-Time Workshop storage type qualifier**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Block State Storage Classes" in the *Real-Time Workshop User's Guide*.

Real-Time Workshop storage type qualifier

Specify Real-Time storage type qualifier.

Settings

Default: `' '`

If left blank, no qualifier is assigned.

Dependencies

Setting **Real-Time Workshop storage class** to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Direct Feedthrough	The following ports support direct feedthrough: • Reset port • Integrator and filter initial condition port • Input port, for every integration method except Forward Euler
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Supported for gain parameters **P**, **I**, and **D** and for filter coefficient **N**
States	Inherited from driving block and parameters
Dimensionalized	Yes
Zero-Crossing Detection	Yes (in continuous-time domain)

See Also

PID Controller (2 DOF), Gain, Integrator, Discrete-Time Integrator, Derivative, Discrete Derivative.

PID Controller (2 DOF)

Purpose

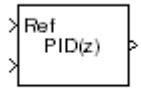
Simulate continuous- or discrete-time two-degree-of-freedom PID controllers

Library

Continuous, Discrete

Description

PID Controller (2DOF)



Discrete PID Controller (2DOF)

Implement a continuous- or discrete-time two-degree-of-freedom controller (PID, PI, or PD) in your Simulink model. The PID Controller (2DOF) block allows you to implement setpoint weighting in your controller to achieve both smooth setpoint tracking and good disturbance rejection.

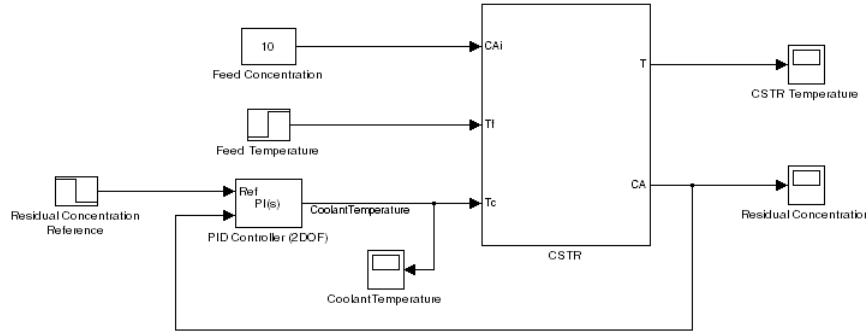
The PID Controller (2DOF) block generates an output signal based on the difference between a reference signal and a measured system output. The block computes a weighted difference signal for each of the proportional, integral, and derivative actions according to the setpoint weights you specify. The block output is the sum of the proportional, integral, and derivative actions on the respective difference signals, where each action is weighted according to the gain parameters. A first-order pole filters the derivative action. Controller gains are tunable either manually or automatically. Automatic tuning requires Simulink Control Design software (PID Tuner or SISO Design Tool).

Configurable options in the PID Controller (2DOF) block include:

- Controller type and form
- Time domain (continuous or discrete)
- Initial conditions and reset trigger
- Output saturation limits and built-in anti-windup mechanism

- Signal tracking for bumpless control transfer and multiloop control

In one common implementation, the PID Controller (2DOF) block operates in the feedforward path of the feedback loop. The block receives a reference signal at the `Ref` input and a measured system output at the other input. For example:



For a single-input block that accepts an error signal (a difference between a setpoint and a system output), see the PID Controller block reference page.

You can generate code to implement your controller using any Simulink data type, including fixed-point data types. (Code generation requires Real-Time Workshop software; fixed-point implementation requires Fixed-Point Toolbox.)

Data Type Support

The PID Controller (2DOF) block accepts real signals of any numeric data type that Simulink software supports, including fixed-point data types. See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Parameters

The following table summarizes the PID Controller (2DOF)block parameters, accessible via the block parameter dialog box.

Task	Parameters
Choose controller form and type.	• **Controller Form** in Main tab • **Controller**
Choose discrete or continuous time.	• **Time-domain** • **Sample time**
Choose an integration method (discrete time).	• **Integrator method** • **Filter method**
Set and tune controller gains.	• **Proportional (P)** in Main tab • **Integral (I)** in Main tab • **Derivative (D)** in Main tab • **Filter coefficient (N)** in Main tab • **Setpoint weight (b)** in Main tab • **Setpoint weight (c)** in Main tab
Set integrator and filter initial conditions.	• **Initial conditions Source** in Main tab • **Integrator Initial condition** in Main tab • **Filter Initial condition** in Main tab • **External reset** in Main tab • **Ignore reset when linearizing** in Main tab

Task	Parameters
Limit block output.	• **Limit output** in PID Advanced tab • **Lower saturation limit** in PID Advanced tab • **Upper saturation limit** in PID Advanced tab • **Ignore saturation when linearizing** in PID Advanced tab
Configure anti-windup mechanism (when you limit block output).	• **Anti-windup method** in PID Advanced tab • **Back-calculation gain (Kb)** in PID Advanced tab
Enable signal tracking.	• **Enable tracking mode** in PID Advanced tab • **Tracking gain (Kt)** in PID Advanced tab
Configure data types.	• **Parameter data type** in Data Type Attributes tab • **Product output data type** in Data Type Attributes tab • **Summation output data type** in Data Type Attributes tab • **Accumulator data type** in Data Type Attributes tab • **Integrator output data type** in Data Type Attributes tab • **Filter output data type** in Data Type Attributes tab

Task	Parameters
	• **Saturation output data type** in Data Type Attributes tab • **Lock output data type setting against changes by the fixed-point tools** in Data Type Attributes tab • **Saturate on integer overflow** in Data Type Attributes tab • **Integer rounding mode** in Data Type Attributes tab
Configure block for code generation.	• **State name** in State Attributes tab • **State name must resolve to Simulink signal object** in State Attributes tab • **Real-Time Workshop storage class** in State Attributes tab • **Real-Time Workshop storage type qualifier** in State Attributes tab

Controller form

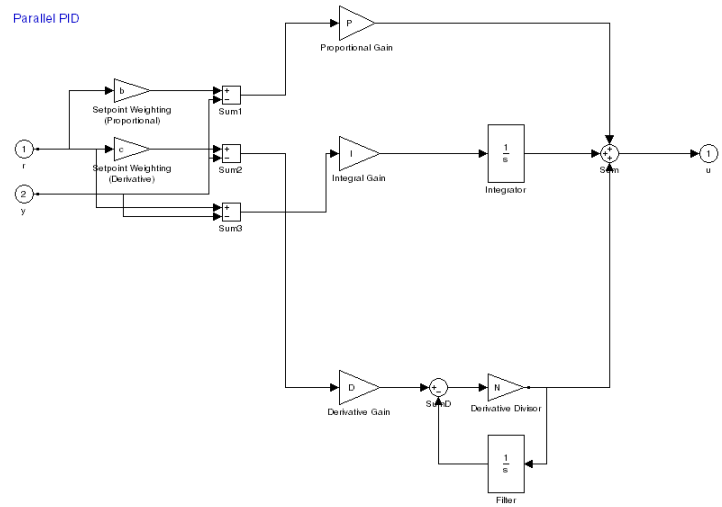
Select the controller form.

Settings

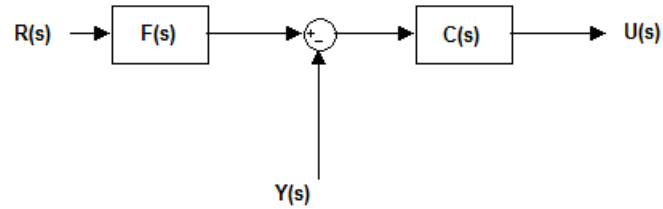
`Parallel` (Default)

Selects a controller form in which the proportional, integral, and derivative gains **P**, **I**, and **D** operate independently. The filter coefficient **N** sets the location of the pole in the derivative filter.

Parallel two-degree-of-freedom PID controller, where input 1 receives a reference signal and input 2 receives feedback from the measured system output:



The parallel two-degree-of-freedom PID controller can be equivalently modeled by the following block diagram:



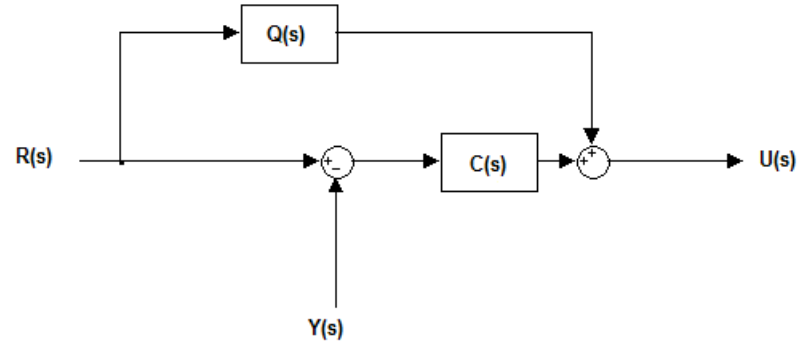
where `R(s)` represents the reference signal and `Y(s)` represents the feedback from measured system output. In this model, `C(s)` is a single degree-of-freedom controller, and `F(s)` acts as a prefilter on the reference signal. For a parallel two-degree-of-freedom PID controller in the `Continuous-time` **Time-domain**, the transfer functions `F(s)` and `C(s)` are:

$$F_{par}(s) = \frac{(bP + cDN)s^2 + (bPN + I)s + IN}{(P + DN)s^2 + (PN + I)s + IN}$$

$$C_{par}(s) = \frac{(P + DN)s^2 + (PN + I)s + IN}{s(s + N)}$$

where `b` and `c` are the **Setpoint weight** parameters.

Alternatively, the parallel two-degree-of-freedom PID controller can be modeled by the following block diagram:



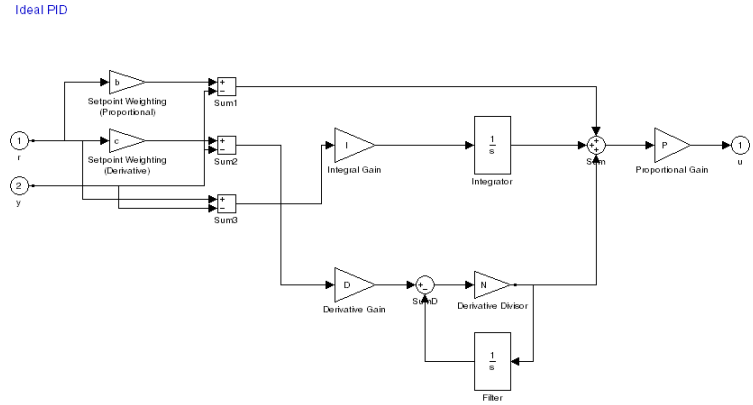
with `R(s)`, `Y(s)`, and `C(s)` as discussed previously. In this realization, In this realization, `Q(s)` acts as feed-forward conditioning on the reference signal `R(s)`. For a parallel PID controller in the `Continuous-time` **Time-domain**, the transfer function `Q(s)` is:

$$Q_{par}(s) = \frac{\left((b-1)P + (c-1)DN\right)s + (b-1)PN}{s+N}$$

`Ideal`

Selects a controller form in which the proportional gain `P` acts on the sum of all actions.

Ideal two-degree-of-freedom PID controller, where input 1 receives a reference signal and input 2 receives feedback from the measured system output:



Similarly to the parallel controller form discussed previously, the ideal two-degree-of-freedom PID controller can be modeled as a single degree-of-freedom controller `C(s)` with a prefilter `F(s)` For an ideal two-degree-of-freedom PID controller in the `Continuous-time` **Time-domain**, the transfer functions `F(s)` and `C(s)` are:

$$F_{id}(s) = \frac{(b+cDN)s^2 + (bN+I)s + IN}{(1+DN)s^2 + (N+I)s + IN}$$

$$C_{id}(s) = P\frac{(1+DN)s^2 + (N+I)s + IN}{s(s+N)}$$

where `b` and `c` are the **Setpoint weight** parameters.

Alternatively, modeling the ideal two-degree-of-freedom PID controller as a one-degree-of-freedom controller `C(s)` with feed-forward conditioning `Q(s)` on the reference signal gives, in continuous-time:

$$Q_{id}(s) = P\frac{\left((b-1)+(c-1)DN\right)s + (b-1)N}{s+N}$$

Controller

Specify the controller type.

Settings

`PID` (Default)

Implements a controller with proportional, integral, and derivative action.

`PI`

Implements a controller with proportional and integral action.

`PD`

Implements a controller with proportional and derivative action.

Time-domain

Select continuous or discrete time domain. The appearance of the block changes to reflect your selection.

Settings

`Continuous-time` (Default)

Selects the continuous-time representation.

`Discrete-time`

Selects the discrete-time representation. Selecting `Discrete-time` also allows you to specify the:

- **Sample time**, which is the discrete interval between samples.
- Discrete integration methods for the integrator and the derivative filter using the **Integrator method** and **Filter method** menus.

Integrator method

(Available only when you set **Time-domain** to `Discrete-time`.) Specify the method used to compute the integrator output. For more information about discrete-time integration methods, see the Discrete-Time Integrator block reference page.

Settings

`Forward Euler` (Default)

Selects the Forward Rectangular (left-hand) approximation.

- This method is best for small sampling times, where the Nyquist limit is large compared to the bandwidth of the controller. For larger sampling times, the `Forward Euler` method can result in instability, even when discretizing a system that is stable in continuous time.

`Backward Euler`

Selects the Backward Rectangular (right-hand) approximation.

- If you are generating code using Real-Time Workshop or using the Fixed-Point Toolbox and you activate the `Back-calculation` **Anti-windup method**, this integration method can cause algebraic loops in your controller. Algebraic loops can lead to slower performance of generated code. For more information about algebraic loops in Simulink models, see "Algebraic Loops" in the Simulink documentation.
- An advantage of the `Backward Euler` method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result.

`Trapezoidal`

Selects the Bilinear approximation.

- If you are generating code using Real-Time Workshop or using the Fixed-Point Toolbox and you activate the `Back-calculation` **Anti-windup method**, this integration method can cause algebraic loops in your controller. Algebraic loops can lead to slower performance of generated code. For

more information about algebraic loops in Simulink models, see "Algebraic Loops" in the Simulink documentation.

- An advantage of the `Trapezoidal` method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result. Of all available integration methods, the `Trapezoidal` method yields the closest match between frequency-domain properties of the discretized system and the corresponding continuous-time system.

Filter method

(Available only when you set **Time-domain** to `Discrete-time`.) Specify the method used to compute the derivative filter output. For more information about discrete-time integration methods, see the Discrete-Time Integrator block reference page.

Settings

`Forward Euler` (Default)

Selects the Forward Rectangular (left-hand) approximation.

- This method is best for small sampling times, where the Nyquist limit is large compared to the bandwidth of the controller. For larger sampling times, the `Forward Euler` method can result in instability, even when discretizing a system that is stable in continuous time.

`Backward Euler`

Selects the Backward Rectangular (right-hand) approximation.

- If you are generating code using Real-Time Workshop or using the Fixed-Point Toolbox, this filter method can cause algebraic loops in your controller. Algebraic loops can lead to slower performance of generated code. For more information about algebraic loops in Simulink models, see “Algebraic Loops” in the Simulink documentation.
- An advantage of the `Backward Euler` method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result. Any filter parameter value `N > 0` yields a stable result with this method.

`Trapezoidal`

Selects the Bilinear approximation.

- If you are generating code using Real-Time Workshop or using the Fixed-Point Toolbox, this filter method can cause algebraic loops in your controller. Algebraic loops can lead to slower performance of generated code. For more information about algebraic loops in Simulink models, see “Algebraic Loops” in the Simulink documentation.

- An advantage of the `Trapezoidal` method is that discretizing a stable continuous-time system using this method always yields a stable discrete-time result. Any filter parameter value `N > 0` yields a stable result with this method. Of all available filter methods, the `Trapezoidal` method yields the closest match between frequency-domain properties of the discretized system and the corresponding continuous-time system.

Sample time (-1 for inherited)

(Available only when you set **Time-domain** to `Discrete-time`.) Specify the discrete interval between samples.

Settings

Default: 1

By default, the block uses a discrete sample time of 1. To specify a different sample time, enter another discrete value, such as 0.1.

If you specify a value of -1, the PID Controller (2DOF) block inherits the sample time from upstream blocks. Do not enter a value of 0; to implement a continuous-time controller, select the **Time-domain** `Continuous-time`.

See "How to Specify the Sample Time" in the online documentation for more information.

Proportional (P)

Specify the proportional gain `P`.

Default: 1

Enter a finite, real gain value into the **Proportional (P)** field. Use either scalar or vector gain values. For a `parallel` PID **Controller form**, the proportional action is independent of the integral and derivative actions. For an `ideal` PID **Controller form**, the proportional action acts on the integral and derivative actions. See **Controller form** for more information about the role of `P` in the controller transfer function.

When you have Simulink Control Design software installed, you can automatically tune the controller gains using the PID Tuner or the SISO Design Tool. See "Designing Compensators" in the Simulink Control Design documentation.

Integral (I)

(Available for PID and PI controllers.) Specify the integral gain `I`.

Default: 1

Enter a finite, real gain value into the **Integral (I)** field. Use either scalar or vector gain values.

When you have Simulink Control Design software installed, you can automatically tune the controller gains using the PID Tuner or the SISO Design Tool. See “Designing Compensators” in the Simulink Control Design documentation.

Derivative (D)

(Available for PID and PD controllers.) Specify the derivative gain `D`.

Default: 0

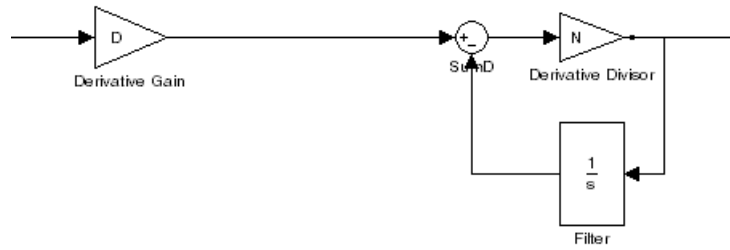
Enter a finite, real gain value into the **Derivative (D)** field. Use either scalar or vector gain values.

When you have Simulink Control Design software installed, you can automatically tune the controller gains using the PID Tuner or the SISO Design Tool. See “Designing Compensators” in the Simulink Control Design documentation.

Filter coefficient (N)

Specifies the filter coefficient of the controller.

(Available for PID and PD controllers.) Specify the filter coefficient `N`, which determines the pole location of the filter in the derivative action:



The filter pole falls at `s = -N` in the `Continuous-time` **Time-domain**. For `Discrete-time`, the location of the pole depends on which **Filter method** you select (for sampling time T_s):

- `Forward Euler`:

  $$z_{pole} = 1 - NT_s$$

- `Backward Euler`:

  $$z_{pole} = \frac{1}{1 + NT_s}$$

- `Trapezoidal`:

  $$z_{pole} = \frac{1 - NT_s/2}{1 + NT_s/2}$$

Default: 100.

Enter a finite, real gain value into the **Filter Coefficient (N)** field. Use either scalar or vector gain values. Note that the PID controller (2DOF) block does not support `N = inf` (ideal unfiltered derivative).

When you have Simulink Control Design software installed, you can automatically tune the controller gains using the PID Tuner or the SISO Design Tool. See "Designing Compensators" in the Simulink Control Design documentation. Automatic tuning requires `N > 0`.

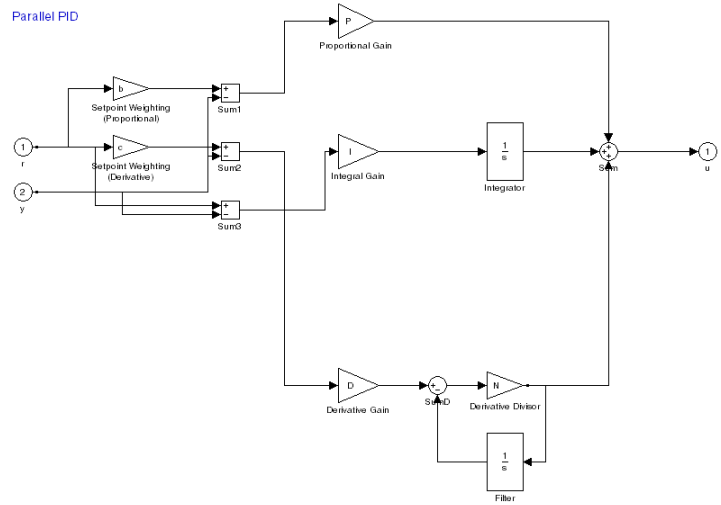
Setpoint weight (b)

Specify the proportional setpoint weight `b`.

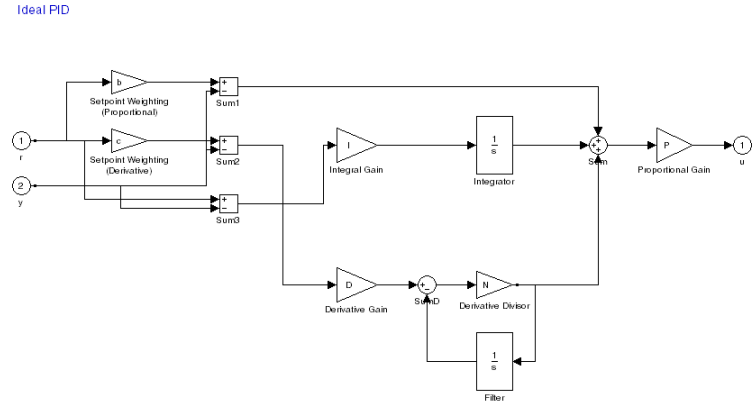
Default: 1

Enter the proportional setpoint weight value into the **Setpoint weight (b)** field. Setting `b = 0` eliminates the proportional action on the reference signal, which can reduce overshoot in the system response to step changes in the setpoint.

The following diagrams show the role of **Setpoint weight (b)** in `Parallel` and `Ideal` PID controllers. See **Controller Form** for a discussion of the corresponding transfer functions.



Parallel Two-Degree-of-Freedom PID Controller



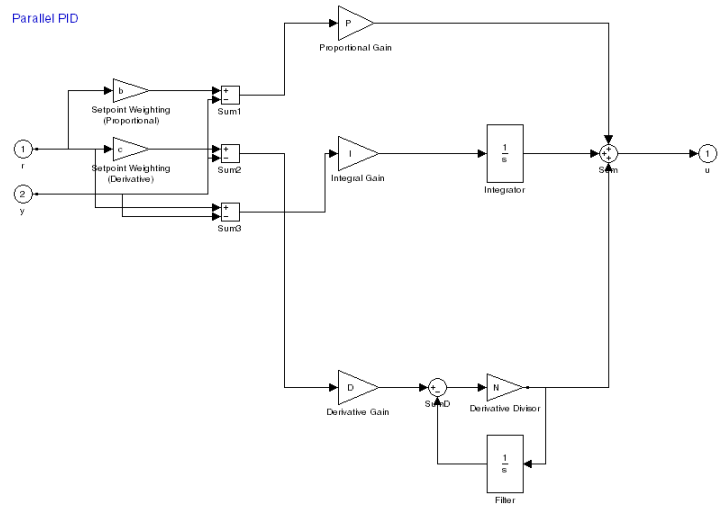
Ideal Two-Degree-of-Freedom PID Controller

Setpoint weight (c)

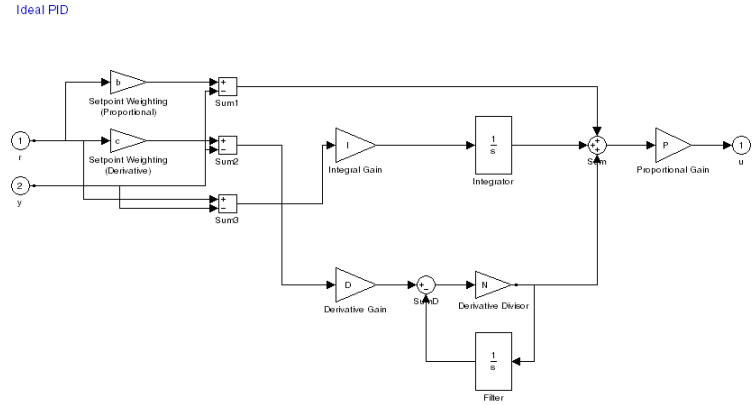
(Available for PID and PD controllers.) Specify the derivative setpoint weight `c`.

Enter the derivative setpoint weight value into the **Setpoint weight (c)** field. To implement a controller that achieves both effective disturbance rejection and smooth setpoint tracking without excessive transient response, set `c = 0`. Setting `c = 0` yields a controller with derivative action on the measured system response but not on the reference input.

The following diagrams show the role of **Setpoint weight (c)** in `Parallel` and `Ideal` PID controllers. See **Controller Form** for a discussion of the corresponding transfer functions.



Parallel Two-Degree-of-Freedom PID Controller



Ideal Two-Degree-of-Freedom PID Controller

Initial conditions Source

Select the source of the integrator and filter initial conditions. Simulink uses initial conditions to initialize the integrator and filter output at the start of a simulation or at a specified trigger event (See **External Reset**). The integrator and filter initial conditions in turn determine the initial block output.

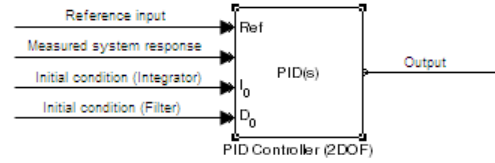
Settings

`internal` (Default)

Specifies the integrator and filter initial conditions explicitly using the **Integrator Initial condition** and **Filter Initial condition** parameters.

`external`

Specifies the integrator and filter initial conditions externally. An additional input port appears under the block inputs for each initial condition: I_0 for the integrator and D_0 for the filter:



Integrator Initial condition

(Available only when **Initial conditions Source** is `internal` and the controller includes integral action.) Specify the integrator initial value. Simulink uses the initial condition to initialize the integrator output at the start of a simulation or at a specified trigger event (see **External Reset**). The integrator initial condition, together with the filter initial condition, determines the initial output of the PID Controller (2DOF) block.

Default: 0

Simulink does not permit the integrator initial condition to be `inf` or `NaN`.

Filter Initial condition

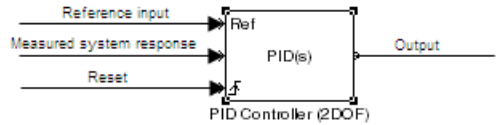
(Available only when **Initial conditions Source** is `internal` and the controller includes integral action.) Specify the filter initial value. Simulink uses the initial condition to initialize the filter output at the start of a simulation or at a specified trigger event (see **External Reset**). The filter initial condition, together with the integrator initial condition, determines the initial output of the PID Controller (2DOF) block.

Default: 0

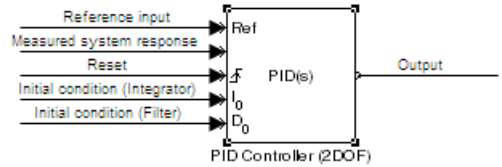
Simulink does not permit the filter initial condition to be `inf` or `NaN`.

External reset

Select the trigger event that resets the integrator and filter outputs to the initial conditions you specify in the **Integrator Initial condition** and **Filter Initial condition** fields. Selecting any option other than `none` enables a reset input on the block for the external reset signal, as shown:



Or, if the **Initial conditions Source** is `External`,



Settings

`none` (Default)
: Does not reset the integrator and filter outputs to initial conditions.

`rising`
: Resets the outputs when the reset signal has a rising edge.

`falling`
: Resets the outputs when the reset signal has a falling edge.

`either`
: Resets the outputs when the reset signal either rises or falls.

`level`

Resets and holds the outputs to the initial conditions while the reset signal is nonzero.

Note To be compliant with the Motor Industry Software Reliability Association (MISRA) software standard, your model must use Boolean signals to drive the external reset ports of the PID controller (2DOF) block.

Ignore reset when linearizing

Force Simulink linearization commands to ignore any reset mechanism that you have chosen with the **External reset** menu. Ignoring reset states allows you to linearize a model around an operating point even if that operating point causes the PID Controller (2DOF) block to reset.

Settings


`Off` (Default)

Simulink linearization commands do not ignore states corresponding to the reset mechanism.


`On`

Simulink linearization commands ignore states corresponding to the reset mechanism.

Enable zero-crossing detection

Enable zero-crossing detection in continuous-time models upon reset and upon entering or leaving a saturation state.

Zero-crossing detection can accurately locate signal discontinuities without resorting to excessively small time steps that can lead to lengthy simulation times. If you select **Limit output** or activate an **External reset** in your PID Controller (2DOF) block, activating zero-crossing detection can reduce computation time in your simulation. For more information, see "Zero-Crossing Detection" in the Simulink documentation.

Settings


`On` (Default)

Uses zero-crossing detection at any of the following events: reset; entering or leaving an upper saturation state; and entering or leaving a lower saturation state.


`Off`

Does not use zero-crossing detection.

Enabling zero-crossing detection for the PID Controller (2DOF) block also enables zero-crossing detection for all under-mask blocks that include the zero-crossing detection feature.

Limit output

Limit the block output to values you specify as the **Lower saturation limit** and **Upper saturation limit** parameters.

Activating this option limits the block output internally to the block, obviating the need for a separate Saturation block after the controller in your Simulink model. It also allows you to activate the built-in anti-windup mechanism (see **Anti-windup method**).

Settings

 `Off` (Default)

Does not limit the block output, which is the weighted sum of the proportional, integral, and derivative actions.

 `On`

Limits the block output to the **Lower saturation limit** or the **Upper saturation limit** whenever the weighted sum exceeds those limits. Allows you to select an **Anti-windup method**.

Lower saturation limit

(Available only when you select the **Limit Output** box.) Specify the lower limit for the block output. The block output is held at the **Lower saturation limit** whenever the weighted sum of the proportional, integral, and derivative actions goes below that value.

Default: `-inf`

Upper saturation limit

(Available only when you select the **Limit Output** box.) Specify the upper limit for the block output. The block output is held at the **Upper saturation limit** whenever the weighted sum of the proportional, integral, and derivative actions exceeds that value.

Default: `inf`

Anti-windup method

(Available only when you select the **Limit Output** option and the controller includes integral action.) Select an anti-windup mechanism to discharge the integrator when the block is saturated, which occurs when the sum of the block components exceeds the output limits.

When you select the **Limit output** check box and the weighted sum of the controller components exceeds the specified output limits, the block output holds at the specified limit. However, the integrator output can continue to grow (integrator wind-up), increasing the difference between the block output and the sum of the block components. Without a mechanism to prevent integrator wind-up, two results are possible:

- If the sign of the input signal never changes, the integrator continues to integrate until it overflows. The overflow value is the maximum or minimum value for the data type of the integrator output.
- If the sign of the input signal changes once the weighted sum has grown beyond the output limits, it can take a long time to discharge the integrator and return the weighted sum within the block saturation limit.

In both cases, controller performance can suffer. To combat the effects of wind-up without an anti-windup mechanism, it may be necessary to detune the controller (for example, by reducing the controller gains), resulting in a sluggish controller. Activating an anti-windup mechanism can improve controller performance.

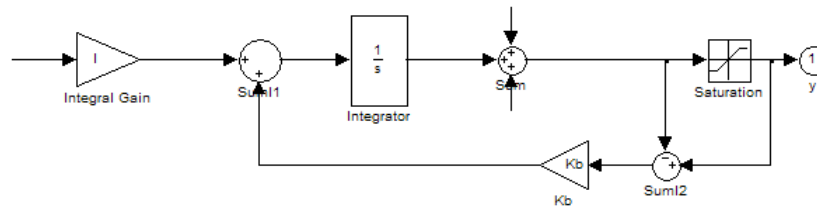
Settings

`none` (Default)

Does not use an anti-windup mechanism. This setting can cause the block's internal signals to be unbounded even if the output appears to be bounded by the saturation limits. This can result in slow recovery from saturation or unexpected overflows.

`back-calculation`

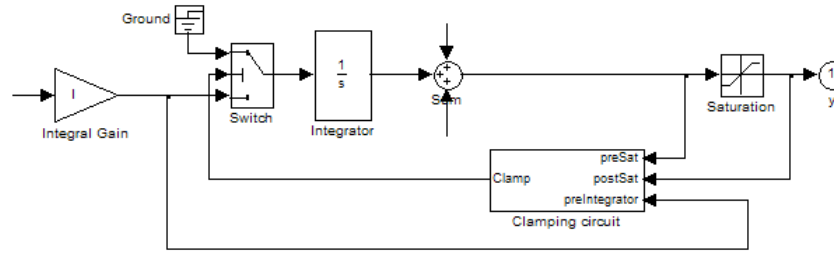
Discharges the integrator when the block output saturates using this feedback loop:



You can also specify a value for the **Back-calculation coefficient (Kb)**.

`clamping`

Stops integration when the sum of the block components exceeds the output limits and the integrator output and block input have the same sign. Resumes integration when the sum of the block components exceeds the output limits and the integrator output and block input have opposite sign. The integrator portion of the block is:



where the clamping circuit implements the logic necessary to determine whether integration continues.

Back-calculation gain (Kb)

(Available only when the `back-calculation` **Anti-windup method** is active.) Specify the gain coefficient of the anti-windup feedback loop.

The `back-calculation` anti-windup method discharges the integrator on block saturation using a feedback loop having gain coefficient `Kb`.

Default: 1

Ignore saturation when linearizing

Force Simulink linearization commands ignore PID Controller (2DOF) block output limits. Ignoring output limits allows you to linearize a model around an operating point even if that operating point causes the PID Controller (2DOF) block to exceed the output limits.

Settings

 `Off` (Default)

Simulink linearization commands do not ignore states corresponding to saturation.

 `On`

Simulink linearization commands ignore states corresponding to saturation.

Enable tracking mode

(Available for any controller with integral action.) Activate signal tracking, which lets the output of the PID Controller (2DOF) block follow a tracking signal. Provide the tracking signal to the block at the `TR` port, which becomes active when you select **Enable tracking mode**.

When signal tracking is active, the difference between the tracked signal and the block output is fed back to the integrator input with a gain `Kt`. You can also specify the value of the **Tracking coefficient (Kt)**.

For information about using tracking mode to implement bumpless control transfer scenarios and multiloop controllers, see "Enable tracking mode" on page 2-886 in the PID Controller reference page.

Settings

 `Off` (Default)

Disables signal tracking and removes `TR` block input.

 `On`

Enables signal tracking and activates `TR` input.

Tracking gain (Kt)

(Available only when you select **Enable tracking mode**.) Specify `Kt`, which is the gain of the signal tracking feedback loop.

Default: 1

Parameter data type

Select the data type of the gain parameters **P**, **I**, **D**, **N**, **Kb**, and **Kt** and the setpoint weighting parameters **b** and **c**.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Inherit via back propagation`

Use data type of the driving block.

`Inherit: Same as input`

Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Product output data type

Select the product output data type of the gain parameters **P**, **I**, **D**, **N**, **Kb**, and **Kt** and the setpoint weighting parameters **b** and **c** .

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Inherit via back propagation`

Use data type of the driving block.

`Inherit: Same as input`

Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Summation output data type

Select the summation output data type of the sums **Sum**, **Sum1**, **Sum2**, **Sum3**, **Sum D**, **Sum I1** , **SumI2** ,and **SumI3**, which are sums computed internally within the block. To see where Simulink computes each of these sums , right-click the PID Controller (2DOF) block in your model and select `Look Under Mask`:

- **Sum** is the weighted sum of the proportional, derivative, and integral signals.
- **Sum1** is the difference between the reference input weighted by **b** and the measured system response.
- **Sum2** is the difference between the reference input weighted by **c** and the measured system response.
- **Sum3** is the difference between the unweighted reference input and the measured system response.
- **SumD** is the sum in the derivative filter feedback loop.
- **SumI1** is the sum of the block input signal (weighted by the integral gain `I`) and **SumI2**. **SumI1** is computed only when **Limit output** and **Anti-windup method** `back-calculation` are active.
- **SumI2** is the difference between the weighted sum **Sum** and the limited block output. **SumI2** is computed only when **Limit output** and **Anti-windup method** `back-calculation` are active.
- **SumI3** is the difference between the block output and the signal at the block's tracking input. **SumI3** is computed only when you select the **Enable tracking mode** box.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range

and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Same as first input`
Use data type of first input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
Name of a data type object. For example, `Simulink.NumericType`.

Accumulator data type

Specify the accumulator data type.

Settings

Default: `Inherit:  Inherit via internal rule`

`Inherit:  Inherit via internal rule`
: Use internal rule to determine accumulator data type.

`Inherit:  Same as first input`
: Use data type of first input signal.

`double`
: Accumulator data type is `double`.

`single`
: Accumulator data type is `single`.

`int8`
: Accumulator data type is `int8`.

`uint8`
: Accumulator data type is `uint8`.

`int16`
: Accumulator data type is `int16`.

`uint16`
: Accumulator data type is `uint16`.

`int32`
: Accumulator data type is `int32`.

`uint32`
: Accumulator data type is `uint32`.

`fixdt(1,16,0)`
: Accumulator data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Accumulator data type is fixed point `fixdt(1,16,2^0,0)`.

`<data type expression>`
: The name of a data type object, for example `Simulink.NumericType`

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Integrator output data type

Select the data type of the integrator output.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Same as input`

Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Filter output data type

Select the data type of the filter output.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Inherit via internal rule` (Default)

Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory. This memory requirement accommodates the calculated output range and maintains the output precision of the block and 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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)` (a generic 32-bit microprocessor) is the specified target hardware, the output data type is `int32`.

`Inherit: Same as input`

Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Saturation output data type

Select the saturation output data type.

See "Data Types Supported by Simulink" in the Simulink documentation for more information.

Settings

`Inherit: Same as input` (Default)
: Use data type of input signal.

`double`

`single`

`int8`

`uint8`

`int16`

`uint16`

`int32`

`uint32`

`fixdt(1,16)`

`fixdt(1,16,0)`

`fixdt(1,16,2^0,0)`

`<data type expression>`
: Name of a data type object. For example, `Simulink.NumericType`.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Saturate on integer overflow

Specify whether overflows saturate.

Settings

Default: Off

 On

Overflows saturate.

 Off

Overflows do not saturate.

Tips

- When you select this check box, saturation applies to every internal operation on the block, not just the output or result.
- In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Floor`

`Ceiling`
: Round both positive and negative numbers toward positive infinity.

`Convergent`
: Round number to the nearest representable value. If a tie occurs, round to the nearest even stored value.

`Floor`
: Round both positive and negative numbers toward negative infinity.

`Nearest`
: Round number to the nearest representable value. If a tie occurs, round toward positive infinity.

`Round`
: Round number to the nearest representable value. If a tie occurs, round positive numbers toward positive infinity and round negative numbers toward negative infinity.

`Simplest`
: This option provides for an optimization of the rounding code for several blocks.

`Zero`
: Round number toward zero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Rounding” in the *Simulink Fixed Point User's Guide*.

State name

Assign unique name to each state. The state names apply only to the selected block.

To assign a name to a single state, enter the name between quotes; for example, `'velocity'`.

To assign names to multiple states, enter a comma-delimited list surrounded by braces; for example, `{'a', 'b', 'c'}`. Each name must be unique. To assign state names with a variable that has been defined in the MATLAB workspace, enter the variable without quotes. The variable can be a string, cell, or structure.

Settings

Default: `' '` (no name)

State name must resolve to Simulink signal object

Require that state name resolve to Simulink signal object.

Settings

Default: Off

 On

Require that state name resolve to Simulink signal object.

 Off

Do not require that state name resolve to Simulink signal object.

Dependencies

State name enables this parameter.

This parameter enables **Real-Time Workshop storage class**.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Real-Time Workshop storage class

Select state storage class.

Settings

Default: `Auto`

`Auto`
: Auto is the appropriate storage class for states that you do not need to interface to external code.

`ExportedGlobal`
: State is stored in a global variable

`ImportedExtern`
: *`model`*`_private.h` declares the state as an extern variable.

`ImportedExternPointer`
: *`model`*`_private.h` declares the state as an extern pointer.

Dependencies

State name enables this parameter.

Setting this parameter to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables **Real-Time Workshop storage type qualifier**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Block State Storage Classes" in the *Real-Time Workshop User's Guide*.

Real-Time Workshop storage type qualifier

Specify Real-Time storage type qualifier.

Settings

Default: `' '`

If left blank, no qualifier is assigned.

Dependencies

Setting **Real-Time Workshop storage class** to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Direct Feedthrough	The following ports support direct feedthrough: • Reset port • Integrator and filter initial condition port • Input port, for every integration method except Forward Euler
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Supported for gain parameters **P**, **I**, and **D** for filter coefficient **N**, and for setpoint weights **b** and **c**
States	Inherited from driving block and parameters
Dimensionalized	Yes
Zero-Crossing Detection	Yes (in continuous-time domain)

See Also

PID Controller, Gain, Integrator, Discrete-Time Integrator, Derivative, Discrete Derivative.

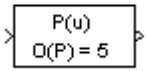
Purpose

Perform evaluation of polynomial coefficients on input values

Library

Math Operations

Description



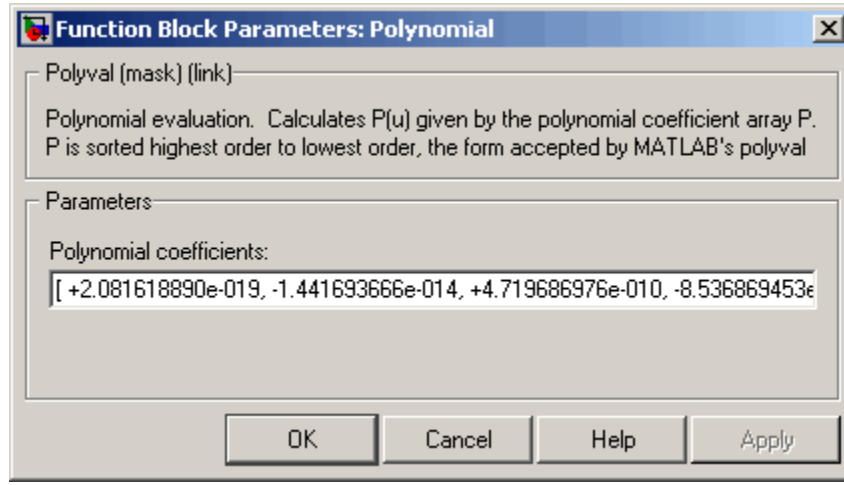
The Polynomial block uses a coefficients parameter to evaluate a real polynomial for the input value.

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

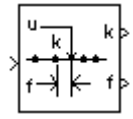
Purpose

Compute index and fraction for Interpolation Using Prelookup block

Library

Lookup Tables

Description



Comparison with the Lookup Table (n-D) Block

The Prelookup block works best with the Interpolation Using Prelookup block. The Prelookup block calculates the index and interval fraction that specifies how its input value relates to the breakpoint data set. You feed the resulting index and fraction values into an Interpolation Using Prelookup block to interpolate an *n*-dimensional table. This combination of blocks performs the same operation that a single instance of the Lookup Table (n-D) block performs. However, the Prelookup and Interpolation Using Prelookup blocks offer greater flexibility that can provide more efficient simulation and code generation.

How the Block Works with an Interpolation Using Prelookup Block

To use this block, you must define a set of breakpoint values. Typically, this breakpoint data set corresponds to one dimension of the **Table data** parameter in an Interpolation 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 forms an interval containing the input
- The resulting fractional value that is a number $0 \le f < 1$, representing the input value's normalized position on the breakpoint interval between the index and the next index value for in-range input

For example, if the breakpoint data set is

```
[ 0 5 10 20 50 100 ]
```

and the input value `u` is 55, the index is 4 and the fractional value is 0.1, denoted respectively as `k` and `f` on the block. 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 **Process out-of-range input** block parameter for more information.

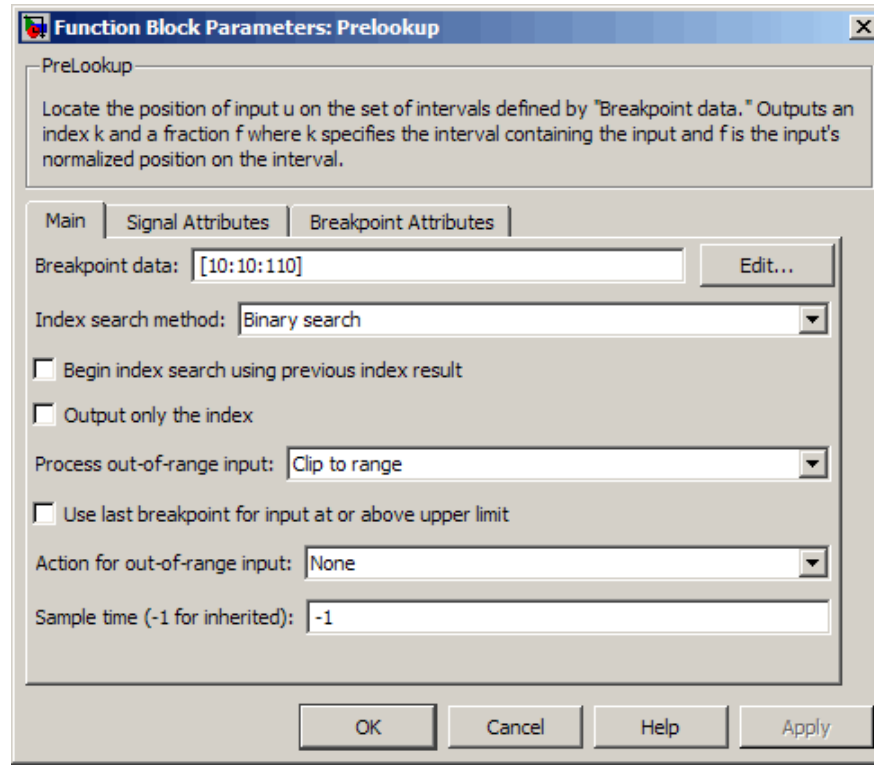
Data Type Support

The Prelookup block accepts real signals of any numeric data type supported by Simulink software, except `Boolean`. The Prelookup block supports fixed-point data types for signals and breakpoint data.

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

Parameters and Dialog Box

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



Breakpoint data

The set of numbers to search. Specify a strictly monotonically increasing vector that contains two or more elements.

Click the **Edit** button to open the Lookup Table Editor (see "Lookup Table Editor" in the Simulink documentation).

Tip For information on how to define evenly-spaced breakpoints, see "Formulation of Evenly-Spaced Breakpoints" in the Simulink documentation.

Index search method

Select `Evenly spaced points`, `Linear search`, or `Binary search` (the default). Each search method has speed advantages in different circumstances:

- For evenly spaced breakpoint sets (for example, 10, 20, 30, and so on), you achieve optimal speed by selecting `Evenly spaced points` to calculate table indices.

  This algorithm uses only the first two breakpoints of a set to determine the offset and spacing of the remaining points.

- For unevenly spaced breakpoint sets, follow these guidelines:

  - If input signals do not vary much between time steps, selecting `Linear search` with **Begin index search using previous index result** produces the best performance.

  - If input signals jump more than one or two table intervals per time step, selecting `Binary search` produces the best performance.

A suboptimal choice of index search method can lead to slow performance of models that rely heavily on lookup tables.

Note Real-Time Workshop generated code stores only the first breakpoint, the spacing, and the number of breakpoints when:

- The breakpoint data is not tunable.
- The index search method is `Evenly spaced points`.

Begin index search using previous index result

Select this check box when you want the block to start its search using the index found at the previous time step. For inputs that change slowly with respect to the interval size, enabling this option can improve performance. Otherwise, the linear search and binary search methods can take longer, especially for large breakpoint sets.

Output only the index

Select this check box when you want the block to output only the resulting index value.

Typical applications include:

- Feeding a Direct Lookup Table (n-D) block, with no interpolation on the interval
- Feeding selection ports of a sub-table selection for an Interpolation Using Prelookup block
- Performing nonlinear quantizations

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 (for example, `0`) and `0` for the interval fraction. If the input is greater than the last breakpoint, return the index of the next-to-last breakpoint and `1` for the interval fraction.

  Suppose the range is `[1 2 3]` and you select this option. If the input is `0.5`, the index is `0` and the interval fraction is `0`; if the input is `3.5`, the index is `1` and the interval fraction is `1`.

- `Linear extrapolation`

  If the input is less than the first breakpoint, return the index of the first breakpoint (for example, `0`) 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-last breakpoint and an interval fraction that represents the linear distance from the next-to-last breakpoint to the input.

Suppose the range is `[1 2 3]` and you select this option. If the input is `0.5`, the index is `0` and the interval fraction is `-0.5`; if the input is `3.5`, the index is `1` and the interval fraction is `1.5`.

The Prelookup block supports `Linear extrapolation` only if all of these conditions apply:

- The block input, breakpoint data, and fraction output specify floating-point data types.
- The data type of its index specifies a built-in integer.

Use last breakpoint for input at or above upper limit

Specifies how to index inputs that are greater than or equal to the last breakpoint. The index value is zero-based.

If this check box is...	The block returns these values when the input equals the last breakpoint...
Selected	• Index of the last element in the breakpoint data set • Interval fraction of `0`
Not selected	• Index of the next-to-last breakpoint • Interval fraction of `1`

This check box is visible only when:

- **Output only the index** is cleared.
- **Process out-of-range input** is `Clip to range`.

However, if **Output only the index** is selected and **Process out-of-range input** is `Clip to range`, the block behaves as if this check box is selected even though it is invisible.

Tip If you select **Use last breakpoint for input at or above upper limit** for a Prelookup block, you must also select **Valid index input may reach last index** for the Interpolation Using Prelookup block to which it connects. This action allows the blocks to use the same indexing convention when accessing the last elements of their **Breakpoint data** and **Table data** parameters.

Action for out-of-range input

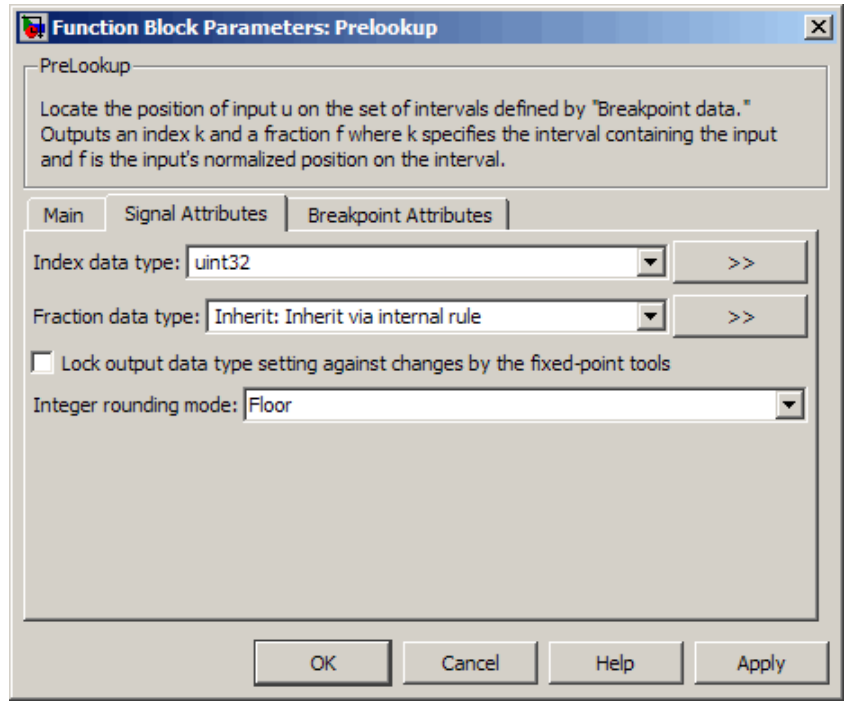
Specifies whether to produce a warning or error message if the input is out of range. Options include:

- `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

Sample time

Specifies the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in the Simulink User's Guide for more information.

The **Signal Attributes** pane of the Prelookup block dialog box appears as follows:



Index data type

Specify how the data type of the index is designated. You can choose a built-in integer data type from the list, or you can specify an integer data type using a fixed-point representation. The data type that you specify must be capable of indexing all elements in the **Breakpoint data** parameter.

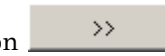
Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Index data type** parameter.

See “Using the Data Type Assistant” in the *Simulink User’s Guide* for more information.

Fraction data type

Specify how the data type of the interval fraction is designated. You can choose a built-in data type from the list, specify that the data type is inherited through an internal rule, or specify a fixed-point data type using either the [Slope Bias] or the binary-point-only scaling representation. If using the [Slope Bias] representation, the scaling must be trivial — i.e., the slope is `1` and the bias is `0`. If using the binary-point-only representation, the fixed power-of-two exponent must be less than or equal to zero.

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Fraction data type** parameter.

See “Using the Data Type Assistant” in the *Simulink User’s Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

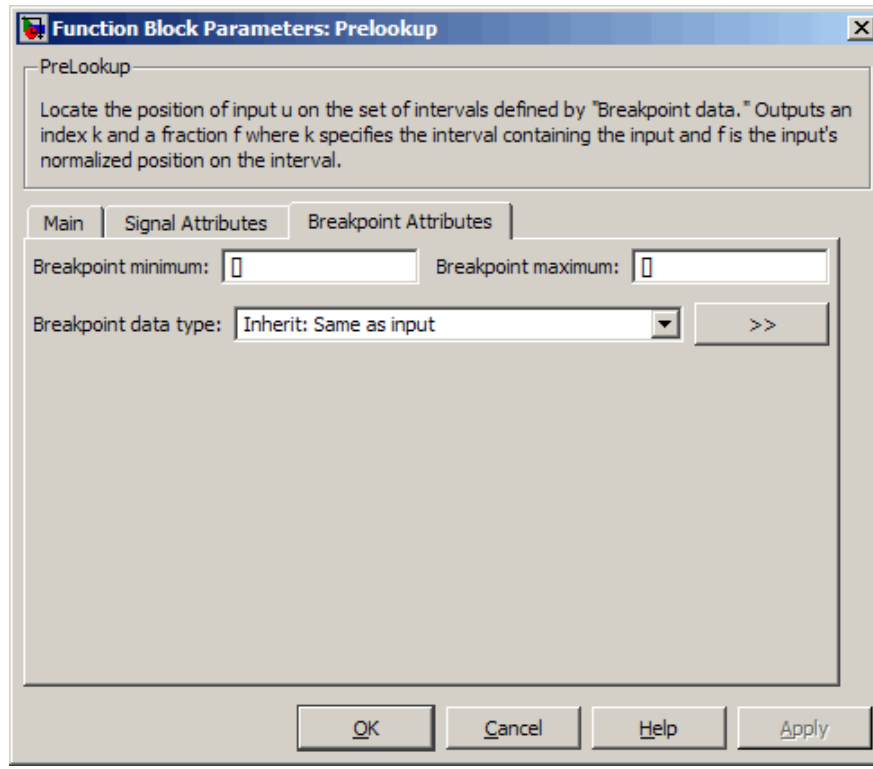
Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see “Rounding” in the *Simulink Fixed Point User’s Guide*.

Block parameters such as **Breakpoint data** are always rounded to the nearest representable value. To control the rounding of a block parameter, enter an expression using a MATLAB rounding function into the mask field.

The **Breakpoint Attributes** pane of the Prelookup block dialog box appears as follows:



Breakpoint minimum

Specify the minimum value that the breakpoint data can have. The default value, `[]`, is equivalent to `-Inf`.

Breakpoint maximum

Specify the maximum value that the breakpoint data can have. The default value, `[]`, is equivalent to `Inf`.

Breakpoint data type

Specify the breakpoint data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as input`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Breakpoint data type** parameter.

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Tip Specify a breakpoint data type different from the input data type for these cases:

- Lower memory requirement for storing breakpoint data that uses a smaller type than the input signal
- Sharing of prescaled breakpoint data between two Prelookup blocks with different input data types
- Sharing of custom storage breakpoint data in Real-Time Workshop generated code for blocks with different input data types

---

Examples

See "Examples for Prelookup and Interpolation Blocks" in the Simulink documentation.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes
Dimensionalized	Yes
Zero-Crossing Detection	No

See Also

Interpolation Using Prelookup

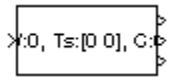
Purpose

Output 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

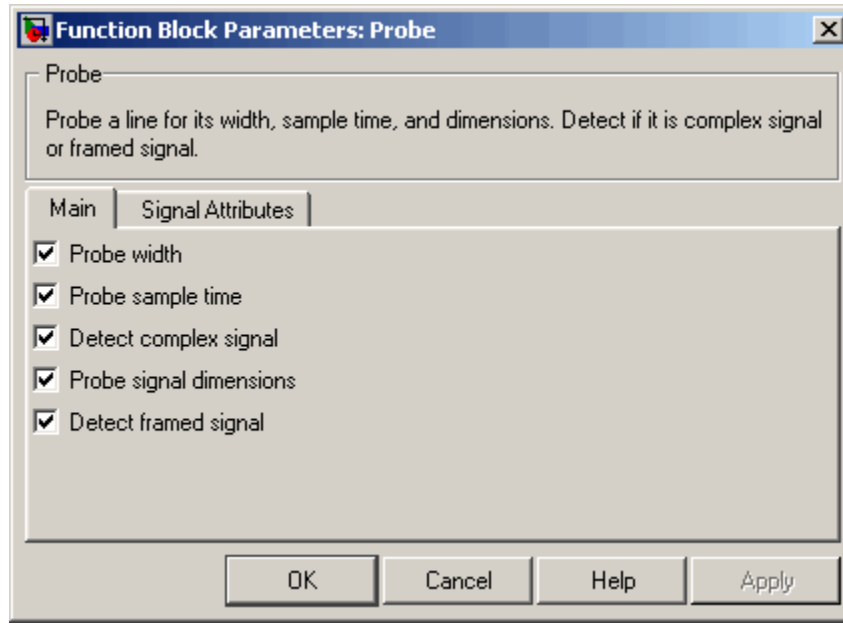
The Probe block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

For more information, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

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



Probe width
: Select to output the width, or number of elements, of the probed signal.

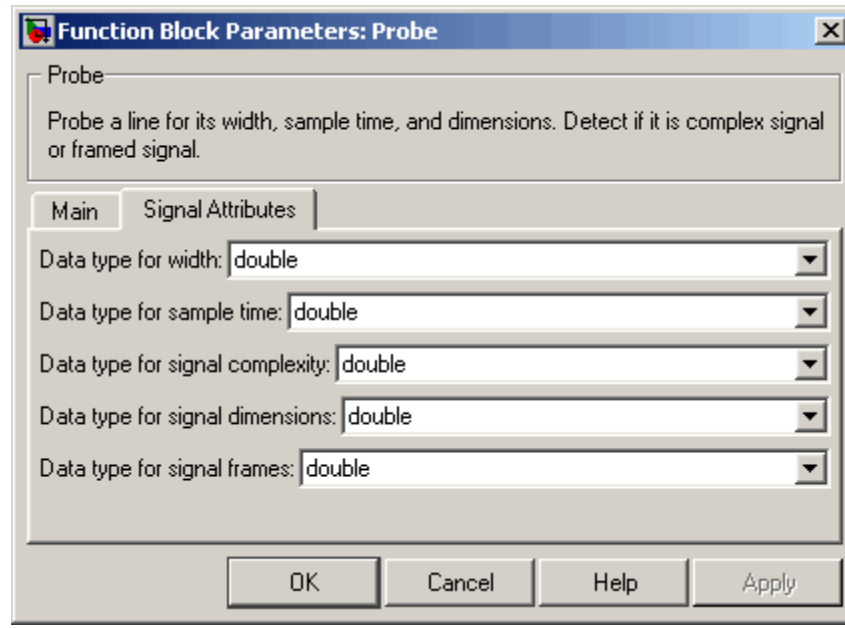
Probe sample time
: Select to output the sample time of the probed signal. The output is a two-element vector that specifies the period and offset of the sample time, respectively. See "How to Specify the Sample Time" for more information.

Detect 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 Attributes** pane of the Probe block dialog box appears as follows:



Note The Probe block ignores the **Data type override** setting of the Fixed-Point Tool.

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
Multidimensionalized	Yes
Zero-Crossing Detection	No

Purpose

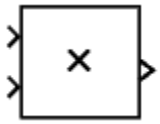
Multiply and divide scalars and nonscalars or multiply and invert matrices

Library

Math Operations

Description

Default Product Block Use



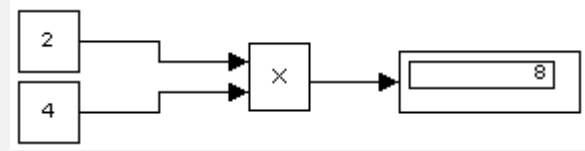
By default, the Product block outputs the result of multiplying two inputs: two scalars, a scalar and a nonscalar, or two nonscalars that have the same dimensions. The default parameter values that specify this behavior are:

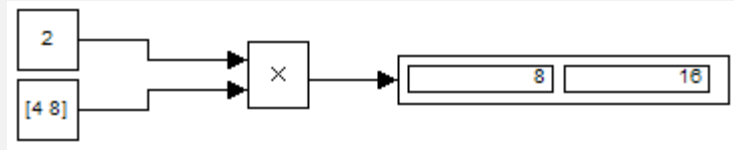
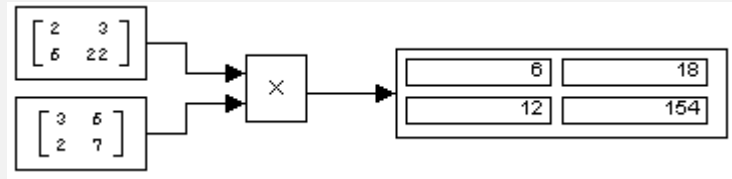
- **Multiplication:** `Element-wise(.*)`
- **Number of inputs:** `2`

Setting non-default values for either of these parameters can change a Product block to function as a Divide block or a Product of Elements block. See the documentation of those two blocks for more information.

If all you need is to multiply two inputs to create an output, you can use the Product block with default parameter values. If you need additional capabilities, see “Product Block Capabilities” on page 2-990.

The following table shows the output of the Product block for some typical inputs using default block parameter values.

Inputs and Behavior	Example
Scalar X Scalar Output the product of the two inputs.	

Inputs and Behavior	Example
Scalar X Nonscalar Output a nonscalar having the same dimensions as the input nonscalar. Each element of the output nonscalar is the product of the input scalar and the corresponding element of the input nonscalar.	
Nonscalar X Nonscalar Output a nonscalar having the same dimensions as the inputs. Each element of the output is the product of corresponding elements of the inputs.	

Product Block Capabilities

The Product block, the Divide block, and the Product of Elements block are actually the same underlying block with different default values for the **Number of inputs** parameter. All three blocks can therefore provide the same capabilities. The Product block (or the Divide block or Product of Elements block if appropriately configured) can:

- Numerically multiply and divide any number of scalar, vector, or matrix inputs
- Perform matrix multiplication and division on any number of matrix inputs

The Product block performs scalar or matrix multiplication, depending on the value of the **Multiplication** parameter. The block accepts one or more inputs, depending on the **Number of inputs** parameter. The

Number of inputs parameter also specifies the operation to perform on each input.

The Product block can input any combination of scalars, vectors, and matrices for which the operation to perform has a mathematically defined result. The block performs the specified operations on the inputs, then outputs the result.

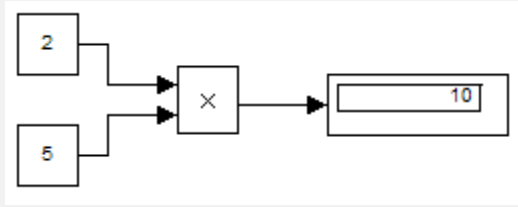
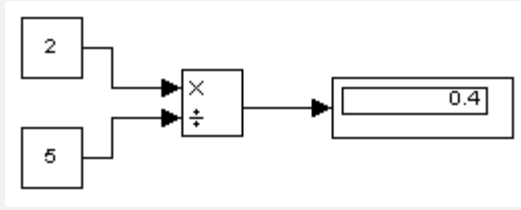
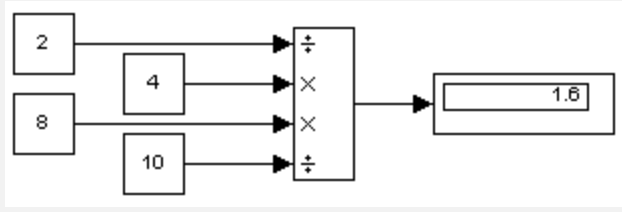
The Product block has two modes: *Element-wise mode*, which processes nonscalar inputs element by element, and *Matrix mode*, which processes nonscalar inputs as matrices. The next two sections describe these two modes.

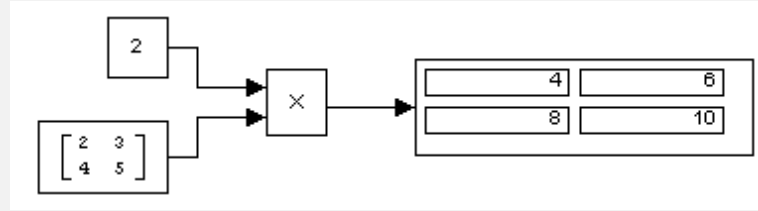
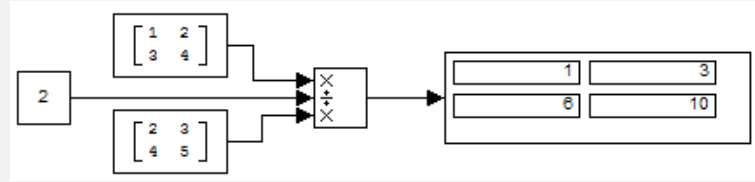
Element-wise Mode

When the value of the **Multiplication** parameter is `Element-wise(.*)`, the Product block is in *Element-wise mode*, in which it operates on the individual numeric elements of any nonscalar inputs. The MATLAB equivalent is the `.*` operator. In element-wise mode, the Product block can perform a variety of multiplication, division, and arithmetic inversion operations.

The value of the **Number of inputs** parameter controls both how many inputs exist and whether each is multiplied or divided to form the output. When the Product block in Element-wise mode has only one input, it is functionally equivalent to a Product of Elements block. When the block has multiple inputs, any nonscalar inputs must have identical dimensions, and the block outputs a nonscalar with those dimensions. To calculate the output, the block first expands any scalar input to a nonscalar that has the same dimensions as the nonscalar inputs.

This table shows the output of the Product block for some typical inputs, using the indicated values for the **Number of inputs** parameter.

Parameter Values	Examples
Number of inputs: 2	
Number of inputs: */	
Number of inputs: /** /	

Parameter Values	Examples
Number of inputs: **	
Number of inputs: */*	

Matrix Mode

When the value of the **Multiplication** parameter is `Matrix(*)`, the Product block is in *Matrix mode*, in which it processes nonscalar inputs as matrices. The MATLAB equivalent is the `*` operator. In Matrix mode, the Product block can invert a single square matrix, or multiply and divide any number of matrices that have dimensions for which the result is mathematically defined.

The value of the **Number of inputs** parameter controls both how many inputs exist and whether each input matrix is multiplied or divided to form the output. The syntax of **Number of inputs** is the same as in Element-wise mode. The difference between the modes is in the type of multiplication and division that occur.

Data Type Support

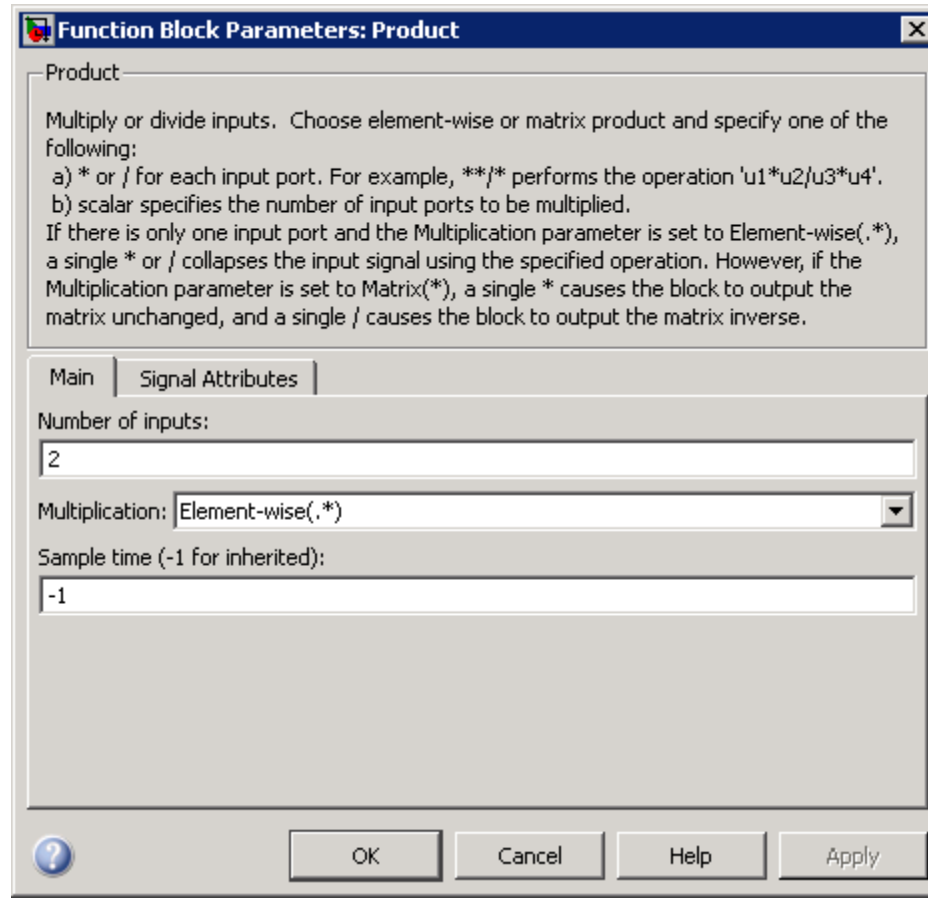
The Product block accepts real or complex signals of any numeric data type supported by Simulink software including fixed-point data types. For a discussion of the data types supported by Simulink software, see "Data Types Supported by Simulink".

The Product block does not support numeric division for complex signals with `boolean` or fixed-point data types. For other types, the block accepts complex signals as divisors only when the input and output signals all specify the same built-in data type. In this case, however, the block ignores its specified rounding mode.

The Product block accepts multidimensional signals when operating in Element-wise mode, but not when operating in Matrix mode. See "Signal Dimensions", "Element-wise Mode" on page 2-991, and "Matrix Mode" on page 2-993 for more information.

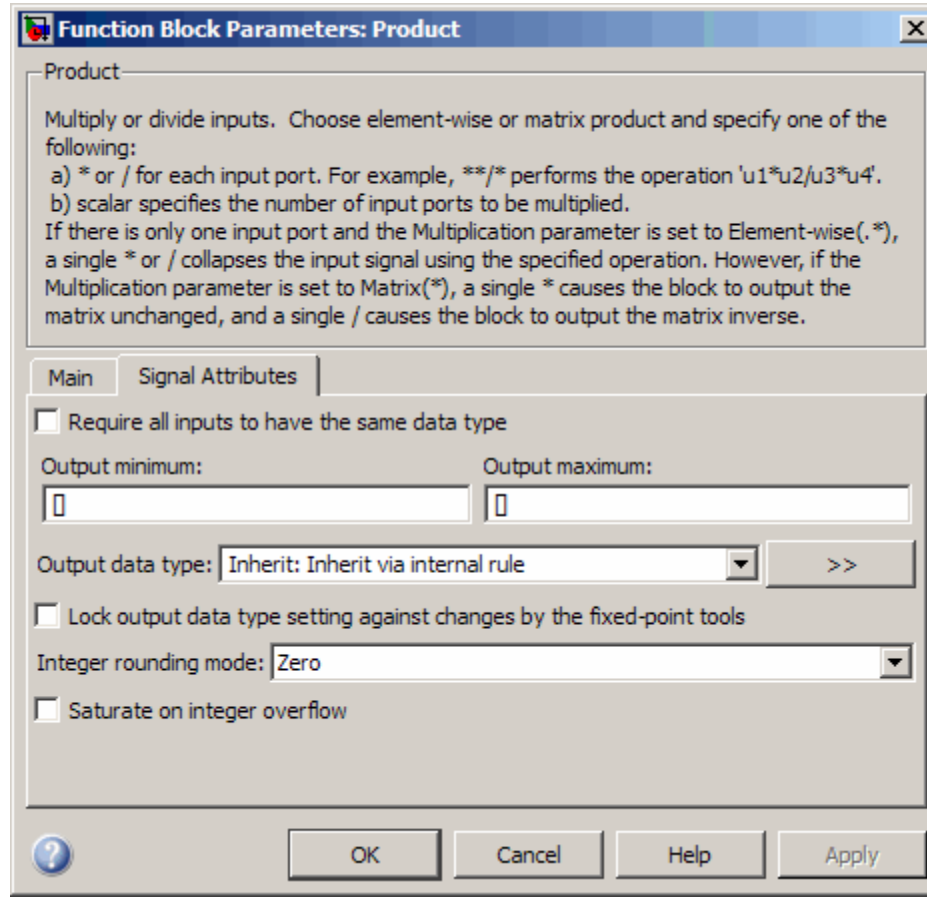
Parameters and Dialog Box

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



Signal Attributes Pane

The **Signal Attributes** pane of the Product block dialog box appears as follows:


Function Block Parameters: Product
Product
Multiply or divide inputs. Choose element-wise or matrix product and specify one of the following:
a) * or / for each input port. For example, **/* performs the operation 'u1*u2/u3*u4'.
b) scalar specifies the number of input ports to be multiplied.
If there is only one input port and the Multiplication parameter is set to Element-wise(.*), a single * or / collapses the input signal using the specified operation. However, if the Multiplication parameter is set to Matrix(*), a single * causes the block to output the matrix unchanged, and a single / causes the block to output the matrix inverse.
Main
Signal Attributes
Require all inputs to have the same data type
Output minimum:
Output maximum:
Output data type:
Inherit: Inherit via internal rule
>>
Lock output data type setting against changes by the fixed-point tools
Integer rounding mode:
Zero
Saturate on integer overflow
OK
Cancel
Help
Apply

Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Number of inputs

Control two properties of the Product block:

- The number of input ports on the block
- Whether each input is multiplied or divided into the output

Settings

Default:2

- **1 or * or /**

  Has one input. In Element-wise mode, the input is processed as described for the Product of Elements block. In Matrix mode, if the parameter value is 1 or * the block outputs the input value. If the value is / the input must be a square matrix (including a scalar as a degenerate case) and the block outputs the matrix inverse. See "Element-wise Mode" on page 2-991 and "Matrix Mode" on page 2-993 for more information.

- **An integer value > 1**

  Has number of inputs given by the integer value. The inputs are multiplied together in Element-wise mode or Matrix mode, as specified by the **Multiplication** parameter. See "Element-wise Mode" on page 2-991 and "Matrix Mode" on page 2-993 for more information.

- **An unquoted string of two or more * and / characters**

  Has the number of inputs given by the length of the string. Each input that corresponds to a * character is multiplied into the output. Each input that corresponds to a / character is divided into the output. The operations occur in Element-wise mode or Matrix mode, as specified by the **Multiplication** parameter. See "Element-wise Mode" on page 2-991 and "Matrix Mode" on page 2-993 for more information.

Dependency

Setting **Number of inputs** to `*` and selecting `Element-wise(.*)` for **Multiplication** enable the following parameter:

- **Multiply over**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Multiplication

Specify whether the Product block operates in Element-wise mode or Matrix mode.

Settings

Default: `Element-wise(.*)`

`Element-wise(.*)`
: Operate in Element-wise mode.

`Matrix(*)`
: Operate in Matrix mode.

Dependency

Selecting `Element-wise(.*)` and setting **Number of inputs** to `*` enable the following parameter:

- **Multiply over**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Multiply over

Affect multiplication on matrix input.

Settings

Default: `All dimensions`

`All dimensions`
: Output a scalar that is product of all elements of the matrix, or the product of their inverses, depending on the value of **Number of inputs**.

`Specified dimension`
: Output a vector, the composition of which depends on the value of the **Dimension** parameter.

Dependencies

- Enable this parameter by selecting `Element-wise(.*)` for **Multiplication** and setting **Number of inputs** to `*` or `1` or `/`.
- Setting this parameter to `Specified dimension` enables the **Dimension** parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Dimension

Affect multiplication on matrix input.

Settings

Default: 1

Minimum: 1

Maximum: 2

1
: Output a vector that contains an element for each column of the input matrix.

2
: Output a vector that contains an element for each row of the input matrix.

Tips

Each element of the output vector contains the product of all elements in the corresponding column or row of the input matrix, or the product of the inverses of those elements, depending on the value of **Number of inputs**:

- 1 or *

  Multiply the values of the column or row elements

- /

  Multiply the inverses of the column or row elements

Dependency

Enable this parameter by selecting `Specified dimension` for **Multiply over**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits or specify another appropriate sample time such as continuous or inherited.

Settings

Default: -1

By default, the block inherits its sample time based upon the context of the block within the model. To set a different sample time, enter a valid sample time based upon the table in "Types of Sample Time".

See also "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Require all inputs to have the same data type

Require that all inputs have the same data type.

Settings

Default: Off

 On

Require that all inputs have the same data type.

 Off

Do not require that all inputs have the same data type.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Zero`

`Ceiling`
: Round both positive and negative numbers toward positive infinity.

`Convergent`
: Round number to the nearest representable value. If a tie occurs, round to the nearest even stored value.

`Floor`
: Round both positive and negative numbers toward negative infinity.

`Nearest`
: Round number to the nearest representable value. If a tie occurs, round toward positive infinity.

`Round`
: Round number to the nearest representable value. If a tie occurs, round positive numbers toward positive infinity and round negative numbers toward negative infinity.

`Simplest`
: This option provides for an optimization of the rounding code for several blocks.

`Zero`
: Round number toward zero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Rounding” in the *Simulink Fixed Point User’s Guide*.

Saturate on integer overflow

Specify whether overflows saturate.

Settings

Default: Off

 On

Overflows saturate.

 Off

Overflows do not saturate.

Tips

- When you select this check box, saturation applies to every internal operation on the block, not just the output or result.
- In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `Inherit: Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block), `Inherit: Inherit from 'Constant value'` (Constant block), `Inherit: Inherit via back propagation` (Data Type Conversion block), `Inherit: Same as input` (Saturation block)

`Inherit: Inherit via internal rule`
: This option appears for some blocks. Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the calculated 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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 software displays an error message in the Simulation Diagnostics Viewer.

`Inherit: Inherit from 'Constant value'`
: Use data type of **Constant value** parameter. This option appears for some blocks.

`Inherit: Inherit via back propagation`
: Use data type of the driving block.

`Inherit: Same as input`
: Use data type of sole input signal. This option appears for some blocks.

`Inherit: Same as first input`
: Use data type of first input signal. This option appears for some blocks.

`Inherit: Same as accumulator`
: Output data type is the same as accumulator data type. This option appears for some blocks.

`double`
: Output data type is `double`.

`single`
: Output data type is `single`.

`int8`
: Output data type is `int8`.

`uint8`
: Output data type is `uint8`.

`int16`
: Output data type is `int16`.

`uint16`
: Output data type is `uint16`.

`int32`
: Output data type is `int32`.

`uint32`
: Output data type is `unt32`.

`fixdt(1,16,0)`
: Output data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Output data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
: Use an enumerated data type, for example, `Enum: BasicColors`. This option appears for some blocks.

`<data type expression>`
: Use a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<class name>`

`Expression`
: Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

See “Specifying a Fixed-Point Data Type” in the *Simulink User’s Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes
Dimensionalized	Yes
Multidimensionalized	Yes, only when the **Multiplication** parameter specifies `Element-wise(.*)`
Zero Crossing	No

See Also

- Divide
- Dot Product
- Product of Elements

Product of Elements

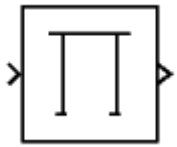
Purpose

Copy or invert one scalar input, or collapse one nonscalar input

Library

Math Operations

Description



The Product of Elements block inputs one scalar, vector, or matrix. You can use the block to:

- Copy a scalar input unchanged
- Invert a scalar input (divide 1 by it)
- Collapse a vector or matrix to a scalar by multiplying together all elements or their inverses
- Collapse a matrix to a vector by multiplying together the elements or element inverses of each row or column

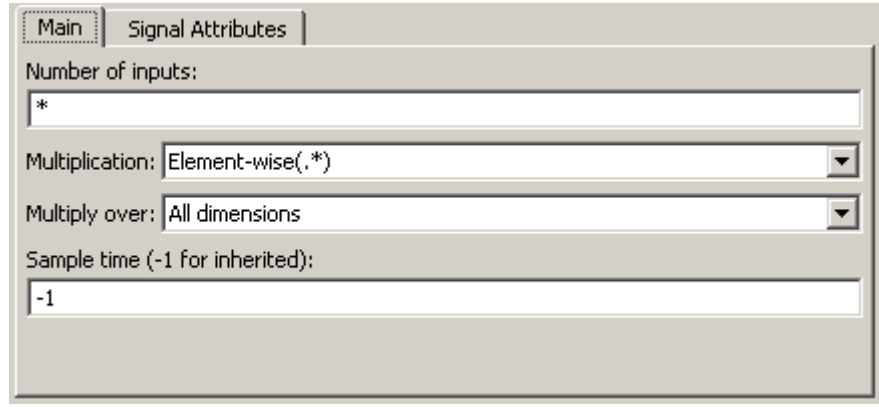
The Product of Elements block is functionally a Product block that has two preset parameter values:

- **Multiplication:** `Element-wise(.*)`
- **Number of inputs:** `*`

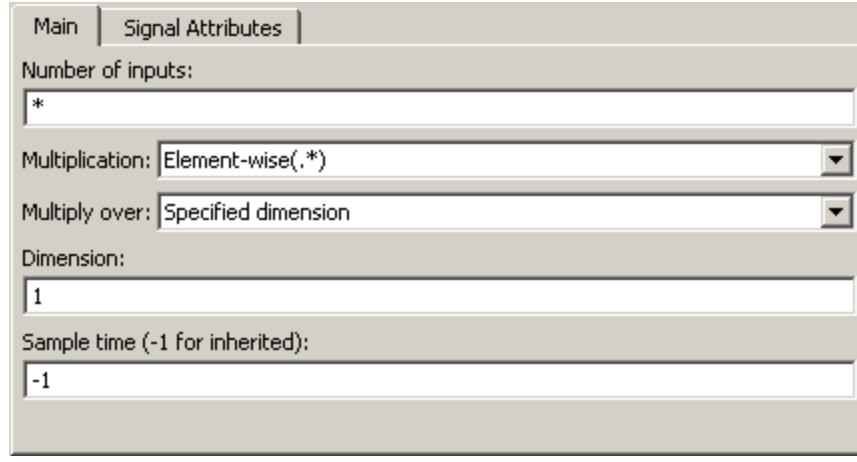
Setting non-default values for either of those parameters can change a Product of Elements block to be functionally equivalent to a Product block or a Divide block. See the documentation of those two blocks for more information.

Parameters and Dialog Box

The Product of Elements block has the same parameters and dialog box as the Product block, plus the parameter **Multiply over**, which has the default value `All dimensions`:



If you set **Multiply over** to `Specified dimension`, the **Dimension** parameter appears.



To copy, invert, or collapse one input to create an output, you can use the Product of Elements block with default values for all parameters

except **Number of inputs**, **Multiply over**, and **Dimension**. These values can require change. For other capabilities, see the Product block documentation, which also describes the "Signal Attributes Pane" on page 2-995 of the Product of Elements block.

Number of inputs

This parameter is the same as in the Product block, but the value must be `*` (the default), `1`, or `/` to achieve the behavior of a Product of Elements block.

Parameter Value	Block Behavior	Block Icon
`*` or `1`	• Copies a scalar input unchanged • Collapses a vector input to a scalar by multiplying all elements together • Collapses a matrix input to a scalar or vector by multiplying elements together based on the **Multiply over** parameter	
`/`	• Outputs the arithmetic inverse of a scalar input • Collapses a vector input to a scalar by multiplying the inverses of all elements together • Collapses a matrix input to a scalar or vector by multiplying the inverses of elements together based on the **Multiply over** parameter	

Multiply over

This parameter appears only when **Multiplication** is `Element-wise(.*)` and **Number of inputs** is `*`, `1`, or `/`. The parameter affects only a matrix input. The possible values are:

`All dimensions` — The block outputs a scalar that is the product of all matrix elements, or the product of their inverses, depending on the value of **Number of inputs**.

`Specified dimension` — The block outputs a vector, in which the composition depends on the value of the **Dimension** parameter.

Dimension

This parameter appears only when the **Multiply over** parameter appears and is set to `Specified dimension`. The parameter affects only a matrix input, and must be 1 for a scalar or vector input. The possible values are:

`1` — Output a vector that contains an element for each column of the input matrix.

`2` — Output a vector that contains an element for each row of the input matrix.

Each element of the output vector contains the product of all elements in the corresponding column or row of the input matrix, or the product of the inverses of those elements. The output vector depends on the value of **Number of inputs**:

- **`* or 1`**

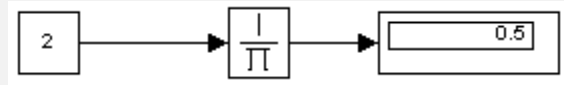
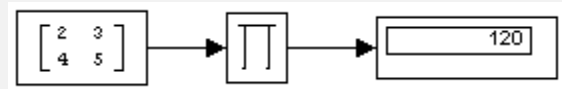
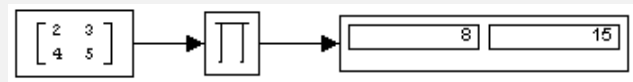
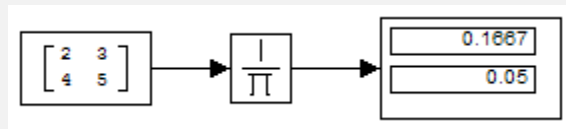
  Multiply the values of the column or row elements

- `/`

  Multiply the inverses of the column or row elements

Examples

This table shows the output of the Product of Elements block for some typical inputs using default block parameter values, except as shown in the table.

Parameter Values	Examples
Multiplication: `Element-wise(.*)` **Number of inputs:** `*`	
Multiplication: `Element-wise(.*)` **Number of inputs:** `/`	
Multiplication: `Element-wise(.*)` **Number of inputs:** `*`	
Multiplication: `Element-wise(.*)` **Number of inputs:** `*` **Multiply over:** `All dimensions`	
Multiplication: `Element-wise(.*)` **Number of inputs:** `*` **Multiply over:** `Specified dimension` **Dimension:** `1`	
Multiplication: `Element-wise(.*)` **Number of inputs:** `/` **Multiply over:** `Specified dimension` **Dimension:** `2`	

Purpose

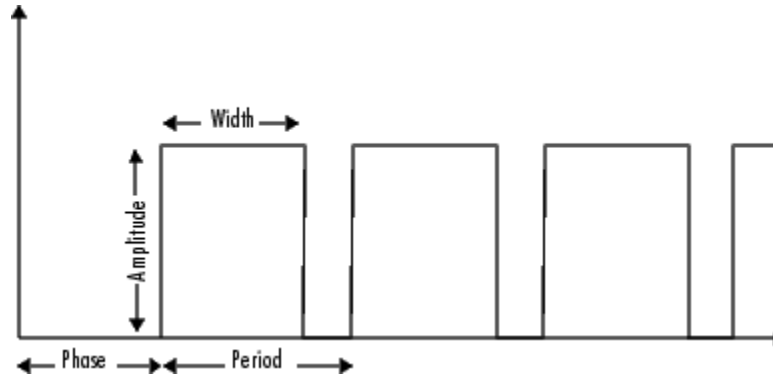
Generate square wave pulses at regular intervals

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.

Use the **Pulse type** parameter 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 software computes the block's outputs only at times when the output actually changes. This choice can result in fewer computations for computing 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 has a variable sample time. Simulink software uses brown as the sample time color of such blocks (see "Displaying Sample Time Colors" for more information).

Simulink software cannot use a fixed-step 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 software computes a fixed sample time for the time-based pulse generators. Then the time-based pulse generators simulate as sample-based.

Tip 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 **Solver** pane of the Configuration Parameters dialog box.

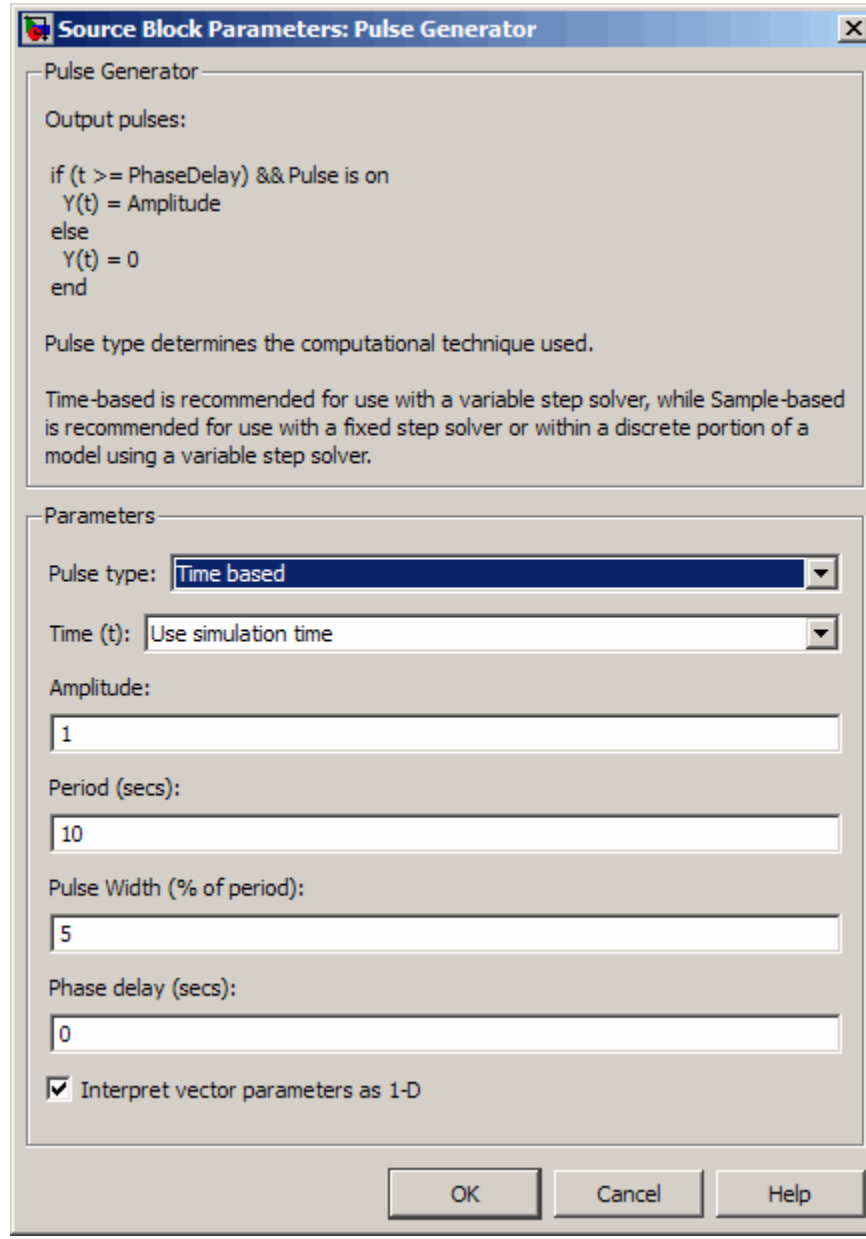
If you select time-based as the block's pulse type, you must specify the pulse's phase delay and period in units of seconds. If you specify sample-based, you must specify the block's sample time in seconds, using the **Sample Time** parameter, then specify the block's phase delay and period as integer multiples of the sample time. For example, suppose that you specify a sample time of 0.5 second and 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 numeric data type supported by Simulink software, including fixed-point data types. 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 software, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box



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

Pulse type

The pulse type for this block: time-based or sample-based. The default is time-based.

Time

Specifies whether to use simulation time or an external signal as the source of values for the output pulse's time variable. If you specify an external source, the block displays an input port for connecting the source. The output pulse differs as follows:

- If you select `Use simulation time`, the block generates an output pulse where the time variable equals the simulation time.
- If you select `Use external signal`, the block generates an output pulse where the time variable equals the value from the input port, which can differ from the simulation time.

Note If you select `Use external signal` and click **OK**, an **Apply** button appears the next time that you open the block dialog box. This **Apply** button disappears if you select `Use simulation time` and click **OK**.

Amplitude

The pulse amplitude. The default is `1`.

Period

The pulse period specified in seconds if the pulse type is time-based or as number of sample times if the pulse type is sample-based. The default is `10` seconds.

Pulse width
The duty cycle specified as the percentage of the pulse period that the signal is on if time-based or as number of sample times if sample-based. The default is `5` percent.

Phase delay
The delay before the pulse is generated specified in seconds if the pulse type is time-based or as number of sample times if the pulse type is sample-based. The default is `0` seconds.

Sample time
The length of the sample time for this block in seconds. This parameter appears only if the block's pulse type is sample-based. See "How to Specify the Sample Time" in the Simulink User's Guide for more information.

Interpret vector parameters as 1-D
If you select this check box 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. See "Determining the Output Dimensions of Source Blocks" in the "Working with Signals" chapter of the Simulink User's Guide.

Characteristics

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

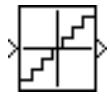
Purpose

Discretize input at specified interval

Library

Discontinuities

Description



The Quantizer block passes its input signal through a stair-step function so that many neighboring points on the input axis are mapped to one point on the output axis. The effect is to quantize a smooth signal into a stair-step output. The output is computed using the round-to-nearest method, which produces an output that is symmetric about zero.

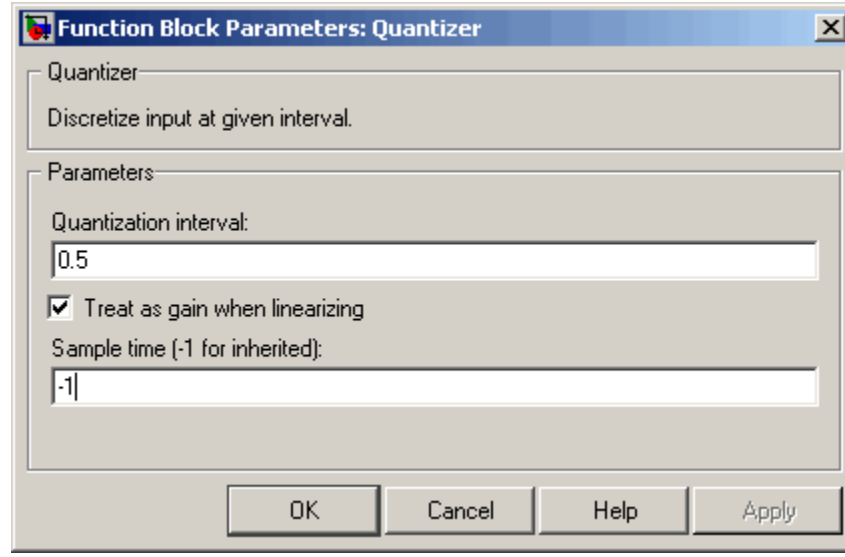
```
y = q * round(u/q)
```

where `y` is the output, `u` the input, and `q` the **Quantization interval** parameter.

Data Type Support

The Quantizer block accepts and outputs real or complex signals of type `single` or `double`.

Parameters and Dialog Box



Quantization interval
The interval around which the output is quantized. Permissible output values for the Quantizer block are `n*q`, where `n` is an integer and `q` the **Quantization interval**. The default is `0.5`.

Treat as gain when linearizing
Simulink software by default treats the Quantizer block as unity gain when linearizing. This is the large signal linearization case. If you clear this box, the linearization routines assume the small signal case and set the gain to zero.

Sample time (-1 for inherited)
Specify the sample time of this Outport block. See "How to Specify the 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.

Characteristics

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

Purpose

Generate constantly increasing or decreasing signal

Library

Sources

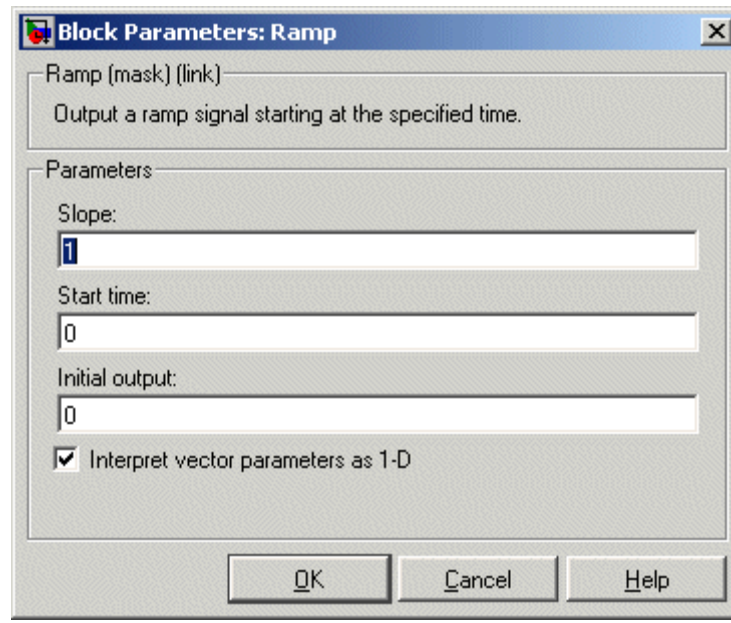
Description

The Ramp block generates a signal that starts at a specified time and value and changes by a specified rate. The block's **Slope**, **Start time**, and **Initial output** parameters determine the characteristics of the output signal. All must have the same dimensions after scalar expansion.

Data Type Support

The Ramp block outputs signals of type `double`.

Parameters and Dialog Box



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

Slope
The rate of change of the generated signal. The default is `1`.

Start time
The time at which the signal begins to be generated. The default is `0`.

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. See "Determining the Output Dimensions of Source Blocks" in the "Working with Signals" chapter of the Using Simulink documentation.

Characteristics

Sample Time	Inherited from driven block
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	Yes

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 **Seed** parameter as a vector.

To generate uniformly distributed random numbers, use the Uniform Random Number block.

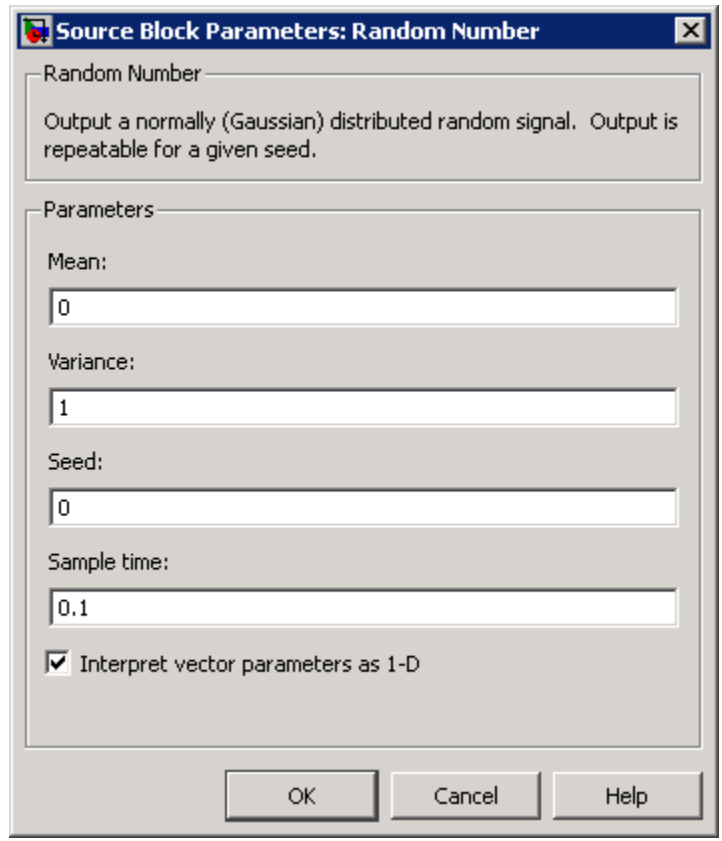
Avoid integrating a random signal, because solvers are meant to integrate relatively smooth signals. Instead, use the Band-Limited White Noise block.

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 During Simulation" 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`.

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 `0.1`, which matches the default sample time of the Band-Limited White Noise block. See "How to Specify the 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. See "Determining the Output Dimensions of Source Blocks" in the "Working with Signals" chapter of the Using Simulink documentation.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Multidimensionalized	Yes
Zero Crossing	No

Rate Limiter

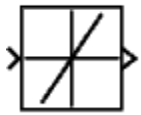
Purpose

Limit rate of change of 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:

  $$y(i) = u(i)$$

When the block is running in continuous mode (for example, **Sample time mode** is `inherited` and **Sample time** of the driving block is zero), the **Initial condition** is ignored. The block output at `t = 0` is equal to the initial input:

$$y(0) = u(0)$$

When the block is running in discrete mode (for example, **Sample time mode** is `inherited` and **Sample time** of the driving block is nonzero), the **Initial condition** is preserved:

$$y(-1) = Ic$$

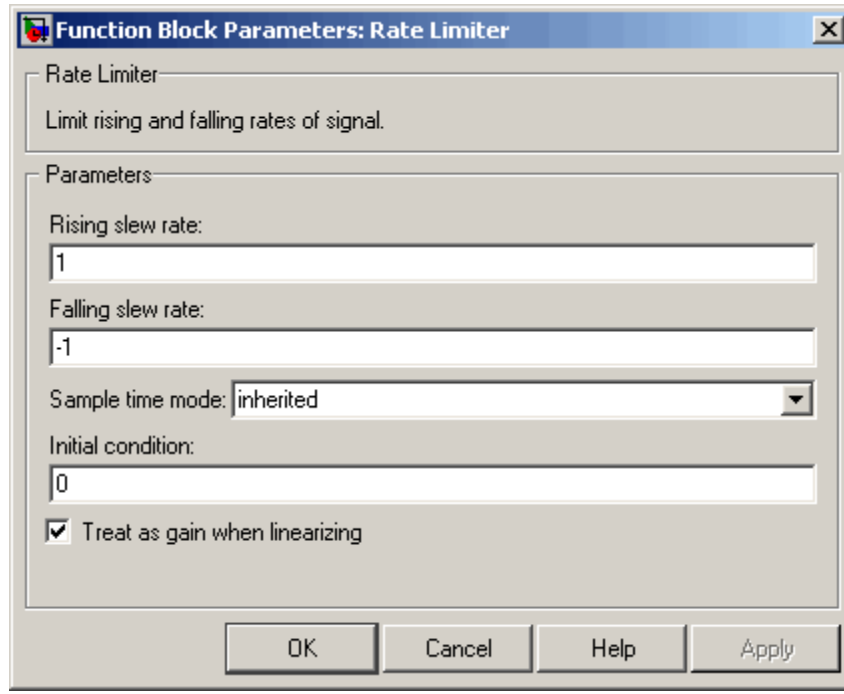
where *Ic* is the initial condition. The block output at `t = 0` is calculated as if *rate* is outside the bounds of *R* and *F*. For `t = 0`, *rate* is calculated as follows:

$$rate = \frac{u(0) - y(-1)}{sample\ time}$$

Data Type Support

The Rate Limiter block accepts and outputs signals of any numeric data type supported by Simulink software, 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 software 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 software 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

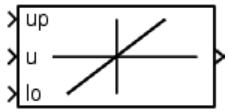
Purpose

Limit rising and falling rates of 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.

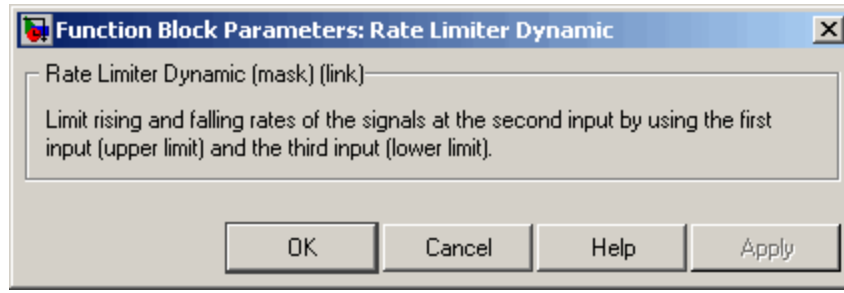
The external signal `lo` sets the lower limit on the falling rate of the signal.

Note You cannot use a variable-step solver to simulate models that contain this block. You must use a fixed-step solver.

Data Type Support

The Rate Limiter Dynamic block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

Rate Limiter

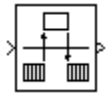
Purpose

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 a block operating at a different rate. Use the block parameters to trade data integrity and deterministic transfer for faster response or lower memory requirements. To learn about data integrity and deterministic data transfer, see "Data Transfer Problems" in the Real-Time Workshop User's Guide.

Options

Transition Handling Options	Block Parameter Settings
• Data integrity • Deterministic data transfer • Maximum latency	Select: • **Ensure data integrity during data transfer** • **Ensure deterministic data transfer**
• Data integrity • Nondeterministic data transfer • Minimum latency • Higher memory requirements	Select: • **Ensure data integrity during data transfer** Clear: • **Ensure deterministic data transfer**
• Potential loss of data integrity • Nondeterministic data transfer • Minimum latency • Lower memory requirements	Clear: • **Ensure data integrity during data transfer** • **Ensure deterministic data transfer**

Dependencies

The behavior of the Rate Transition block depends on:

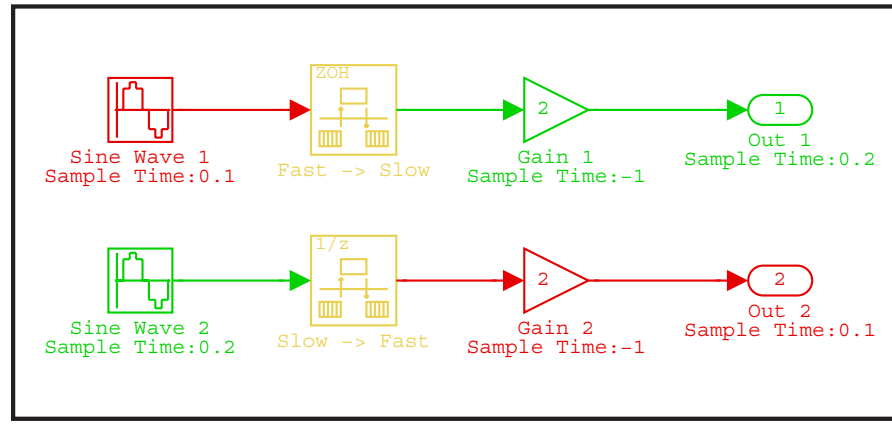
- Sample times of the ports to which the block connects (see "Effects of Synchronous Sample Times" on page 2-1047 and "Effects of Asynchronous Sample Times" on page 2-1049)
- Priorities of the tasks for the source and destination sample times (see "Sample time properties" in the Simulink documentation)
- Whether the model specifies a fixed- or variable-step solver (see "Solvers" in the Simulink documentation)

Block Labels

When you update your diagram, a label appears on the Rate Transition block to indicate simulation behavior.

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 block behavior label shows the method that ensures safe transfer of data between tasks operating at different rates. You can use the sample-time colors feature (see "Displaying Sample Time Colors" in the Simulink User's Guide) to display the relative rates that the block bridges. Consider, for example, the following diagram.



Sample-time colors and the block behavior label show 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 "Handling Rate Transitions" in the Real-Time Workshop User's Guide for more information.

Effects of Synchronous Sample Times

The following table summarizes how each label appears if the sample times of the input and output ports (`inTs` and `outTs`) are periodic, or synchronous.

Block Settings		Block Label		
Rate Transition	**Conditions for Rate Transition Block**	**With Data Integrity and Determinism**	**With Only Data Integrity**	**Without Data Integrity or Determinism**
`inTs = outTs` (Equal)	`inTsOffset < outTsOffset`	None (error)	`Buf`	`Copy` or `NoOp` (see note that follows the table)
	`inTsOffset = outTsOffset`	`Copy` or `NoOp` (see note that follows the table)	`Copy` or `NoOp` (see note that follows the table)	
	`inTsOffset > outTsOffset`	None (error)	`Db_buf`	
`inTs < outTs` (Fast to slow)	`inTs = outTs / N` and `inTsOffset, outTsOffset = 0`	`ZOH`	`Buf`	
	`inTs = outTs / N` and `inTsOffset ≤ outTsOffset`	None (error)		
	`inTs = outTs / N` and `inTsOffset > outTsOffset`	None (error)	`Db_buf`	
	`inTs ≠ outTs / N`	None (error)		
`inTs > outTs` (Slow to fast)	`inTs = outTs * N` and `inTsOffset, outTsOffset = 0`	`1/z`	`Db_buf`	
	`inTs = outTs * N` and `inTsOffset ≤ outTsOffset`	None (error)		
	`inTs = outTs * N` and `inTsOffset > outTsOffset`	None (error)		
	`inTs ≠ outTs * N`	None (error)		
KEY				

Block Settings		Block Label		
Rate Transition	**Conditions for Rate Transition Block**	**With Data Integrity and Determinism**	**With Only Data Integrity**	**Without Data Integrity or Determinism**
	• `inTs, outTs`: Sample times of input and output ports, respectively • `inTsOffset, outTsOffset`: Sample time offsets of input and output ports, respectively • `N`: Integer value > 1			

Note If you select **Block reduction** in the Optimization pane of the Configuration Parameters dialog box, `Copy` reduces to `NoOp`. No code generation occurs for a Rate Transition block with a `NoOp` label. To prevent a block from being reduced when block reduction is on, add a test point to the block output (see "Working with Test Points" in the Simulink documentation).

Effects of Asynchronous Sample Times

The following table summarizes how each label appears if the sample time of the input or output port (`inTs` or `outTs`) is not periodic, or asynchronous.

Block Settings	Block Label		
	With Data Integrity and Determinism	**With Only Data Integrity**	**Without Data Integrity or Determinism**
`inTs` = `outTs`	`Copy`	`Copy`	`Copy`
`inTs` ≠ `outTs`	None (error)	`Db_buf`	
KEY			

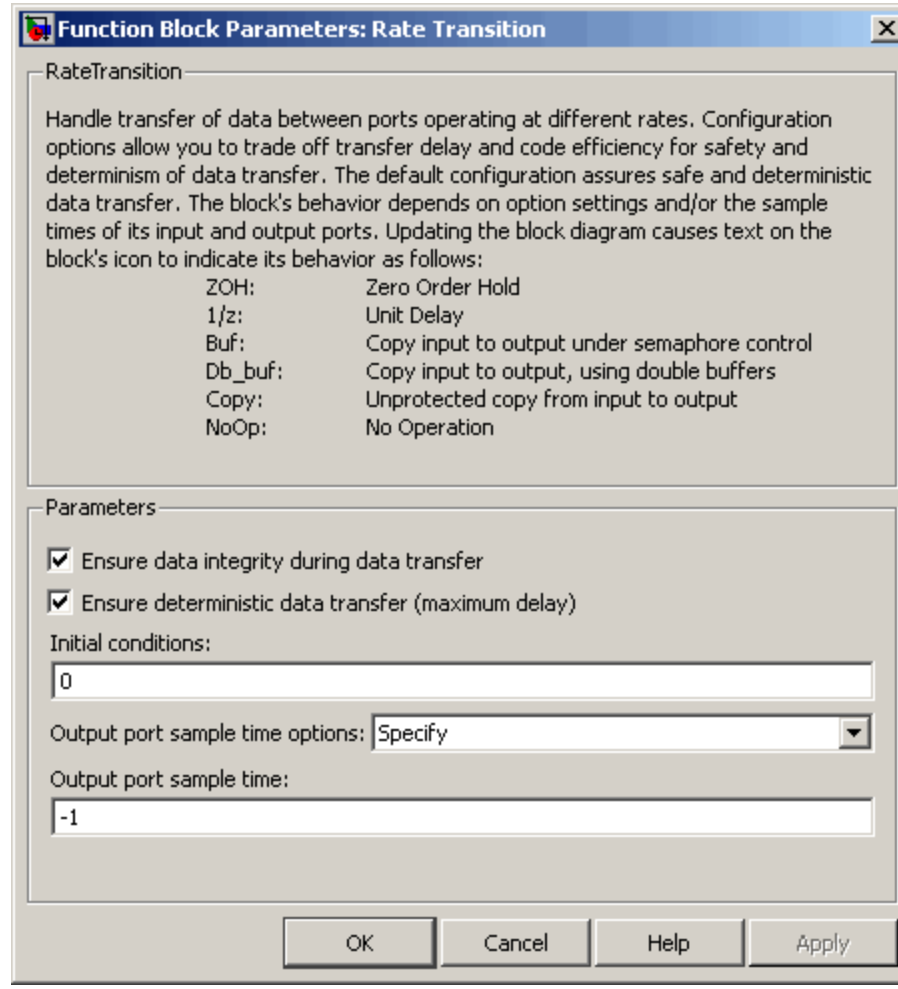
Block Settings	Block Label		
	With Data Integrity and Determinism	With Only Data Integrity	Without Data Integrity or Determinism
• `inTs, outTs`: Sample times of input and output ports, respectively			

Data Type Support

The Rate Transition block accepts signals of any data type supported by Simulink software, including fixed-point and enumerated data types.

For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the Simulink User's Guide.

Parameters and Dialog Box



Ensure data integrity during data transfer

Selecting this check box results in generated code that ensures data integrity when the block transfers data. If you select this check box and the transfer is nondeterministic (see **Ensure deterministic data transfer** 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, which can lead to loss of data during nondeterministic data transfers. Select this check box if you want the generated code to operate with maximum responsiveness (i.e., nondeterministically) and data integrity. See "Rate Transition Block Options" in the Real-Time Workshop User's Guide for more information.

Ensure deterministic data transfer (maximum delay)

Selecting this check box results in generated code that transfers data at the sample rate of the slower block, i.e., deterministically. If you do not select this check box, data transfers occur as soon as new data is available from the source block and the receiving block is ready to receive the data. You avoid transfer delays, thus ensuring that the system operates with maximum responsiveness. However, transfers can occur unpredictably, which is undesirable in some applications. See "Rate Transition Block Options" in the Real-Time Workshop User's Guide for more information.

Initial conditions

This parameter applies only to slow-to-fast transitions. It specifies the initial output of the Rate Transition block at the beginning of a transition when there is no output from the slow block connected to the input of the Rate Transition block. Simulink software does not allow the initial output of this block to be `inf` or `NaN`.

Output port sample time options

Specifies a mode for setting the output port sample time. The options are:

- `Specify` — Allows you to use the **Output port sample time** parameter to specify the output rate to which the Rate Transition block converts its input rate.
- `Inherit` — Specifies that the Rate Transition block inherits an output rate from the block to which its output port is connected.

- `Multiple of input port sample time` — Allows you to use the **Sample time multiple (>0)** parameter to specify the Rate Transition block's output rate as a multiple of its input rate.

Output port sample time

This parameter is visible only if the **Output port sample time options** parameter is set to `Specify`. Enter a value that specifies the output rate to which the block converts its input rate. 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 "How to Specify the Sample Time" in the Simulink User's Guide for information on how to specify the output rate.

Sample time multiple (>0)

This parameter is visible only if the **Output port sample time options** parameter specifies `Multiple of input port sample time`. Enter a positive value that specifies the output rate as a multiple of the input port sample time. The default value (`1`) specifies that the output rate is the same as the input rate. A value of `0.5` specifies that the output rate is half of the input rate, while a value of `2` specifies that the output rate is twice the input rate.

Bus Support

The Rate Transition block is a bus-capable block. The input can be a virtual or nonvirtual bus signal, with the restriction that **Initial conditions** must be zero or a nonzero scalar. All signals in a nonvirtual bus input to a Rate Transition block must have the same sample time, even if the elements of the associated bus object specify inherited sample times. You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus. See "Using Composite Signals" and "Bus-Capable Blocks" in the Simulink User's Guide for more information.

Characteristics

Bus-capable	Yes, with restrictions as noted above
Direct Feedthrough	No, for slow-to-fast transitions that are protected, i.e., for which you select the **Ensure data integrity during data transfer** check box. Yes, otherwise.
Sample Time	This block supports discrete-to-discrete transitions.
Scalar Expansion	Yes, of input.
Dimensionalized	Yes
Multidimensionalized	Yes
Zero Crossing	No

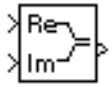
Purpose

Convert real and/or imaginary inputs to 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.

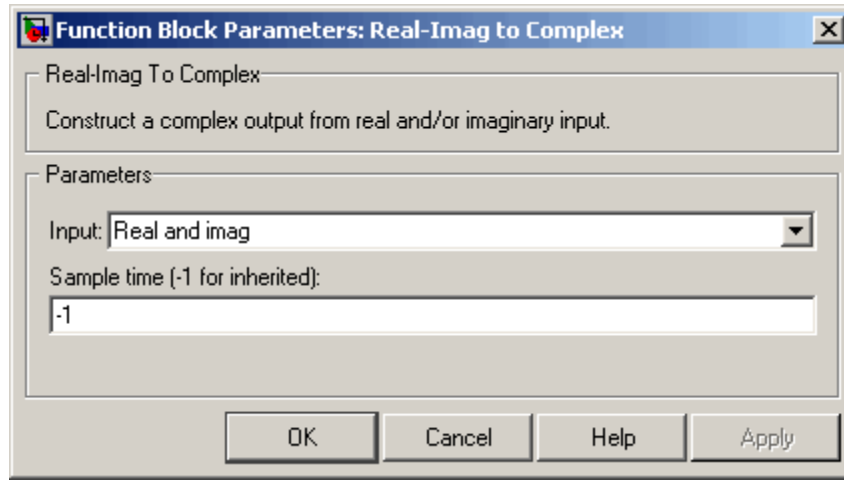
The input signals and real or imaginary output parameter can be of any data type supported by Simulink software, except `Boolean`. The Real-Imag to Complex block supports fixed-point data types. The output is of the same type as the input or parameter that determines the output.

For a discussion on the data types supported by Simulink software, 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 "How to Specify the 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
Multidimensionalized	Yes
Zero Crossing	No

Relational Operator

Purpose

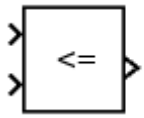
Perform specified relational operation on inputs

Library

Logic and Bit Operations

Description

Two-Input Mode



By default, the Relational Operator block compares two inputs using the **Relational operator** parameter that you specify. The first input corresponds to the top input port and the second input to the bottom input port. (See "How to Rotate a Block" in the Simulink documentation for a description of the port order for various block orientations.)

You can specify one of the following operations in two-input mode.

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 and 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 occurs prior to comparison.

You can specify the output data type using the **Output data type** parameter. The output equals 1 for TRUE and 0 for FALSE.

Tip Select an output data type that represents zero exactly. Data types that satisfy this condition include signed and unsigned integers and any floating-point data type.

One-Input Mode

When you select one of the following operations for **Relational operator**, the block switches to one-input mode.

Operation	Description
`isInf`	TRUE if the input is `Inf`
`isNaN`	TRUE if the input is `NaN`
`isFinite`	TRUE if the input is finite

For an input that is not floating-point, the block produces the following output.

Data Type	Operation	Block Output
Fixed-point Boolean Built-in integer	`isInf`	FALSE
	`isNaN`	FALSE
	`isFinite`	TRUE

Data Type Support

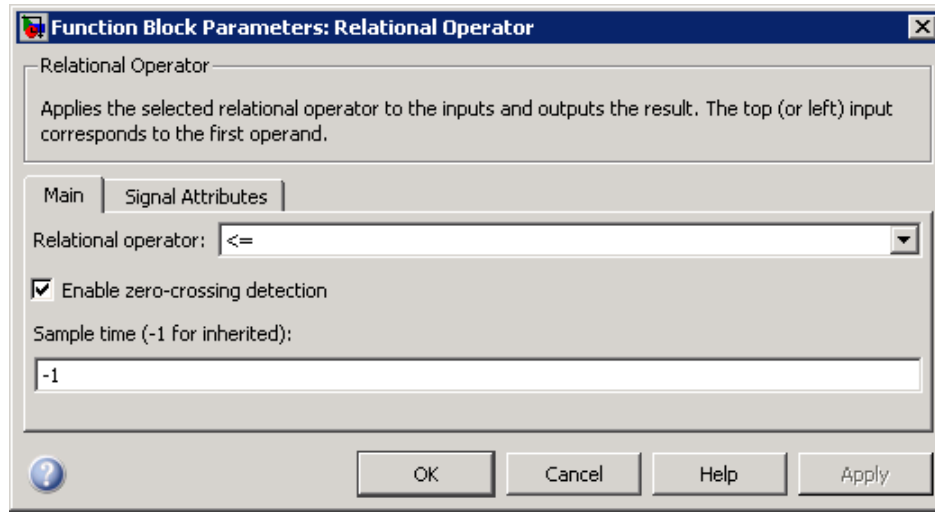
The Relational Operator block accepts real or complex signals of any data type supported by Simulink software, including fixed-point and enumerated data types. For two-input mode, one input can be real and

the other complex when the operator is == or ~=. Complex inputs are supported only for ==, ~=, `isInf`, `isNaN`, and `isFinite`.

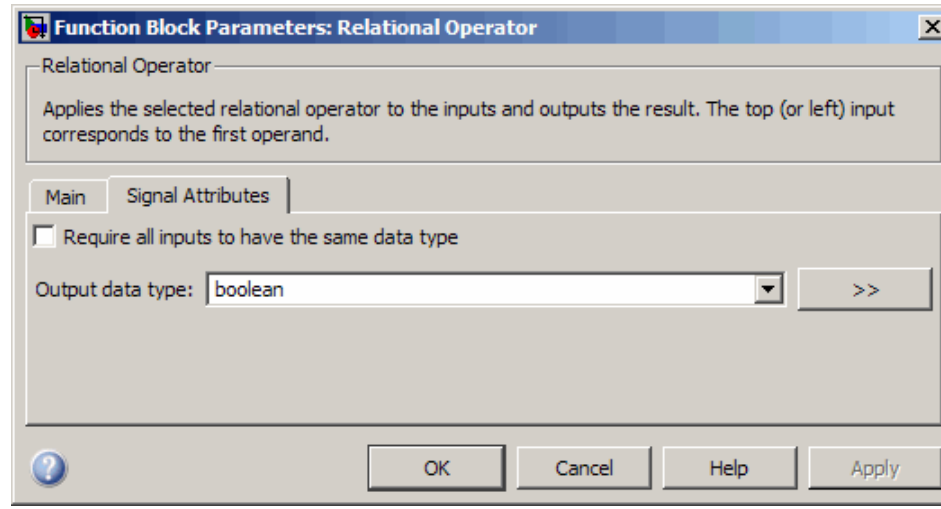
For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Relational Operator block dialog box appears as follows:



The **Signal Attributes** pane of the Relational Operator block dialog box appears as follows:



Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See “Specifying Block Output Data Types” in *Simulink User's Guide* for more information.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Relational operator

Specify the operation for comparing two inputs or determining the signal type of one input.

Settings

Default: `<=`

`==`
: 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

`isInf`
: TRUE if the input is `Inf`

`isNaN`
: TRUE if the input is `NaN`

`isFinite`
: TRUE if the input is finite

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Enable zero-crossing detection

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

Settings

Default: On

 On

Enable zero-crossing detection.

 Off

Do not enable zero-crossing detection.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits or specify another appropriate sample time such as continuous or inherited.

Settings

Default: -1

By default, the block inherits its sample time based upon the context of the block within the model. To set a different sample time, enter a valid sample time based upon the table in "Types of Sample Time".

See also "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Require all inputs to have the same data type

Require that all inputs have the same data type.

Settings

Default: Off

 On

Require that all inputs have the same data type.

 Off

Do not require that all inputs have the same data type.

Dependency

This check box is not available when you select `isInf`, `isNaN`, or `isFinite` for **Relational operator**, because the block is in one-input mode.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `boolean`

`Inherit: Logical (see Configuration Parameters: Optimization)`

Use the **Implement logic signals as Boolean data** configuration parameter (see "Implement logic signals as Boolean data (vs. double)") to specify the output data type.

Note This option supports models created before the `boolean` option was available. Use one of the other options, preferably `boolean`, for new models.

`boolean`

Specifies the output data type as `boolean`.

`fixdt(1,16)`

Specifies the output data type as `fixdt(1,16)`.

`<data type expression>`

Use the name of a data type object (for example, `Simulink.NumericType`)

Tip To enter a built-in data type (`double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`), enclose the expression in single quotes. For example, enter `'double'` instead of `double`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<class name>`

`Expression`
: Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of inputs
Dimensionalized	Yes
Multidimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled

Relay

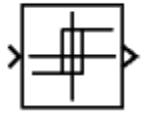
Purpose

Switch output between two constants

Library

Discontinuities

Description



The Relay block allows its output to switch between two specified values. When the relay is on, it remains on until the input drops below the value of the **Switch off point** parameter. When the relay is off, it remains off until the input exceeds the value of the **Switch on point** parameter. The block accepts one input and generates one output.

The **Switch on point** value must be greater than or equal to the **Switch off point**. Specifying a **Switch on point** value greater than the **Switch off point** value models hysteresis, whereas specifying equal values models a switch with a threshold at that value.

Note When the initial input falls between the **Switch off point** and **Switch on point** values, the initial output is the value when the relay is off.

Data Type Support

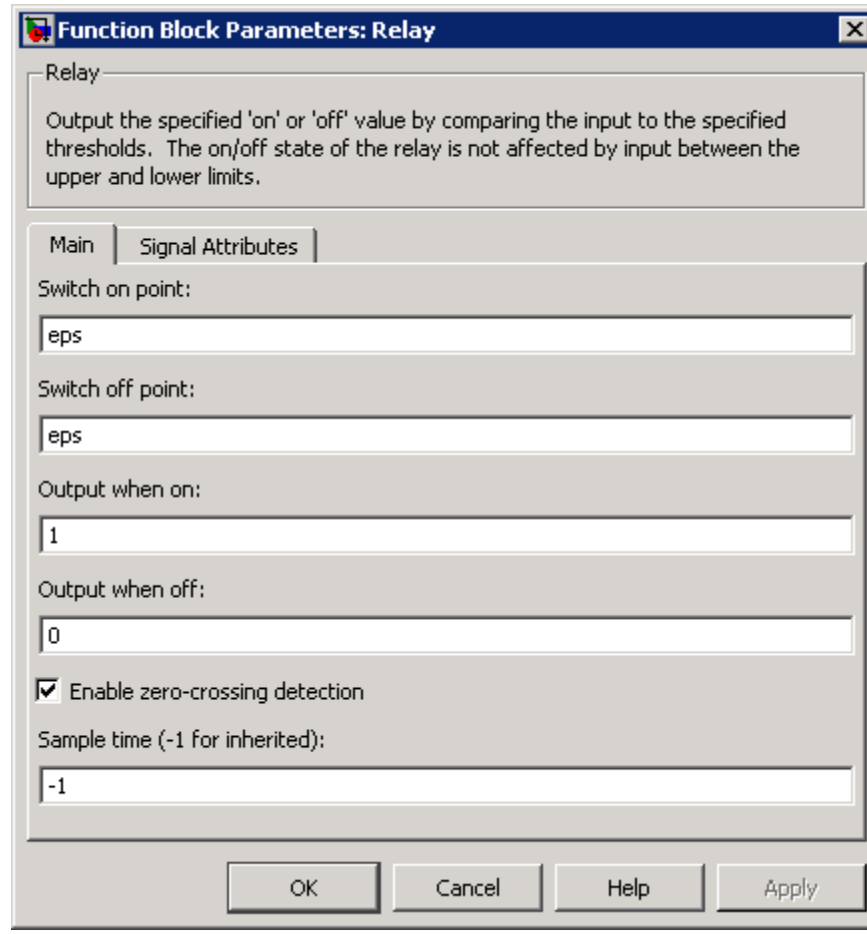
The Relay block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

For more information, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

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



Switch on point

The "on" threshold for the relay. The **Switch on point** parameter is converted to the input data type offline using round-to-nearest and saturation.

Switch off point
The "off" threshold for the relay. The **Switch off point** parameter is converted to the input data type offline using round-to-nearest and saturation.

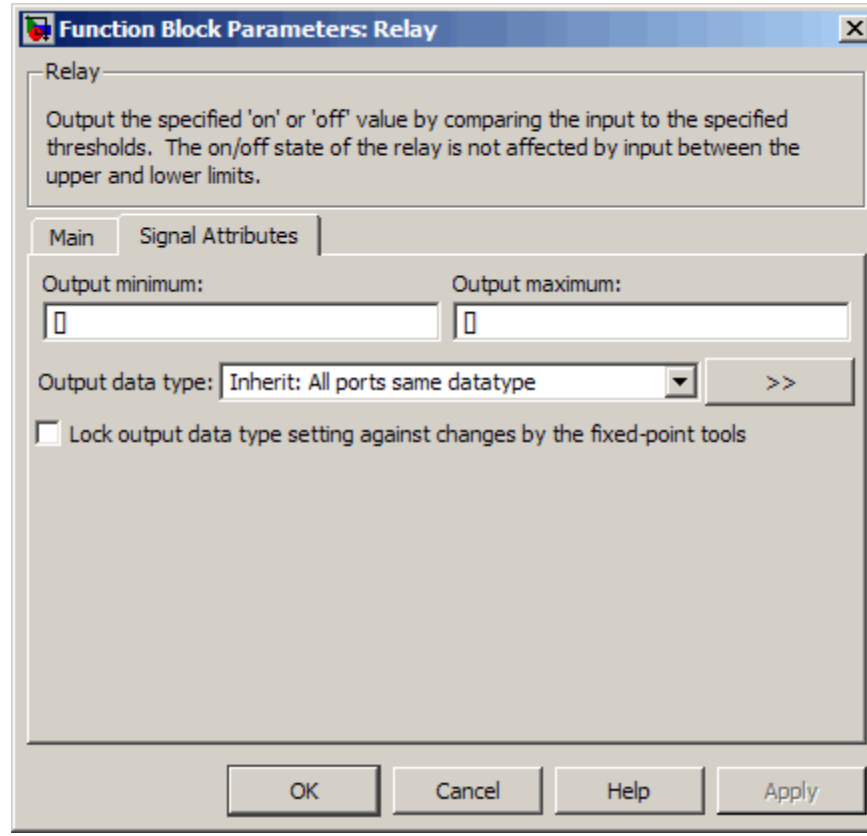
Output when on
The output when the relay is on.

Output when off
The output when the relay is off.

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

Sample time (-1 for inherited)
Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in the Simulink documentation for more information.

The **Signal Attributes** pane of the Relay block dialog box appears as follows:



Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see “Checking Parameter Values”)
- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Characteristics

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

Repeating Sequence

Purpose

Generate arbitrarily shaped periodic signal

Library

Sources

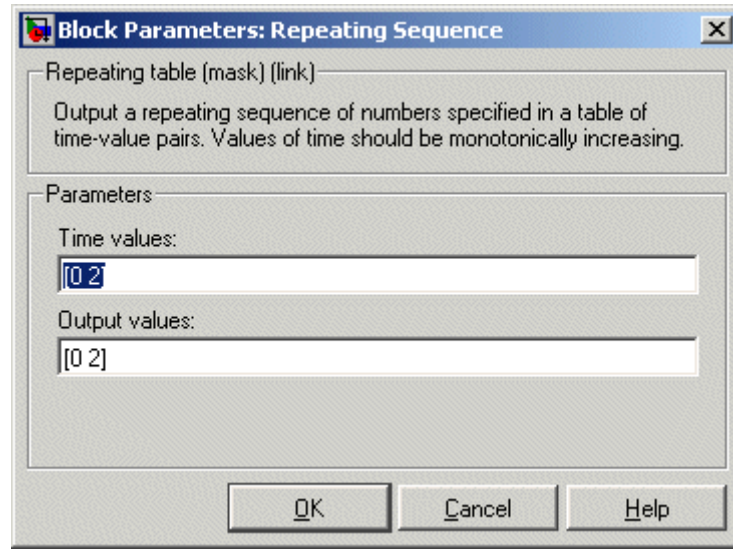
Description

The Repeating Sequence block outputs a periodic scalar signal having a waveform that you specify. You can specify any waveform, using the **Time values** and **Output values** parameters. The **Time values** parameter specifies a vector of sample times. The **Output values** parameter specifies a vector of signal amplitudes at the corresponding sample times. Together, the two parameters specify a sampling of the output waveform at points measured from the beginning of the interval over which the waveform repeats (the period of the signal). By default, the **Time values** and **Output values** parameters are both set to `[0 2]`. This default setting specifies a sawtooth waveform that repeats every `2` seconds from the start of the simulation and has a maximum amplitude of `2`. The Repeating Sequence block uses linear interpolation to compute the value of the waveform between the sample points you specify.

Data Type Support

The Repeating Sequence 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 During Simulation" in the Simulink documentation for details.

Time values

A vector of monotonically increasing time values. The default is `[0 2]`.

Output values

A vector of output values. Each element corresponds to the time value in the same column. The default is `[0 2]`.

Characteristics

Sample Time	Continuous
Scalar Expansion	No
Dimensionalized	No
Zero-Crossing Detection	No

See Also Repeating Sequence Interpolated, Repeating Sequence Stair

Purpose

Output discrete-time sequence and repeat, interpolating between data points

Library

Sources

Description

The Repeating Sequence Interpolated block outputs a discrete-time sequence and then repeats it. Between data points, the block uses the method specified by the **Lookup Method** parameter to determine the output.

Data Type Support

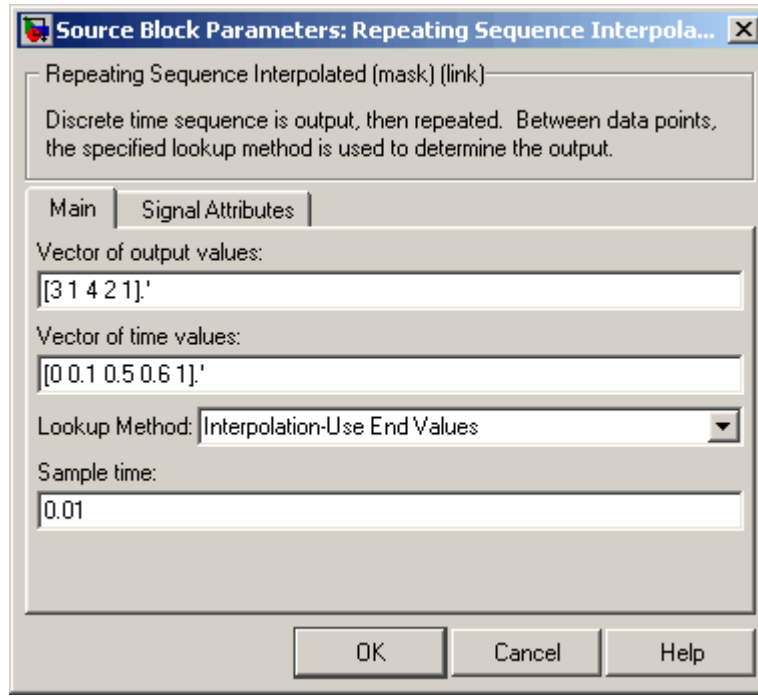
The Repeating Sequence Interpolated block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

For more information, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Repeating Sequence Interpolated block dialog box appears as follows:



Vector of output values

Column vector containing output values of the discrete time sequence.

Vector of time values

Column vector containing time values. The time values must be strictly increasing, and the vector must have the same size as the vector of output values.

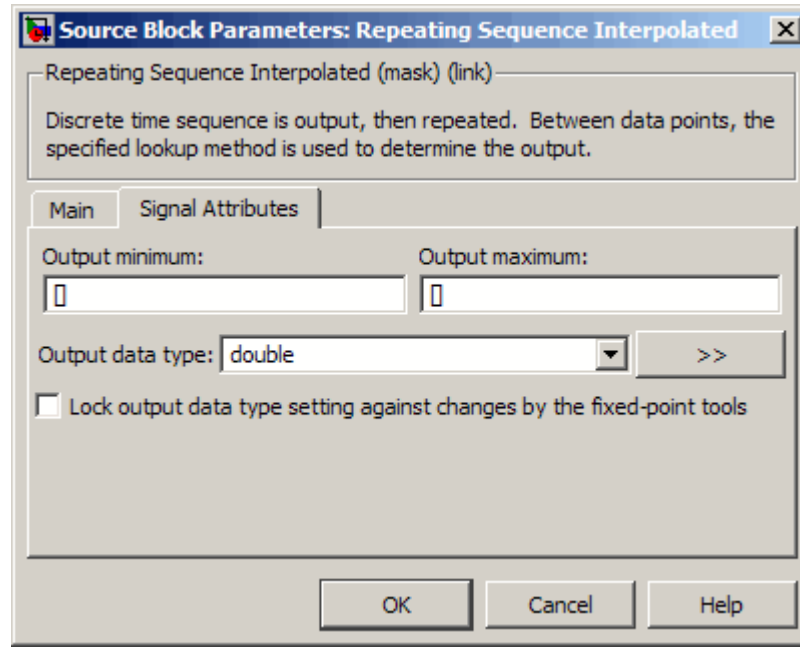
Lookup Method

Specify the lookup method to determine the output between data points.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in the *Simulink User's Guide* for more information.

The **Signal Attributes** pane of the Repeating Sequence Interpolated block dialog box appears as follows:



Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")

- Automatic scaling of fixed-point data types

Output maximum

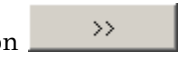
Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes
Zero-Crossing Detection	No

See Also

Repeating Sequence, Repeating Sequence Stair

Repeating Sequence Stair

Purpose

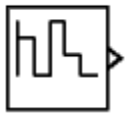
Output and repeat discrete time sequence

Library

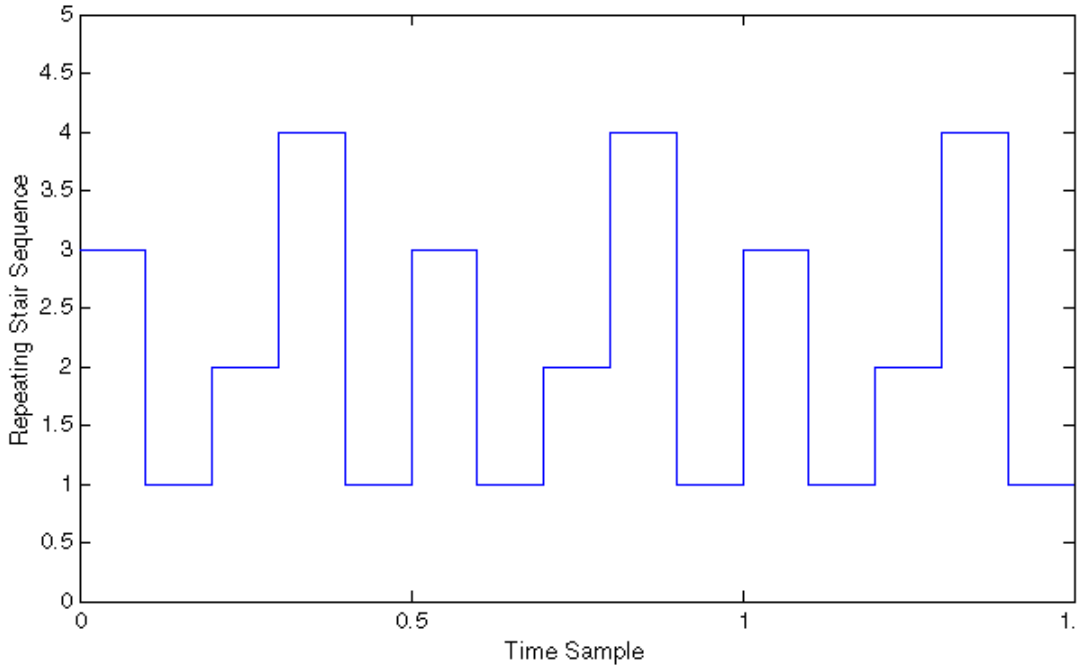
Sources

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.



Data Type Support

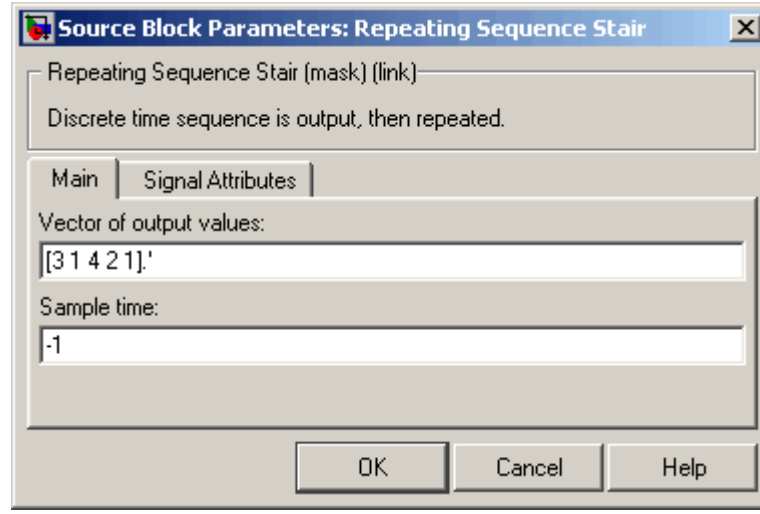
The Repeating Sequence Stair block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

For more information, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Repeating Sequence Stair block dialog box appears as follows:



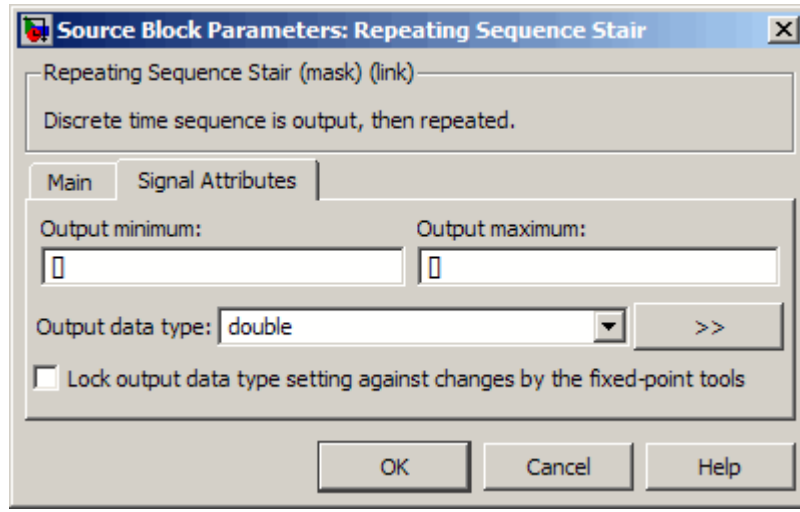
Vector of output values

Vector containing values of the repeating stair sequence.

Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in the *Simulink User's Guide* for more information.

The **Signal Attributes** pane of the Repeating Sequence Stair block dialog box appears as follows:



Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values")
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Repeating Sequence Stair

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Zero-Crossing Detection	No

See Also

Repeating Sequence, Repeating Sequence Interpolated

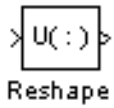
Purpose

Change dimensionality of 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 multidimensional 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, matrix, or multidimensional 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. For multidimensional arrays, the conversion is done along the first dimension.

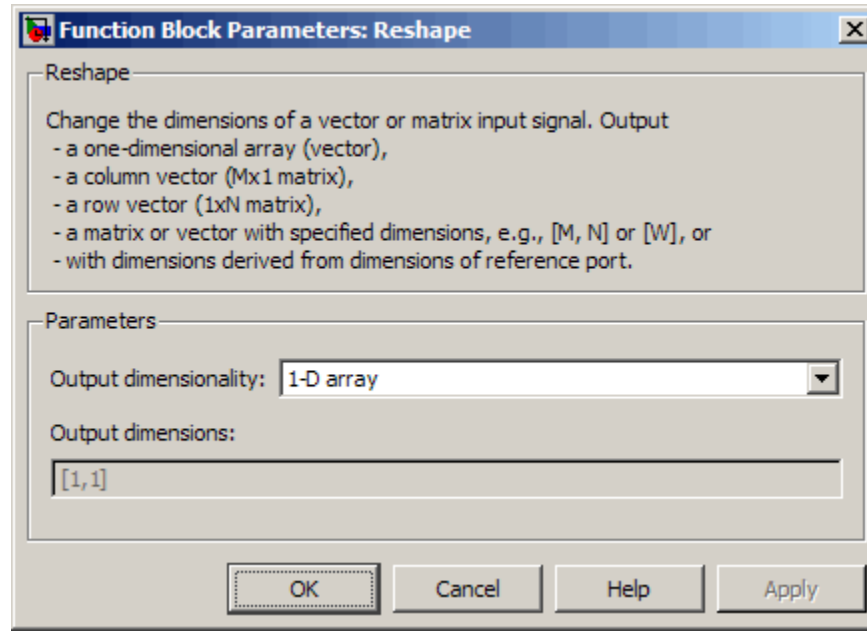
Output Dimensionality	Description
Row vector	Converts a vector, matrix, or multidimensional input signal to a row matrix, i.e., a 1-by-N matrix where N is the number of elements in the input signal. For matrices, the conversion is done in column-major order. For multidimensional arrays, the conversion is done along the first dimension.
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 multi-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 multidimensional arrays, the conversion is done along the first dimension.
Derive from reference input port	Creates a second input port, Ref, on the block. Derives the dimensions of the output signal from the dimensions of the signal input to the Ref input port. Selecting this option disables the **Output dimensions** parameter. When you select this parameter, the input signals for both inport ports, U and Ref, must have the same sampling mode (sample-based or frame-based).

Data Type Support

The Reshape block accepts and outputs signals of any data type supported by Simulink software, including fixed-point and enumerated data types.

For a discussion on the data types supported by Simulink software, 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
Multidimensionalized	Yes
Zero Crossing	No

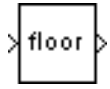
Purpose

Apply rounding function to 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.

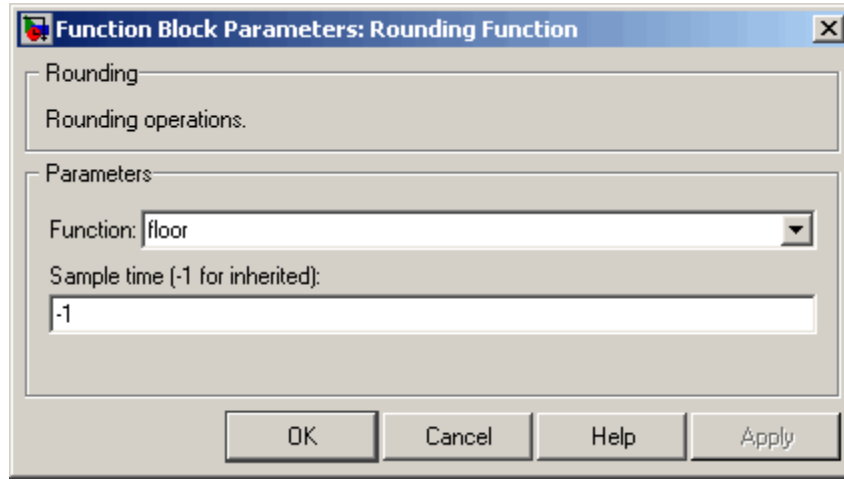
Tip Use the Rounding Function block instead of the Fcn block when you want vector or matrix output, because the Fcn block produces only scalar output.

Also, the Rounding Function block provides two more rounding modes. The Fcn block supports `floor` and `ceil`, but does not support `round` and `fix`.

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 "How to Specify the 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

Saturation

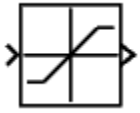
Purpose

Limit range of 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 block clips the signal 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.

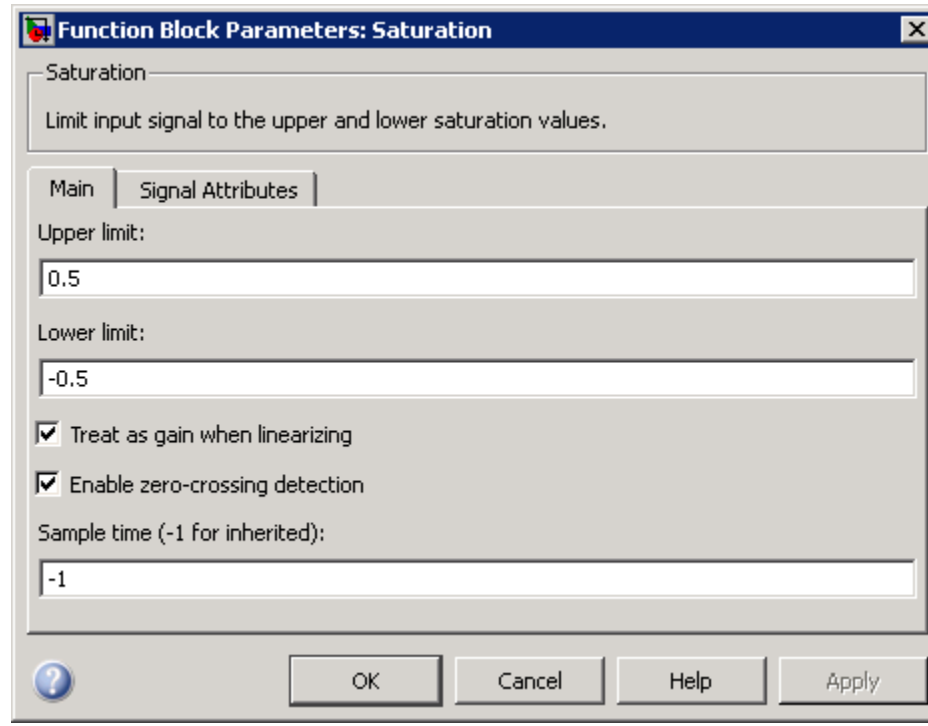
Data Type Support

The Saturation block accepts real signals of any numeric data type supported by Simulink software, except `Boolean`. The Saturation block supports fixed-point data types.

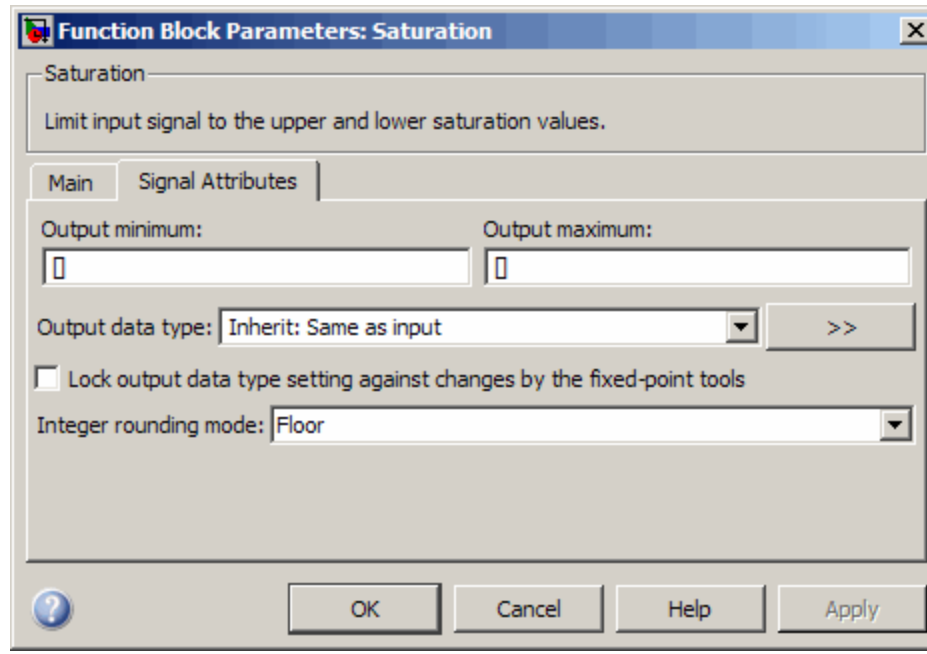
For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

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



The **Signal Attributes** pane of the Saturation block dialog box appears as follows:



Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Upper limit

Specify the upper bound on the input signal.

Settings

Default: `0.5`

Minimum: value from the **Output minimum** parameter

Maximum: value from the **Output maximum** parameter

Tip

- 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 output data type offline using round-to-nearest and saturation.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Lower limit

Specify the lower bound on the input signal.

Settings

Default: `-0.5`

Minimum: value from the **Output minimum** parameter

Maximum: value from the **Output maximum** parameter

Tips

- 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 output data type offline using round-to-nearest and saturation.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Treat as gain when linearizing

Select this parameter to cause the linearization commands to treat the gain as 1

Settings

Default: On

 On

Select to cause the linearization commands to treat the gain as 1.

 Off

Clear to cause the linearization commands to treat the gain as 0.

Tips

Linearization commands in Simulink software treat this block as a gain in state space.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Enable zero-crossing detection

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

Settings

Default: On

 On

Enable zero-crossing detection.

 Off

Do not enable zero-crossing detection.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits or specify another appropriate sample time such as continuous or inherited.

Settings

Default: -1

By default, the block inherits its sample time based upon the context of the block within the model. To set a different sample time, enter a valid sample time based upon the table in "Types of Sample Time".

See also "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Floor`

`Ceiling`
: Round both positive and negative numbers toward positive infinity.

`Convergent`
: Round number to the nearest representable value. If a tie occurs, round to the nearest even stored value.

`Floor`
: Round both positive and negative numbers toward negative infinity.

`Nearest`
: Round number to the nearest representable value. If a tie occurs, round toward positive infinity.

`Round`
: Round number to the nearest representable value. If a tie occurs, round positive numbers toward positive infinity and round negative numbers toward negative infinity.

`Simplest`
: This option provides for an optimization of the rounding code for several blocks.

`Zero`
: Round number toward zero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Rounding” in the *Simulink Fixed Point User's Guide*.

Output minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `Inherit: Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block), `Inherit: Inherit from 'Constant value'` (Constant block), `Inherit: Inherit via back propagation` (Data Type Conversion block), `Inherit: Same as input` (Saturation block)

`Inherit: Inherit via internal rule`
: This option appears for some blocks. Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the calculated 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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 software displays an error message in the Simulation Diagnostics Viewer.

`Inherit: Inherit from 'Constant value'`
: Use data type of **Constant value** parameter. This option appears for some blocks.

`Inherit: Inherit via back propagation`
: Use data type of the driving block.

`Inherit: Same as input`
: Use data type of sole input signal. This option appears for some blocks.

`Inherit: Same as first input`
: Use data type of first input signal. This option appears for some blocks.

`Inherit: Same as accumulator`
: Output data type is the same as accumulator data type. This option appears for some blocks.

`double`
: Output data type is `double`.

`single`
: Output data type is `single`.

`int8`
: Output data type is `int8`.

`uint8`
: Output data type is `uint8`.

`int16`
: Output data type is `int16`.

`uint16`
: Output data type is `uint16`.

`int32`
: Output data type is `int32`.

`uint32`
: Output data type is `unt32`.

`fixdt(1,16,0)`
: Output data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Output data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
: Use an enumerated data type, for example, `Enum: BasicColors`. This option appears for some blocks.

`<data type expression>`
: Use a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<class name>`

`Expression`
: Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

  - `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

See “Specifying a Fixed-Point Data Type” in the *Simulink User’s Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* 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 Detection	Yes, if enabled.

See Also

Saturation Dynamic

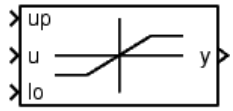
Purpose

Bound range of 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.

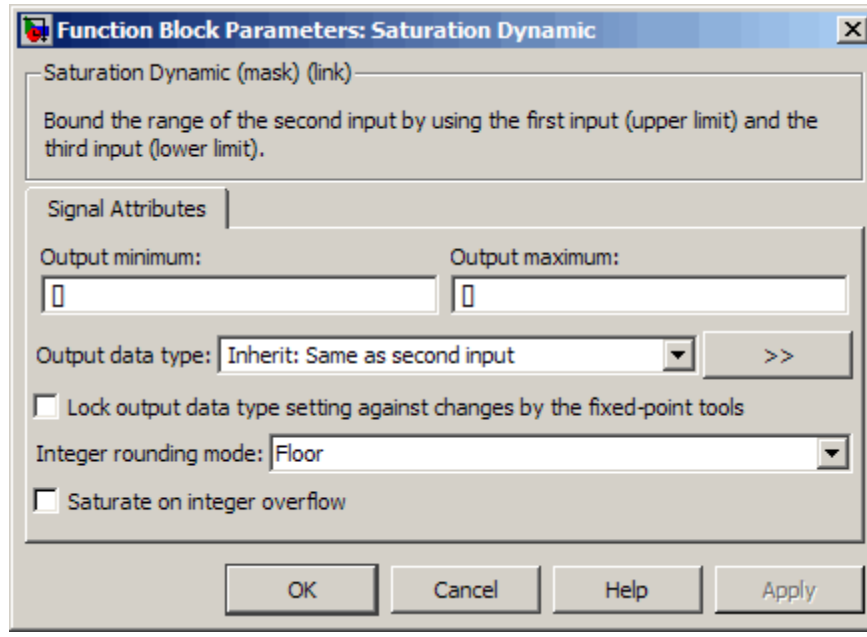
The input for the upper limit is the `up` port, and the input for the lower limit is the `lo` port.

Data Type Support

The Saturation Dynamic block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types.

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

Parameters and Dialog Box



Output minimum

Specify the minimum value that the block should output. The default value, `[]`, is equivalent to `-Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Output maximum

Specify the maximum value that the block should output. The default value, `[]`, is equivalent to `Inf`. Simulink software uses this value to perform:

- Simulation range checking (see “Checking Signal Ranges”)
- Automatic scaling of fixed-point data types

Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Same as second input`
- The name of a built-in data type, for example, `single`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor. For more information, see "Fixed-Point Tool" and "Fixed-Point Advisor" in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate on integer overflow

Select to have overflows saturate.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes

See Also

Saturation

Purpose

Display signals generated during 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) and all axes have a common time range with independent *y*-axes. The Scope block 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.

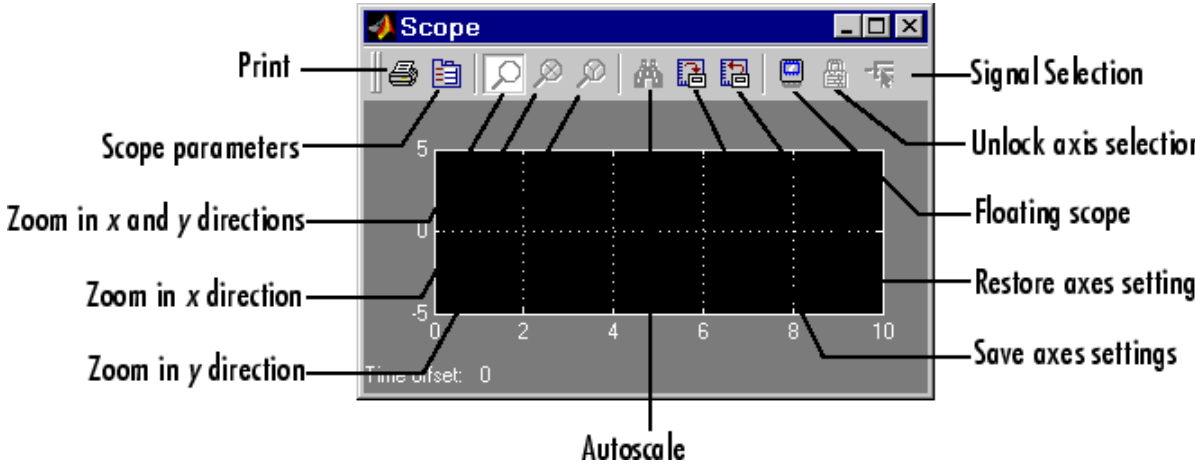
The Scope Block described here is not the same as the Scope Viewer. For help on the scope viewer, see "Things to Know When Using Viewers".

When you start a simulation the Scope windows are not opened, but data is written 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.

Note By default, the Scope block only displays major time step values. However, if a variable-step solver is employed and if the Refine parameter is set to a value greater than 1, minor (intermediate) time step values are displayed in direct proportion to the Refine setting.

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.

Color Coding Used When Displaying Multiple Signals

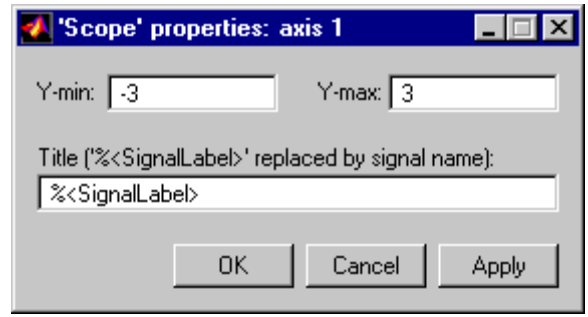
The scope block can display one signal per axes. When displaying a vector or matrix signal on the same axis, the Scope block assigns colors to each signal element, in this order:

1. Yellow
2. Magenta
3. Cyan
4. Red
5. Green
6. Dark Blue

The Scope block cycles through the colors if a signal has more than six elements.

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.

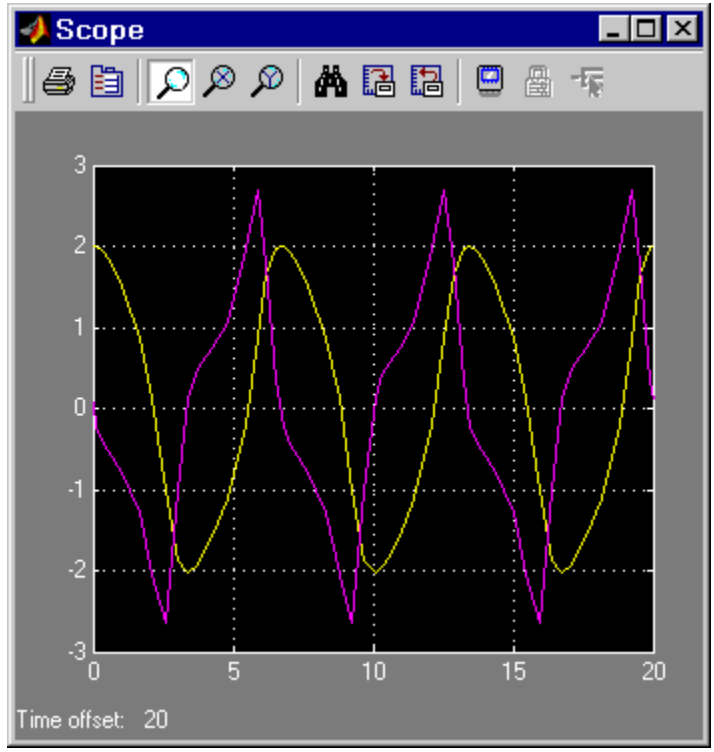
Title
Enter the title of the plot. You can include a signal label in the title by typing `%<SignalLabel>` as part of the title string (`%<SignalLabel>` is replaced by the signal label).

Note You cannot add a title to a floating scope.

Time Offset

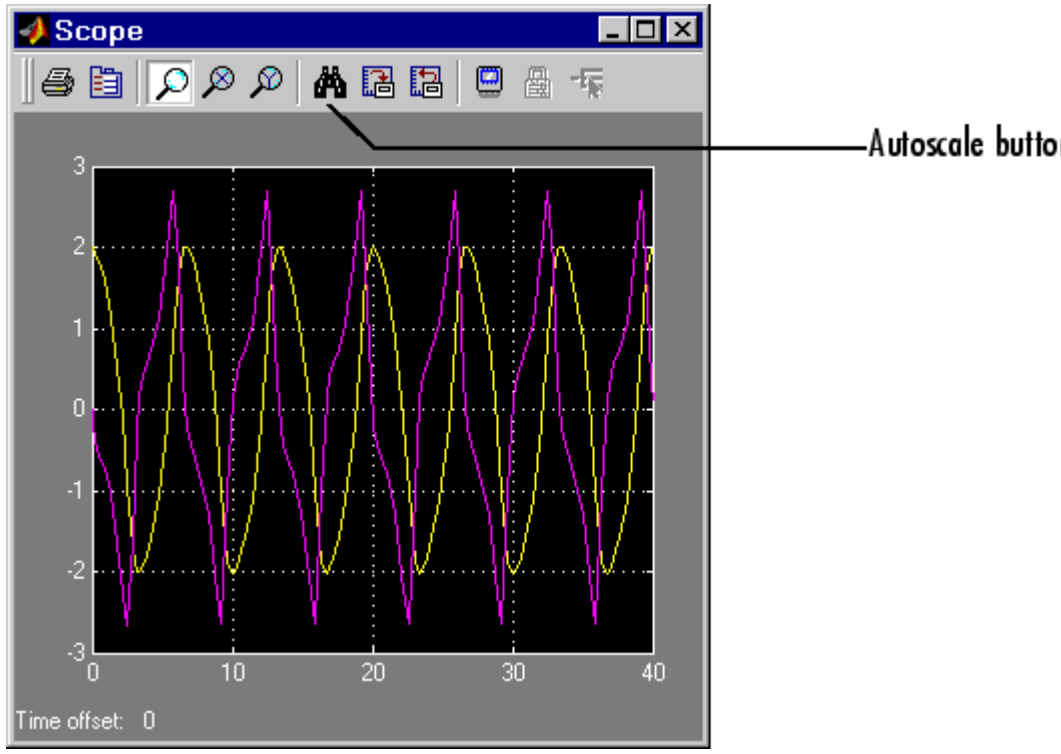
This figure shows the Scope block displaying the output of the `vdp` model. The simulation was run for `40` seconds. Note that this scope shows the final `20` seconds of the simulation. The **Time offset** field displays the time corresponding to `0` on the horizontal axis. Thus, you

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 software 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 software disables (grays) the **Zoom** button on the toolbar of a floating Scope to indicate that it cannot scale its contents.

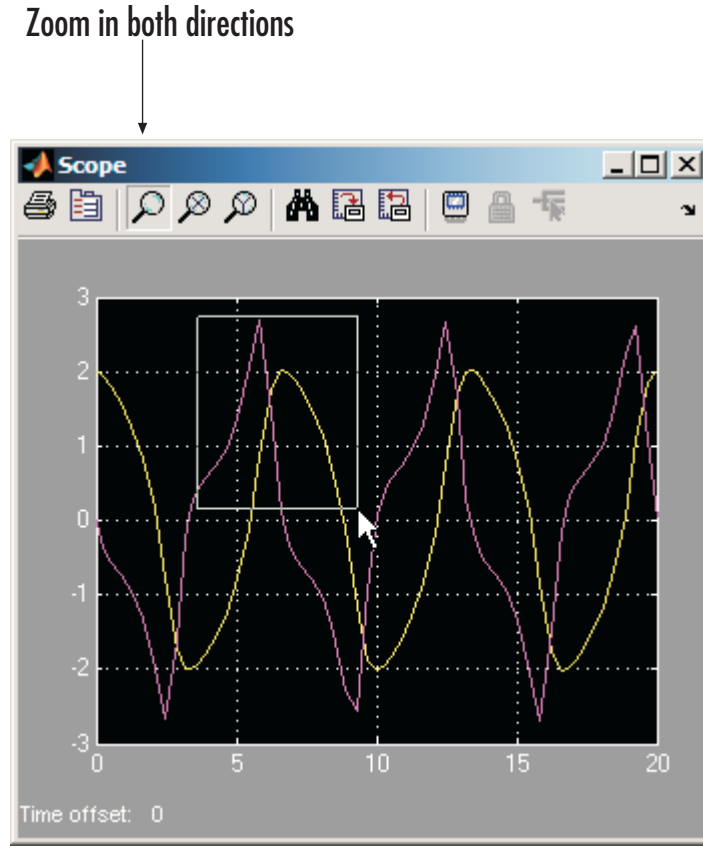
Zooming

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.

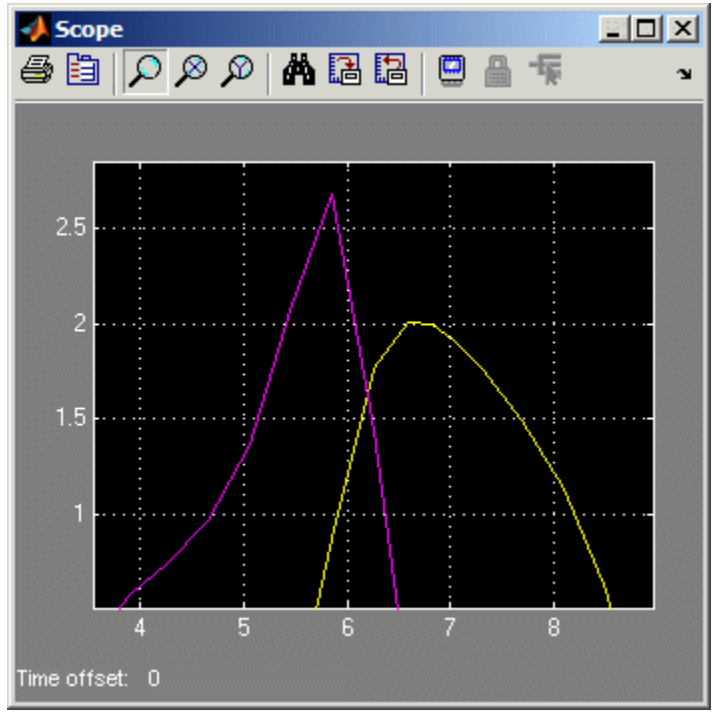
To zoom in on data in both directions at the same time, make sure you select the leftmost **Zoom** toolbar button. Then, define the zoom region using a bounding box. When you release the mouse button, the Scope displays the data in that area. You can also click a point in the area you want to zoom in on.

If the scope has multiple *y*-axes, and you zoom in on one set of *x*-*y* axes, the *x*-limits on all sets of *x*-*y* axes are changed so that they match, because all *x*-*y* axes must share the same time base (*x*-axis).

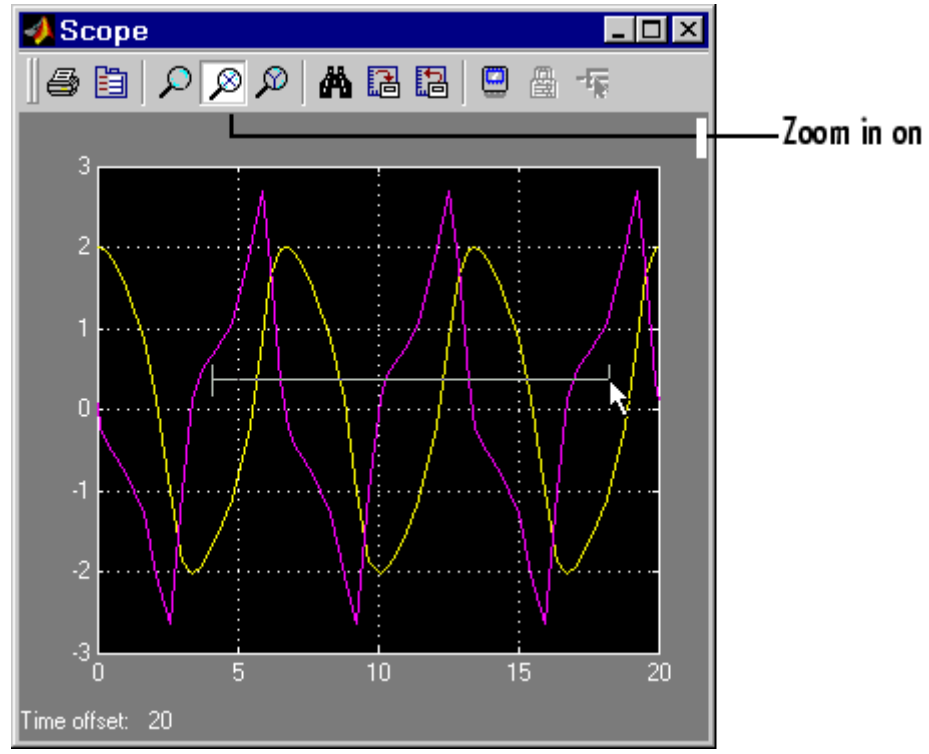
This figure shows a region of the displayed data enclosed within a bounding box.



This figure shows the zoomed region, which appears after you release the mouse button.



To zoom in on data in just the *x* direction, click the middle **Zoom** toolbar button. Define the zoom region by positioning the pointer at one end of the region, pressing and holding down the mouse button, then moving the pointer to the other end of the region. This figure shows the Scope after you define the zoom region, but before you release the mouse button.



When you release the mouse button, the Scope displays the magnified region. You can also click a point in the area you want to zoom in on.

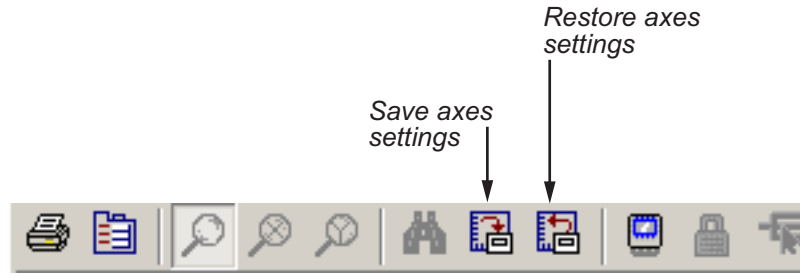
Zooming in the *y* direction works the same way except that you click the rightmost **Zoom** toolbar button before defining the zoom region. Again, you can also click a point in the area you want to zoom in on.

Note Simulink software does not buffer the data that it displays on a floating scope. It therefore cannot zoom the contents of a floating scope. To indicate this, Simulink software disables (grays) the **Zoom** button on the toolbar of a floating scope.

Saving and Restoring the Axes Settings

The **Save axes settings** toolbar button enables you to store the current *x*- and *y*-axes settings so you can apply them to the next simulation.

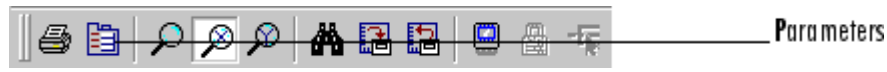
Use the **Restore axes settings** button to restore the saved settings.



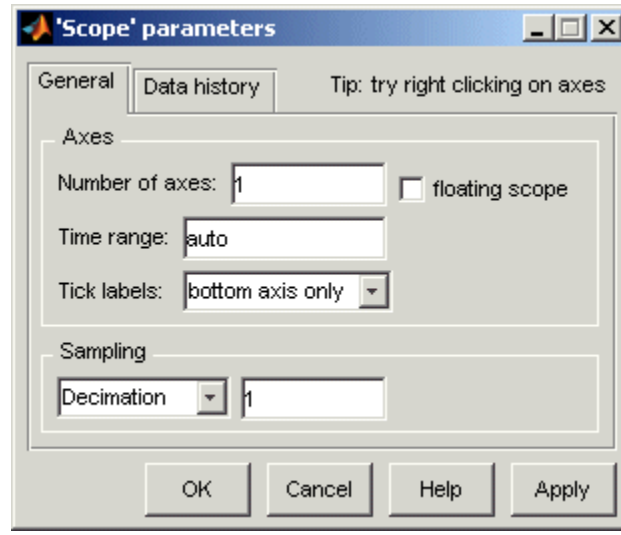
Tip If you select the Save axes settings button on the toolbar, the block specifies its current *y*-limits as the values of the **Y-min** and **Y-max** parameters (see "Y-Axis Limits" on page 2-1131). Similarly, the block specifies its current *x*-axis limits as the value of the **Time range** parameter (see "General Parameters Pane" on page 2-1139).

Scope Parameters

The **Scope Parameters** dialog box lets you change axis limits, set the number of axes, time range, tick labels, sampling parameters, and saving options. To display the dialog, select the **Parameters** button on the toolbar of the Scope block's display



or double-click the Scope viewer's display.



For information on the **General** pane, see "General Parameters Pane" on page 2-1139

For information on the **Data history** pane, see "Data History Parameters Pane" on page 2-1144

General Parameters Pane

You set the axis parameters, time range, tick labels and decimation or sample time in the **General** pane.

Number of axes

Set the number of *y*-axes in this data field. With the exception of the floating scope, there is no limit to the number of axes the Scope block can contain. All axes share the same time base (*x*-axis), but have independent *y*-axes. Note that the number of axes is equal to the number of input ports.

Time range

Change the *x*-axis limits by entering a number or `auto` in the **Time range** field. Entering a number of seconds causes each screen to display the amount of data that corresponds to that

number of seconds. Enter `auto` to set the *x*-axis to the duration of the simulation. Do not enter variable names in these fields.

Tick labels

Specifies whether to label axes ticks. The options are:

`all`	Label ticks on the outside of all axes
`inside`	Place tick labels inside all axes (available only on scope viewers)
`bottom-axis only`	Place tick labels outside the bottom (or only) axes
`none`	Do not label ticks

Sampling

Use this control to select either a **Decimation** factor or **Sample time** interval. Once the selection has been made, enter a number in the data field.

Floating scope

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 **Floating scope**, or by copying the Floating Scope block from the Sinks library into the model window.

To add signals to a floating scope during simulation, you can either click on signals in your block diagram, or use the Signal Selector (for more information on the signal selector, see "The Signal Selector") .

To add signals to a floating scope while the simulation is running by clicking on signals:

- Open the scope

- Select the line to display the signals it carries

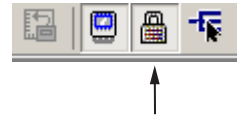
  It might be necessary to click the **Autoscale data** button on the floating scope's toolbar to display the signal

- You can add multiple lines by holding down the **Shift** key while clicking another line

---

Note For you to add signals, the floating scope must have its axes unlocked.

Click the Axes lock icon to lock and unlock the axes.



The axes are highlighted in blue when they are unlocked.

To use the Signal Selector to add signals:

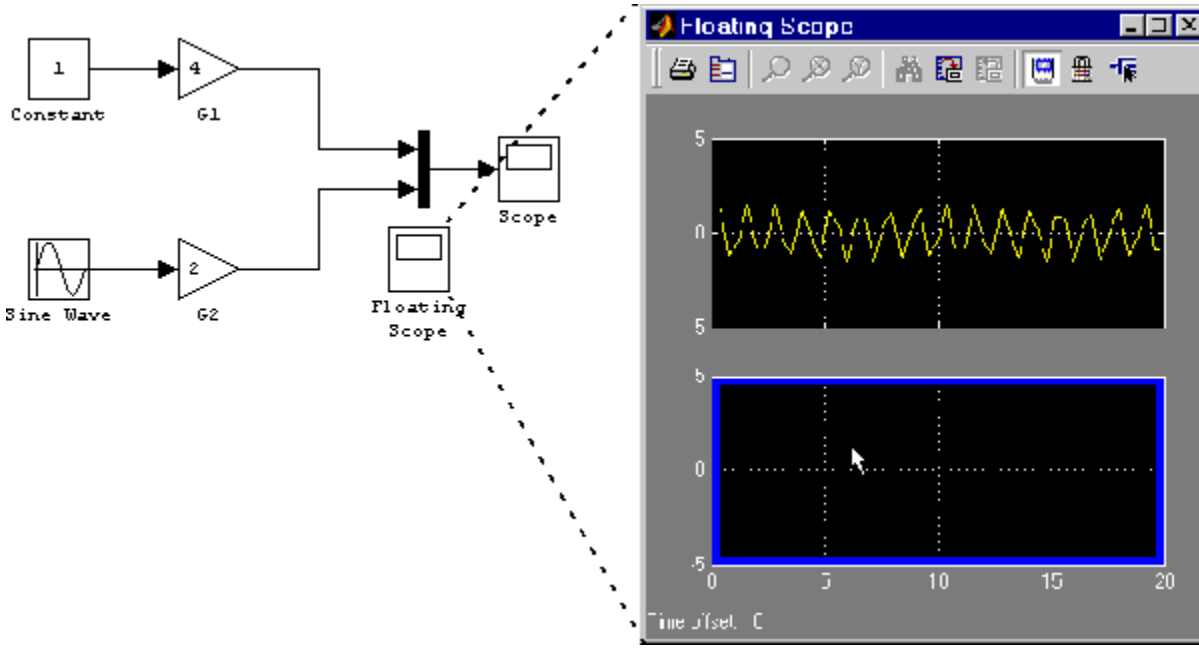
- Open the floating scope
- Right-click your mouse in the floating scope and select **Signal Selection** from the pop-up menu
- From the displayed list, select the signals to be added to the floating scope

  It might be necessary to click the **Autoscale data** button on the floating scope's toolbar to display the signal

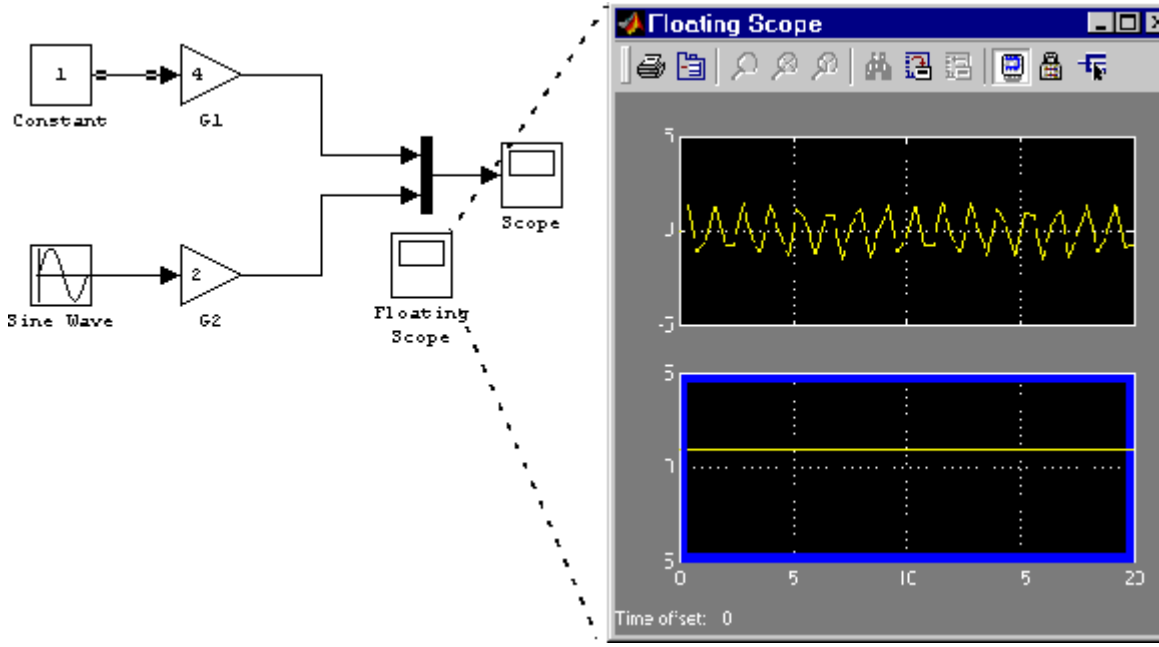
You can have more than one floating scope in a model, but only one set of axes in one scope can be active at a given time. Active floating scopes show the active axes by making them blue. Selecting or deselecting lines affects the active floating scope only.

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 software draws a blue border around the axis.

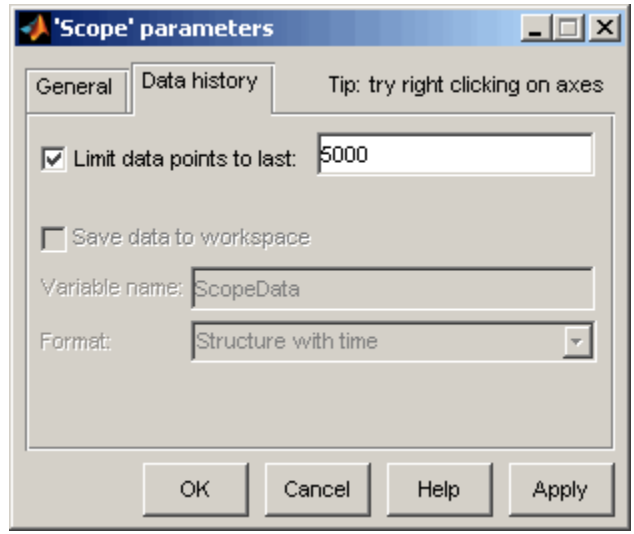


Then click the signal you want to display in the block diagram or the Signal Selector. When you run the model, the selected signal appears in the selected axis.



If you plan to use a floating scope during a simulation, you should disable signal storage reuse. See "Signal storage reuse" in “Optimization Pane” for more information.

Data History Parameters Pane



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.

Note When using a floating scope, **Save data to workspace** is disabled to show that data logging is not supported.

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.

Printing the Contents of a Scope Window

To print the contents of a Scope window, open the **Print** dialog box by clicking the **Print** icon, the leftmost icon on the Scope toolbar.

Creating an Editable Figure from a Scope Block

To create a figure that looks identical to the Scope window but can be annotated using the Plot Editing Tools, use the `simplot` command. Only Scope blocks that save data to the MATLAB workspace from the **Data history** pane are compatible with this command. For example, on the **Data history** pane for the Scope block in `vdp.mdl`, check the **Save data to workspace** option and select `Structure with time` from the

Format list. After running the simulation, a figure can be created with the command

```
simplot(ScopeData)
```

Data Type Support

The Scope block accepts real signals of any data type supported by Simulink software, including fixed-point and enumerated data types. The Scope block accepts homogeneous vectors.

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

Characteristics

Sample Time	Inherited from driving block or can be set
States	0
Multidimensionalized	Yes

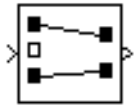
Purpose

Select input elements from vector, matrix, or multidimensional signal

Library

Signal Routing

Description



The Selector block generates as output selected or reordered elements of an input vector, matrix, or multidimensional signal.

A Selector block accepts vector, matrix, or multidimensional signals as input. The parameter dialog box and the block's appearance change to reflect the number of dimensions of the input.

Based on the value you enter for the **Number of input dimensions** parameter, a table of indexing settings is displayed. Each row of the table corresponds to one of the input dimensions in **Number of input dimensions**. For each dimension, you define the elements of the signal to work with. Specify a vector signal as a 1-D signal and a matrix signal as a 2-D signal. When you configure the Selector block for multidimensional signal operations, the block icon changes.

For example, assume a 5-D signal with a one-based index mode. The table of the Selector block dialog changes to include one row for each dimension. If you define each dimension with the following entries:

- **1**

  Index Option, select `Select all`

- **2**

  Index Option, select `Starting index (dialog)`

  Index, enter `2`

  Output Size, enter `5`

- **3**

  Index Option, select `Index vector (dialog)`

  Index, enter `[1 3 5]`

- **4**

Index Option, select `Starting index (port)`

Output Size, enter `8`

- **5**

  Index Option, select `Index vector (port)`

The output will be `Y=U(1:end,2:6,[1 3 5],Idx4:Idx4+7,Idx5)`, where `Idx4` and `Idx5` are the index ports for dimensions 4 and 5.

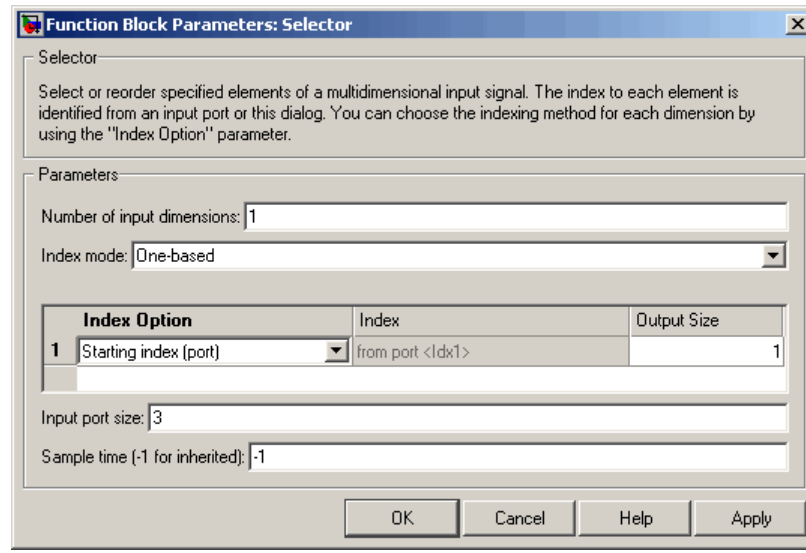
Data Type Support

The data port of the Selector block accepts signals of any signal type and any data type supported by Simulink software, including fixed-point and enumerated data types. The data port accepts mixed-type signals. The index port accepts only built-in data types, except boolean data types. The elements of the output have the same type as the corresponding selected input elements.

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

Parameters and Dialog Box

The parameter dialog box appears as follows when you set **Index Option** to `Starting index (port)`.



Number of input dimensions

Enter the number of dimensions of the input signal.

Index mode

Specifies the indexing mode: `One-based` or `Zero-based`. If `One-based` is selected, an index of 1 specifies the first element of the input vector, 2, the second element, and so on. If `Zero-based` is selected, an index of 0 specifies the first element of the input vector, 1, the second element, and so on.

Index Option

Define, by dimension, how the elements of the signal are to be indexed. From the list, choose:

- `Select all`

  This is the default. No further configuration is required. All elements are selected.

- `Index vector (dialog)`

  Enables the **Index** column. Enter the vector of indices of the elements.

- `Index vector (port)`

  No further configuration is required.

- `Starting index (dialog)`

  Enables the **Index** and **Output Size** columns. Enter the starting index of the range of elements to be selected in the **Index** column and the number of elements to be selected in the **Output Size** column.

- `Starting index (port)`

  Enables the **Output Size** column. Enter the number of elements to be selected in the **Output Size** column.

The **Index** and **Output Size** columns appear as relevant.

Index

If the **Index Option** is `Index vector (dialog)`, enter the index of each element in which you are interested.

If the **Index Option** is `Starting index vector (dialog)`, enter the starting index of the range of elements to be selected.

Output Size

Enter the width (number of elements from the starting point) of the block output signal.

Input port size

Specify the width of the block input signal (`-1` for inherited) — 1-D signals only.

Sample time (-1 for inherited)

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

Note For 1-D signals, the **Sample time** parameter is applicable only with the **Index Option** set to `Starting index (port)` or `Index vector (port)`. For all other **Input Option** settings, the Selector block becomes a virtual block and the **Sample time** parameter does not appear.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	Yes
Multidimensionalized	Yes
Virtual	Yes, when **Number of input dimensions** is `1` and **Index Option** is `Select all`, `Index vector (dialog)`, or `Starting index (dialog)` For more information, see "Virtual Blocks" in the Simulink documentation.
Zero crossing	No

S-Function

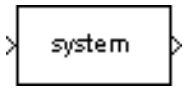
Purpose

Include S-function in model

Library

User-Defined Functions

Description



The S-Function block provides access to S-functions from a block diagram. The S-function named as the **S-function name** parameter can be a Level-1 M-file or a Level-1 or Level-2 C MEX-file S-function (see "Overview of S-Functions" in *Writing S-Functions* for information on how to create S-functions).

Note Use the M-File S-Function block to include a Level-2 M-file S-function in a block diagram.

The S-Function block allows additional parameters to be passed directly to the named S-function. The function parameters can be specified as MATLAB expressions or as variables separated by commas. For example,

```
A, B, C, D, [eye(2,2);zeros(2,2)]
```

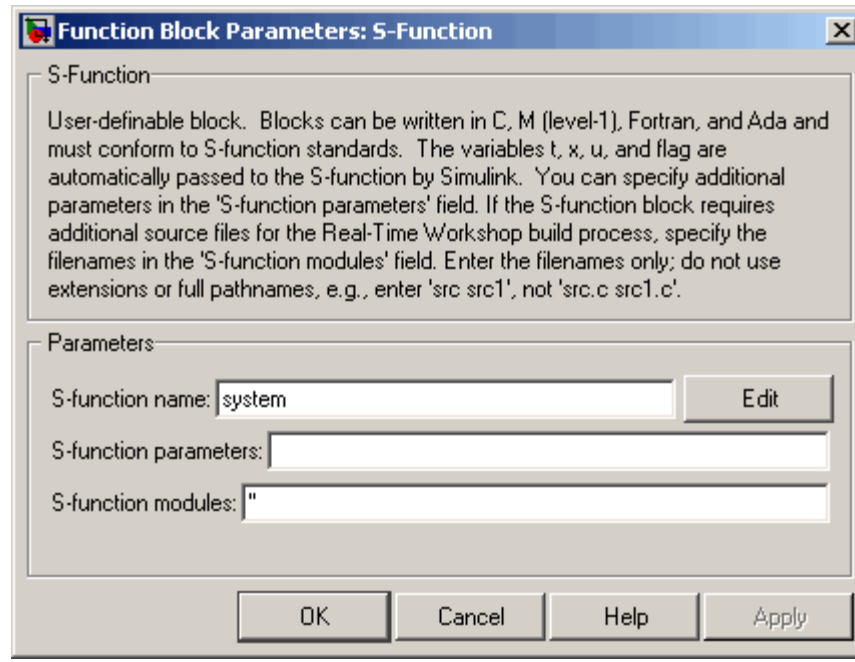
Note that although individual parameters can be enclosed in brackets, the list of parameters must not be enclosed in brackets.

The S-Function block displays the name of the specified S-function and the number of input and output ports specified by the S-function. Signals connected to the inputs must have the dimensions specified by the S-function for the inputs.

Data Type Support

Depends on the implementation of the S-Function block.

Parameters and Dialog Box



S-function name
The S-function name.

S-function parameters
Additional S-function parameters. See the preceding block description for information on how to specify the parameters.

S-function modules
This parameter applies only if this block represents a C MEX-file S-function and you intend to use the Real-Time Workshop software to generate code from the model containing the block. See “Specifying Additional Source Files for an S-Function” in the Real-Time Workshop online documentation for information on using this parameter.

Characteristics

Direct Feedthrough	Depends on contents of S-function
Sample Time	Depends on contents of S-function
Scalar Expansion	Depends on contents of S-function
Dimensionalized	Depends on contents of S-function
Multidimensionalized	Yes
Zero Crossing	No

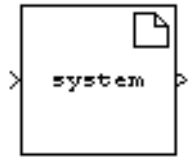
Purpose

Create 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 software invokes the generated S-function associated with each instance to compute the instance's output at each time step.

Note The S-Function Builder block does not support masking. However, you can mask a Subsystem block that contains an S-Function Builder block. See "Working with Block Masks" in the Simulink documentation for more information.

Data Type Support

The S-Function Builder can accept and output complex, 1-D, or 2-D signals and nonvirtual buses. For each of these cases, the signals must have a data type which Simulink software supports.

For a discussion on the data types supported by Simulink software, 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 bits or binary point of signal

Library

Logic and Bit Operations

Description

The Shift Arithmetic block can 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

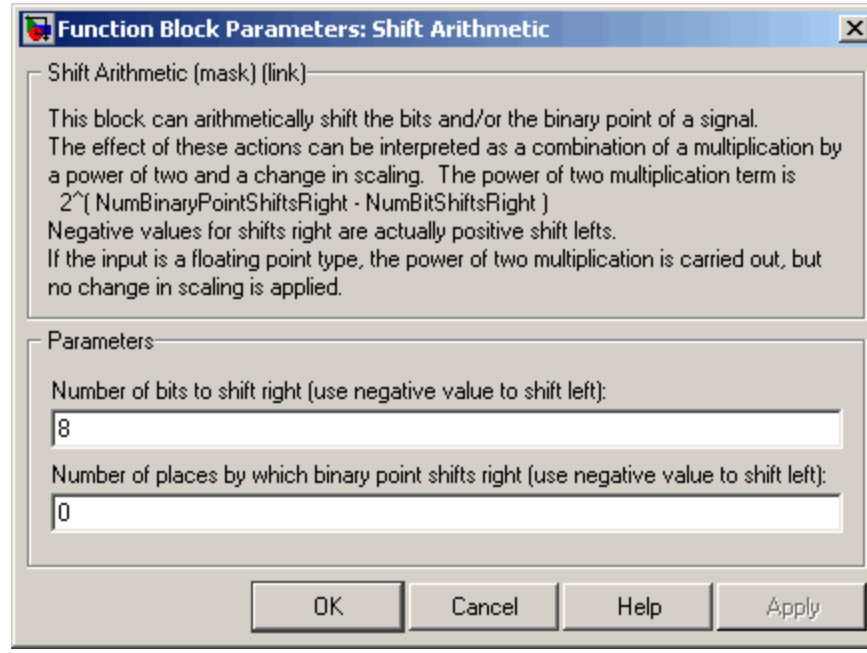
This block performs arithmetic bit shifts on signed numbers. Therefore, the most significant bit is recycled for each bit shift. The effects of bit shifts two places to the right and two places to the left on an input of data type `sfix(8)` follow.

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 numeric data type supported by Simulink software, including fixed-point data types, but not `boolean` type.

Parameters and Dialog Box



Number of bits to shift right

The number of places to shift the bits of the input signal. A positive value indicates a shift right, while a negative value indicates a shift left.

Number of places by which binary point shifts right

The number of places to shift the binary point of the input signal. A positive value indicates a shift right, while a negative value indicates a shift left.

Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from the driving block
Scalar Expansion	Yes

Sign

Purpose

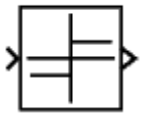
Indicate sign of input

Library

Math Operations

Description

The Sign block indicates the sign of the input:



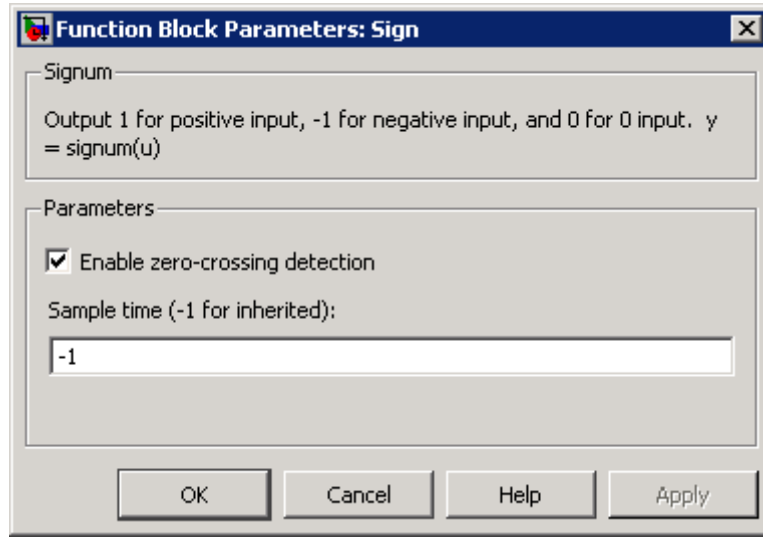
- The output is 1 when the input is greater than zero.
- The output is 0 when the input is equal to zero.
- The output is -1 when the input is less than zero.

Data Type Support

The Sign block accepts real signals of any numeric data type supported by Simulink software, including fixed-point data types. The output is a signed data type with the same number of bits as the input, and with nominal scaling (a slope of one and a bias of zero).

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

Parameters and Dialog Box



Enable zero-crossing detection

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

Sample time (-1 for inherited)

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

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	N/A
Dimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled.

Signal Builder

Purpose

Create and generate interchangeable groups of signals whose waveforms are piecewise linear

Library

Sources

Description

The Signal Builder block allows you to create interchangeable groups of piecewise linear signal sources and use them in a model. See "Working with Signal Groups" in the "Working with Signals" chapter of the Simulink documentation.

Note Use the `signalbuilder` function to create and access Signal Builder blocks programmatically.

Data Type Support

The Signal Builder block outputs a scalar or array of real signals of type `double`.

Parameters and Dialog Box

The Signal Builder block has the same dialog box as that of a Subsystem block. To display the dialog box, select **Subsystem Parameters** from the block's context menu.

Characteristics

Sample Time	Continuous
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Zero Crossing	Yes

Purpose

Convert signal to new type without altering signal values

Library

Signal Attributes

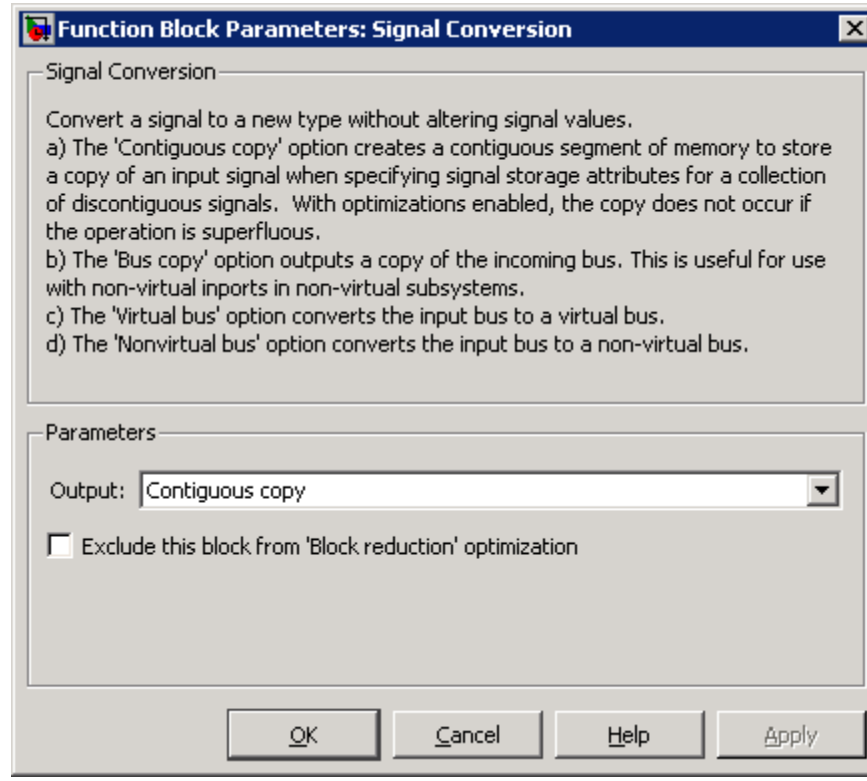
Description

The Signal Conversion block converts a signal from one type to another. Use the **Output** parameter to select the type of conversion to perform.

Data Type Support

The Signal Conversion block accepts virtual or nonvirtual signals of any data type.

Parameters and Dialog Box



Output

Specifies the type of conversion to perform. The options are:

- `Contiguous copy`

  Converts a muxed signal, whose elements occupy discontiguous areas of memory, to a vector signal, whose elements occupy contiguous areas of memory. The block does this by allocating a contiguous area of memory for the elements of the muxed signal and copying the values from the discontiguous areas (represented by the block's input) to the contiguous areas (represented by the block's output) at each time step. You can also use this setting to connect a block with a constant sample time to an output port of an enabled subsystem. For more information, see Using Blocks with Constant Sample Times in Enabled Subsystems. For an example involving Real-Time Workshop software, see "Reusable Code and Referenced Models".

- `Bus copy`

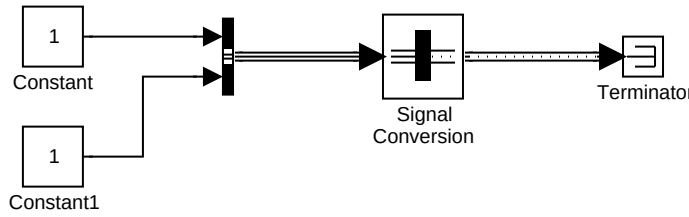
  Outputs a copy of the bus connected to the block's input. This setting is useful when passing a bus signal whose components have different data types to a nonvirtual Inport block in an Atomic Subsystem. See "Using Composite Signals" in the Simulink documentation for more information.

- `Virtual bus`

  Converts a nonvirtual bus to a virtual bus. In general, virtual buses can save memory where nonvirtual buses are not required.

- `Nonvirtual bus`

  Converts a virtual bus to a nonvirtual bus as in the following example. This setting is useful when passing a virtual bus signal to a modeling construct that requires a nonvirtual bus, such as a Model block.



Note The virtual bus to be converted to a nonvirtual bus must be defined by a bus object, for example, an instance of `Simulink.Bus` class. See the `Bus Creator` block for more information.

Exclude this block from 'Block reduction' optimization

This option is available only for `Contiguous copy` conversion. If the elements of the input signal occupy contiguous areas of memory, Simulink software eliminates the block from the compiled model as an optimization. That action does not occur when you select this check box. For more information, see "Block reduction" in the Simulink documentation.

Characteristics

Sample Time	Inherited from the driving block
Scalar Expansion	n/a
Dimensionalized	n/a
Multidimensionalized	Yes
Zero-Crossing Detection	No

Signal Generator

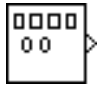
Purpose

Generate various waveforms

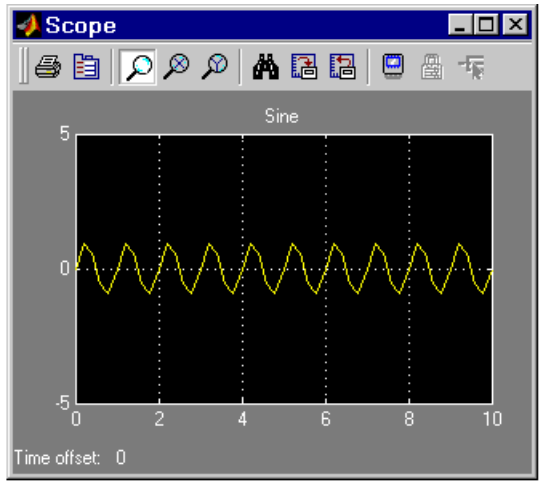
Library

Sources

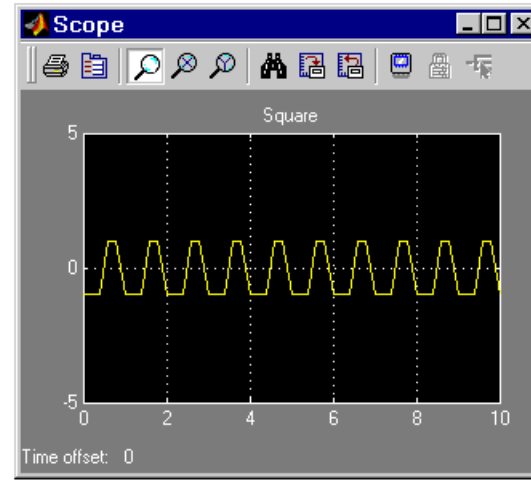
Description



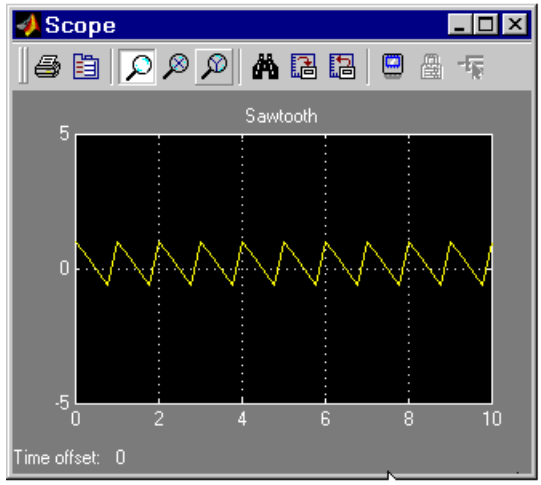
The Signal Generator block can produce one of four different waveforms: sine wave, square wave, sawtooth wave, and random wave. The signal parameters can be expressed in Hertz (the default) or radians per second. This figure shows each signal displayed on a Scope using default parameter values.



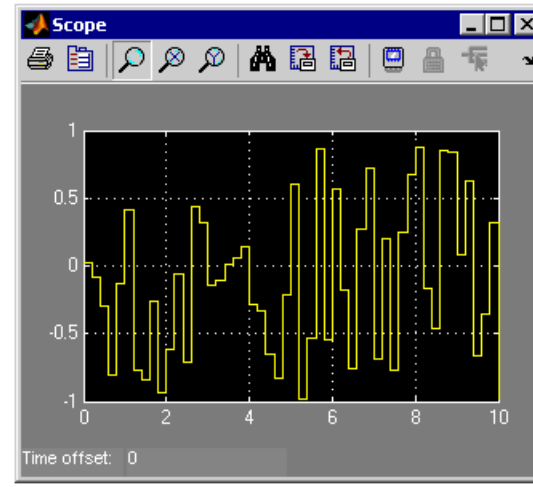
Sine Wave



Square Wave



Sawtooth Wave



Random Wave

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.

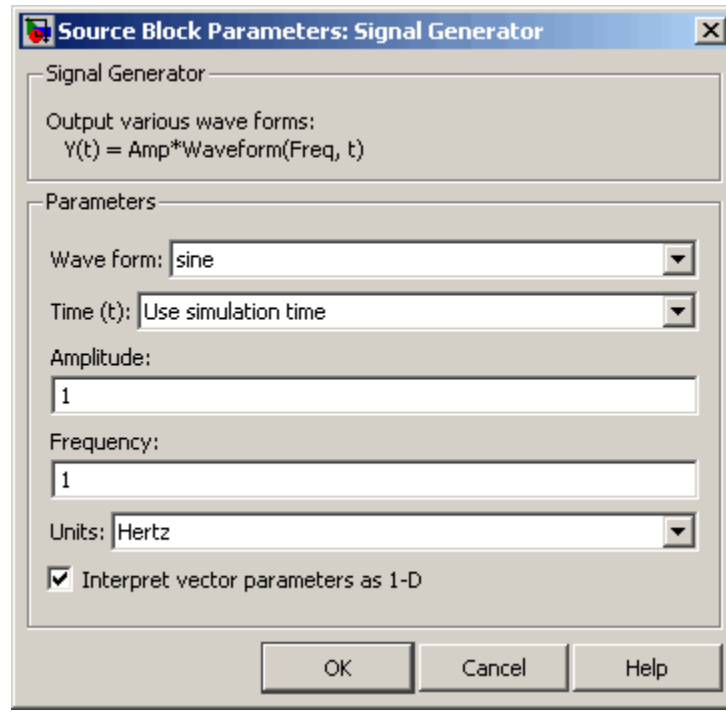
You can vary the output settings of the Signal Generator block while a simulation is in progress. This is useful to determine quickly the response of a system to different types of inputs.

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 During Simulation" in the online Simulink documentation for details.

Wave form

The wave form: a sine wave, square wave, sawtooth wave, or random 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 (see "Determining the Output Dimensions of Source Blocks" in the "Working with Signals" chapter of the Simulink documentation). This option is not available when an external signal specifies time. In this case, if the **Amplitude** and **Frequency** parameters are column or row matrix values, the output is a 1-D vector.

Characteristics

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

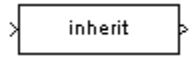
Purpose

Specify desired dimensions, sample time, data type, numeric type, and other attributes of 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 software 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.

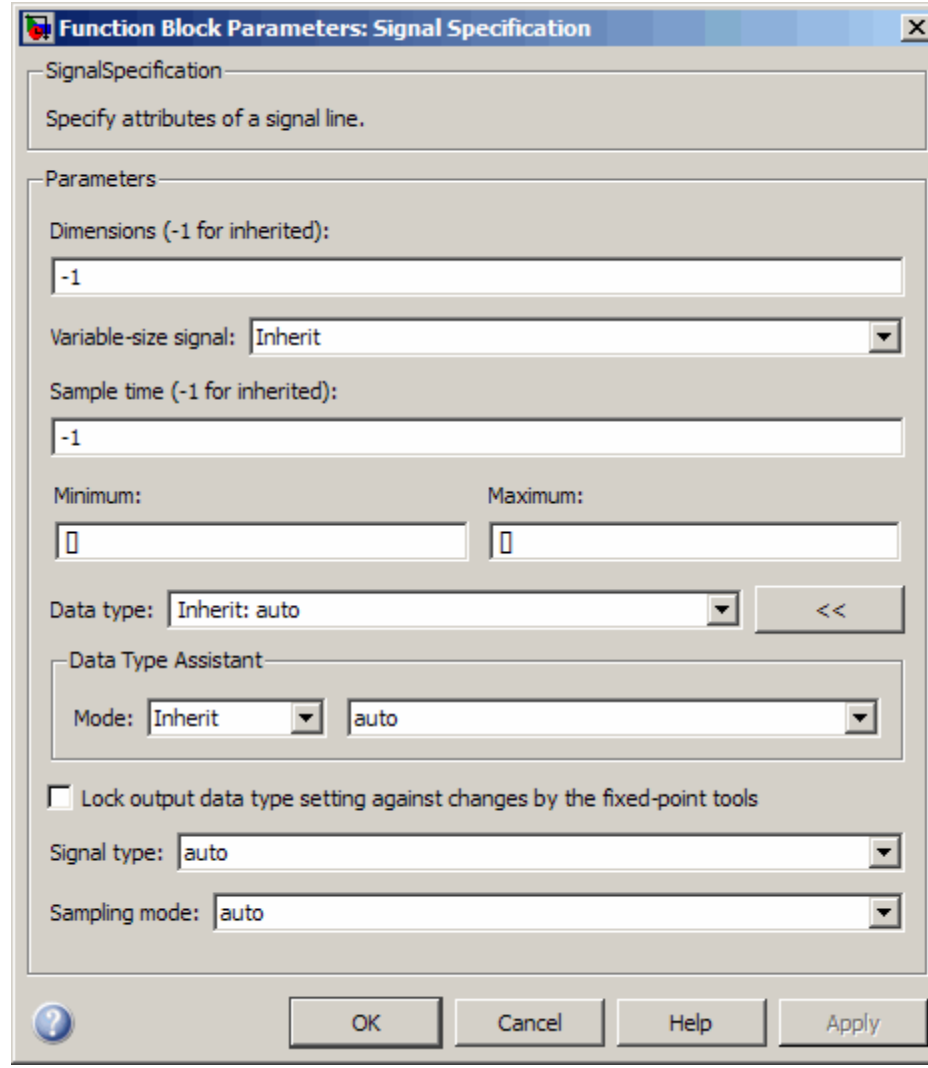
You can use the Signal Specification block to ensure that the actual attributes of a signal meet desired attributes. For example, suppose that you and a colleague are working on different parts of the same model. You use Signal Specification blocks to connect your part of the model with your colleague's. If your colleague changes the attributes of a signal without informing you, the attributes entering the corresponding Signal Specification block do not match. When you try to simulate the model, you get an error.

You can also use the Signal Specification block to ensure correct propagation of signal attributes throughout a model. The capability of allowing the Simulink to propagate attributes from block to block is powerful. However, if some blocks have unspecified attributes for the signals they accept or output, the model does not have enough information to propagate attributes correctly. 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 thatSimulink supports, including fixed-point and enumerated data types. The input data type must match the data type specified

by the **Data type** parameter. For more information, see “Data Types Supported by Simulink” in the Simulink documentation.



Parameters and Dialog Box

- "Dimensions (-1 for inherited)" on page 2-1173
- "Variable-size signal" on page 2-1174

Dimensions (-1 for inherited)

Specify the dimensions of the input and output signals.

Settings

Default: -1

`-1`
: Specifies that signals inherit dimensions.

`n`
: Specifies vector signal of width `n`.

`[m n]`
: Specifies matrix signal having `m` rows and `n` columns.

Command-Line Information

Parameter: `Dimensions`
Type: string
Value: `'-1' | n | [m n |`
Default: `'-1'`

Variable-size signal

Specify a variable-size signal, fixed-size signal, or both.

Settings

Default: `Inherit`

`Inherit`
: Allows variable-size and fixed-size signals.

`No`
: Does not allow variable-size signals.

`Yes`
: Allows only variable-size signals.

Dependencies

When the signal is a variable-size signal, the **Dimensions** parameter specifies the maximum dimensions of the signal.

If you specify a bus object, the simulation allows variable-size signals only with a disabled bus object.

Command-Line Information

Parameter: `VarSizeSig`
Type: string
Value: `'Inherit'| 'No' | 'Yes'`
Default: `'Inherit'`

See Also

"Working with Variable-Size Signals"

Sample time (-1 for inherited)

Specify the time interval when the simulation updates the block.

Settings

Default: -1

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

Command-Line Information

Parameter: `SampleTime`
Type: string
Value: Any valid sample time
Default: `'-1'`

See Also

"How to Specify the Sample Time"

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Signal type

Specify the numeric type of the input and output signals.

Settings

Default: `auto`

`auto`
: Accepts either `real` or `complex` as the numeric type.

`real`
: Specifies the numeric type as a real number.

`complex`
: Specifies the numeric type as a complex number.

Command-Line Information

Parameter: `SignalType`
Type: string
Value: `'auto' | 'real' | 'complex'`
Default: `'auto'`

Sampling mode

Select the sampling mode for this block.

Settings

Default: `auto`

`auto`
: Accepts any sampling mode.

`Sample based`
: Specifies the output signal to be sample-based.

`Frame based`
: Specifies the output signal to be frame-based.

Tips

To generate frame-based signals, you must have the Signal Processing Blockset product installed.

Command-Line Information

Parameter: `SamplingMode`
Type: string
Value: `'auto' | 'Sample based' | 'Frame based'`
Default: `'auto'`

Minimum

Specify the minimum value for the block output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`. Simulink uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tips

This number must be a double scalar value.

Command-Line Information

Parameter: `OutMin`
Type: string
Value: Any valid scalar value
Default: `'[]'`

Maximum

Specify the maximum value for the block output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`. Simulink uses this value to perform:

- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tips

This number must be a double scalar value.

Command-Line Information

Parameter: `OutMax`
Type: string
Value: Any valid double scalar value
Default: `'[]'`

Data type

Specify the output data type.

Settings

Default: `auto`

`Inherit: auto`
: Inherits data type

`double`
: Sets data type `double`.

`single`
: Sets data type `single`.

`int8`
: Sets data type is `int8`.

`uint8`
: Sets data type `uint8`.

`int16`
: Sets data type `int16`.

`uint16`
: Sets data type `uint16`.

`int32`
: Sets data type `int32`.

`uint32`
: Sets data type `uint32`.

`boolean`
: Sets data type `boolean`.

`fixdt(1,16,0)`
: Sets data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Sets data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
Specifies the data type as enumerated.

`<data type expression>`
Specifies the name of a data type object, for example,`Simulink.NumericType`

Dependencies

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Data type** parameters.

Command-Line Information

Parameter: `OutDataTypeStr`
Type: string
Value: `'Inherit: auto' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum:<class name>' | <data type expression> |`
Default: `'Inherit: auto'`

See Also

"Specifying Block Output Data Types" in *Simulink User's Guide*.

Mode

Select the category of data to specify.

Settings

Default: `Inherit`

`Inherit`

Specifies inheritance rules for data types. Selecting `Inherit` enables a list of possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Specifies built-in data types. Selecting `Built in` enables a list of possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`
- `int8`
- `uint8`
- `int16`

- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
Specifies fixed-point data types.

`Enumerated`
Specifies enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a list of possible values, which can vary by block:

- `<class name>`

`Expression`
Specifies expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. Following are the possible values, which can vary by block:

- `<data type expression>`

Dependencies

Clicking the **Show data type assistant** button 

enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specifies fixed-point data as signed.

`Unsigned`
: Specifies the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Specifying a Fixed-Point Data Type" in the *Simulink User's Guide*.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Specifying a Fixed-Point Data Type" in the *Simulink User's Guide*.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Binary point`

`Binary point`
: Specifies binary point location.

`Slope and bias`
: Enters slope and bias.

Dependencies

Selecting `Fixed point` from the **Mode** list enables this parameter.

Selecting `Binary point` from the **Scaling** list enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` from the **Scaling** list enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

See Also

"Specifying a Fixed-Point Data Type" in the *Simulink User's Guide*.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Specifying a Fixed-Point Data Type" in the *Simulink User's Guide*.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Specifying a Fixed-Point Data Type" in the *Simulink User's Guide*.

Bias

Specify bias for the fixed-point data type.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Specifying a Fixed-Point Data Type" in the *Simulink User's Guide*.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified by the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	Yes
Multidimensionalized	Yes
Virtual	Yes For more information, see "Virtual Blocks" in the Simulink documentation.
Zero-Crossing Detection	No

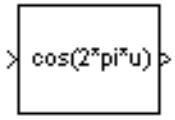
Purpose

Implement sine and/or cosine wave in fixed point using 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 $\cos(2\pi u)$

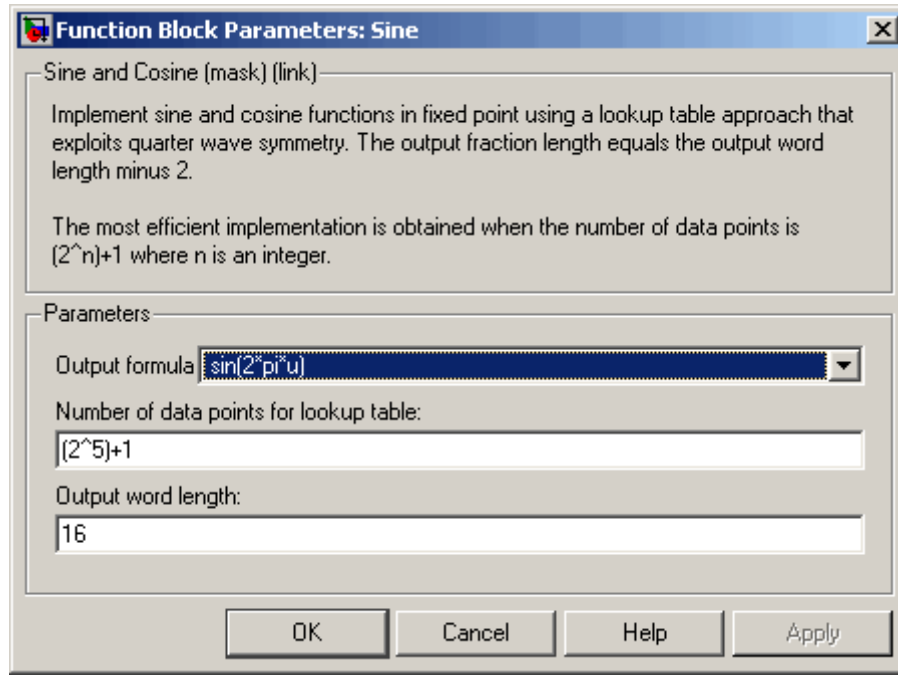
You define the number of lookup table points in the **Number of data points for lookup table** parameter. The block implementation is most efficient when you specify the lookup table data points to be `(2^n)+1`, where `n` is an integer.

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 numeric data type supported by Simulink software, 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

See Also

Sine Wave, Trigonometric Function

Sine Wave

Purpose

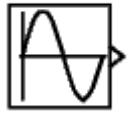
Generate sine wave, using simulation time as time source

Library

Sources

Description

The Sine Wave block provides a sinusoid. The block can operate in time-based or sample-based mode.



Note This block is the same as the Sine Wave Function block that appears in the Math Operations library. If you select `Use external source` for the **Time** parameter in the block dialog box, you get the Sine Wave Function block.

Time-Based Mode

The output of the Sine Wave block is determined by

$$y = Amplitude \times \sin(frequency \times time + phase) + bias$$

Time-based mode has two submodes: continuous mode or discrete mode. The value of the **Sample time** parameter determines whether the block operates in continuous mode or discrete mode:

- `0` (the default) causes the block to operate in continuous mode.
- `>0` causes the block to operate in discrete mode.

See "How to Specify the Sample Time" in the online documentation for more information.

Using the Sine Wave Block in Continuous Mode

A **Sample time** parameter value of `0` causes the block to operate in continuous mode. When operating in continuous mode, the Sine Wave block can become inaccurate due to loss of precision as time becomes very large.

Using the Sine Wave Block in Discrete Mode

A **Sample time** parameter value greater than zero causes the block to behave as if it were driving a Zero-Order Hold block whose sample time is set to that value.

Using the Sine Wave block in this way allows you to build models with sine wave sources that are purely discrete, rather than models that are hybrid continuous/discrete systems. Hybrid systems are inherently more complex and as a result take longer to simulate.

The Sine Wave block in discrete mode uses an incremental algorithm rather than one based on absolute time. As a result, the block can be useful in models intended to run for an indefinite length of time, such as in vibration or fatigue testing.

The incremental algorithm computes the sine based on the value computed at the previous sample time. This method makes use of the following identities:

$$\sin(t+\Delta t) = \sin(t)\cos(\Delta t) + \sin(\Delta t)\cos(t)$$
$$\cos(t+\Delta t) = \cos(t)\cos(\Delta t) - \sin(t)\sin(\Delta t)$$

These identities can be written in matrix form:

$$\begin{bmatrix}\sin(t+\Delta t)\\ \cos(t+\Delta t)\end{bmatrix} = \begin{bmatrix}\cos(\Delta t) & \sin(\Delta t)\\ -\sin(\Delta t) & \cos(\Delta t)\end{bmatrix}\begin{bmatrix}\sin(t)\\ \cos(t)\end{bmatrix}$$

Since Δt is constant, the following expression is a constant:

$$\begin{bmatrix}\cos(\Delta t) & \sin(\Delta t)\\ -\sin(\Delta t) & \cos(\Delta t)\end{bmatrix}$$

Therefore the problem becomes one of a matrix multiplication of the value of `sin(`t`)` by a constant matrix to obtain `sin(`$t+\Delta t$`)`.

Discrete mode reduces but does not eliminate accumulation of roundoff errors. This is because the computation of the block's output at each time step depends on the value of the output at the previous time step.

Sample-Based Mode

Sample-based mode uses the following formula to compute the output of the Sine Wave block.

$$y = A \times \sin(2 \times \pi \times (k + o)/p) + b$$

where

- `A` is the amplitude of the sine wave.
- `p` is the number of time samples per sine wave period.
- `k` is a repeating integer value that ranges from 0 to p-1.
- `o` is the offset (phase shift) of the signal.
- `b` is the signal bias.

In this mode, Simulink software sets `k` equal to `0` at the first time step and computes the block's output, using the preceding formula. At the next time step, Simulink software increments `k` and recomputes the output of the block. When `k` reaches `p`, Simulink software resets `k` to `0` before computing the block's output. This process continues until the end of the simulation.

The sample-based method of computing the block's output does not depend on the result of the previous time step to compute the result at the current time step. It therefore avoids roundoff error accumulation. However, it has one potential drawback. If the block is in a conditionally executed subsystem and the conditionally executed subsystem pauses and then resumes execution, the output of the Sine Wave block might no longer be in sync with the rest of the simulation. Thus, if the accuracy of your model requires that the output of conditionally executed Sine Wave blocks remain in sync with the rest of the model, you should use time-based mode for computing the output of the conditionally executed blocks.

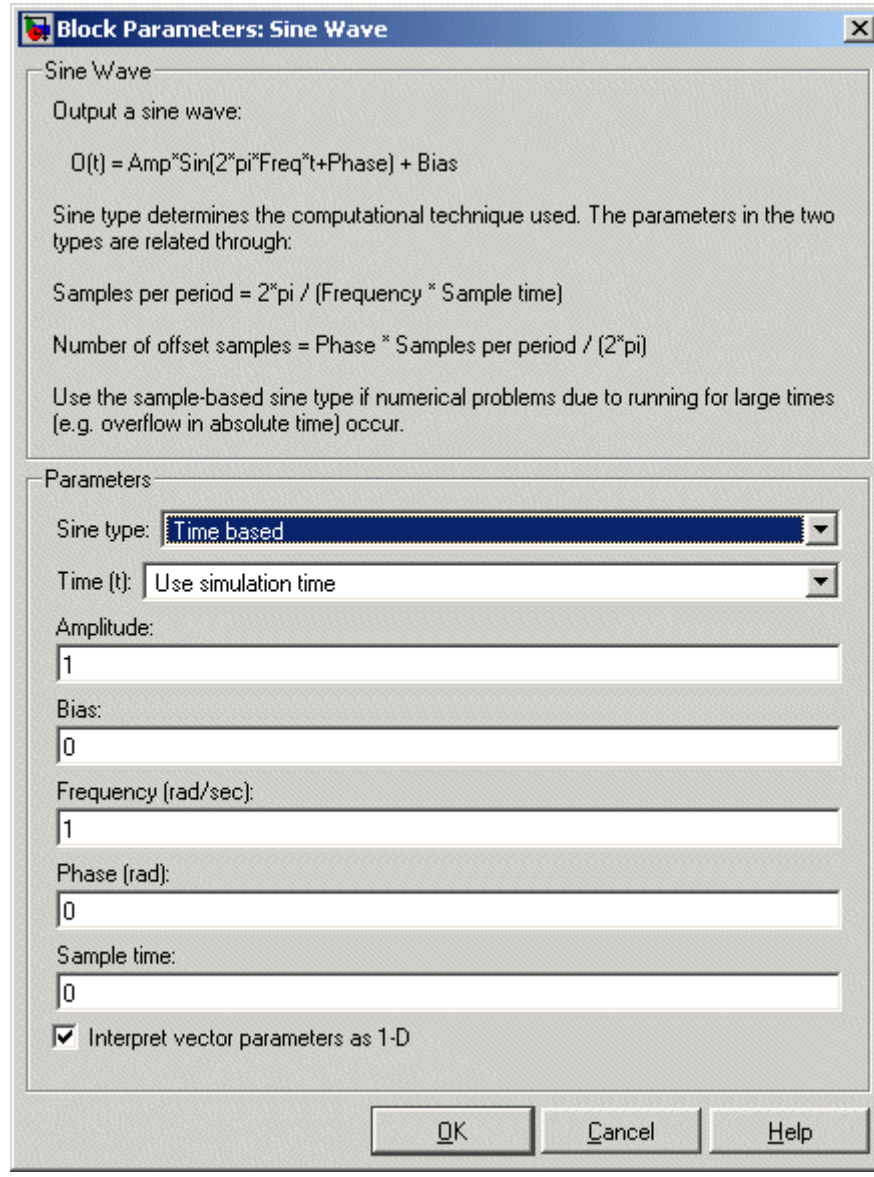
Parameter Dimensions

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 Sine Wave block accepts and 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 During Simulation" in the online Simulink documentation for details.

Sine type
Type of sine wave that this block generates, either time- or sample-based. Some parameters in the dialog box appear depending on whether you select time-based or sample-based.

Time
Specifies whether to use simulation time as the source of values for the sine wave's time variable or an external source. If you 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 per second. The default is `1`. 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`. 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 "How to Specify the Sample Time" in the online documentation for more information.

Interpret vector parameters as 1-D

If selected, column or row matrix values for numeric parameters result in a vector output signal. Otherwise, the block outputs a signal of the same dimensionality as the parameters. If you do not select this check box, the block always outputs a signal of the same dimensionality as the numeric parameters. See "Determining the Output Dimensions of Source Blocks" in the "Working with Signals" chapter of the Simulink documentation. This option is not available when an external signal specifies time. In this case, if numeric parameters are column or row matrix values, the output is a 1-D vector.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Zero-Crossing Detection	No

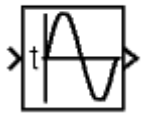
Purpose

Generate sine wave, using external signal as time source

Library

Math Operations

Description



This block is the same as the Sine Wave block that appears in the Sources library. If you select `Use simulation time` for the **Time** parameter in the block dialog box, you get the Sine Wave block. See the documentation for the Sine Wave block for more information.

Slider Gain

Purpose

Vary scalar gain using slider

Library

Math Operations

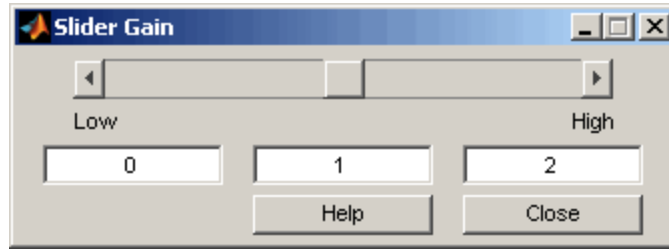
Description

Use the Slider Gain block 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
Multidimensionalized	Yes
Zero Crossing	No

Squeeze

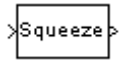
Purpose

Remove singleton dimensions from multidimensional signal

Library

Math Operations

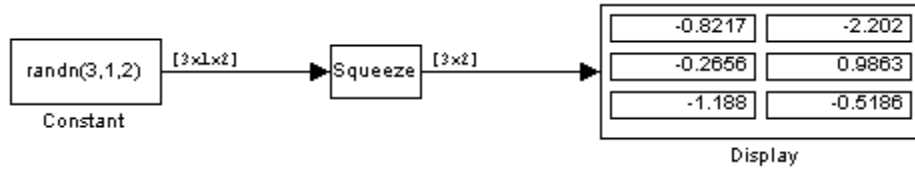
Description



The Squeeze block removes singleton dimensions from its multidimensional input signal. A singleton dimension is any dimension whose size is one.

Note The Squeeze block operates only on signals whose number of dimensions is greater than two. Scalar, one-dimensional (vector), and two-dimensional (matrix) signals pass through the Squeeze block unchanged.

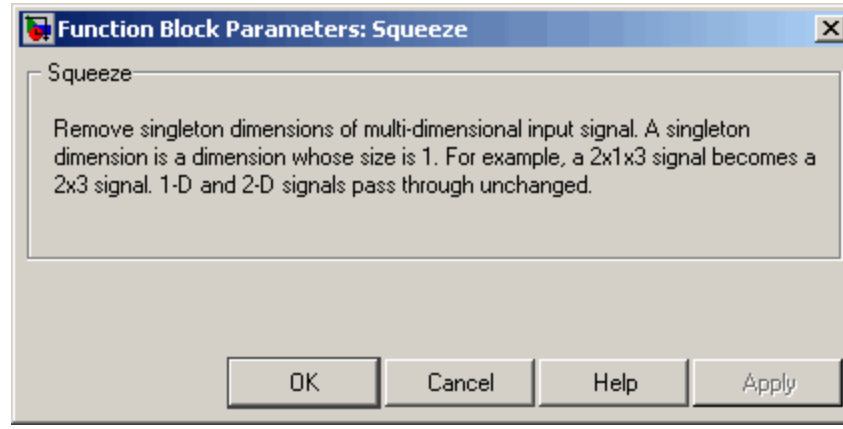
For example, the Squeeze block in the following diagram converts a multidimensional array of size 3-by-1-by-2 into a 3-by-2 signal.



Data Type Support

The Squeeze block accepts input signals of any dimension and of any data type that Simulink software supports, including fixed-point and enumerated data types. For a discussion on the data types supported by Simulink software, see “Data Types Supported by Simulink”.

Parameters and Dialog Box



Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	N/A
Dimensionalized	Yes
Multidimensionalized	Yes
Zero Crossing	No

See Also

Reshape

State-Space

Purpose

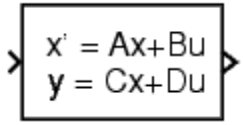
Implement linear state-space system

Library

Continuous

Description

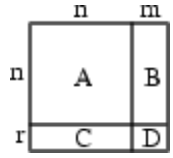
The State-Space block implements a system whose behavior you define as



$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

where x is the state vector, u is the input vector, and y is the output vector. The matrix coefficients must have these characteristics:

- **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.



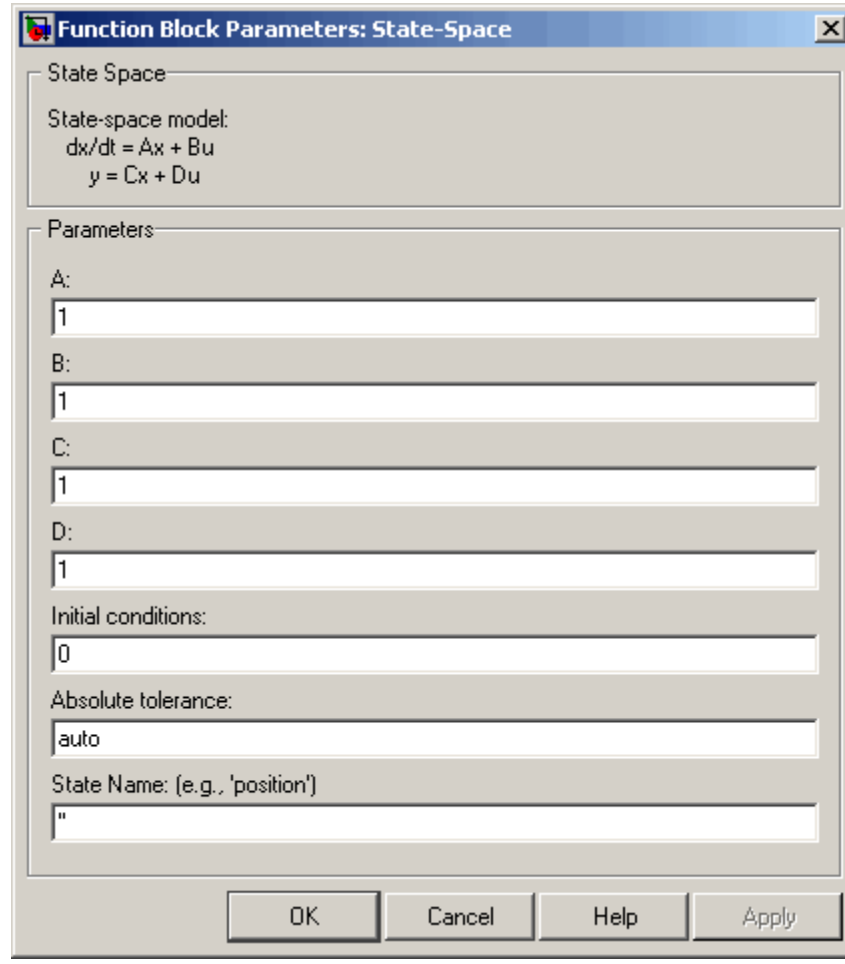
The block accepts one input and generates one output. The input vector width depends on the number of columns in the **B** and **D** matrices. The output vector width depends on the number of rows in the **C** and **D** matrices.

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

Data Type Support

A State-Space block accepts and outputs real signals of type `double`.

Parameters and Dialog Box



A

Specify the n-by-n matrix coefficient, where n is the number of states.

Settings

Default: 1

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

B

Specify the n-by-m matrix coefficient, where n is the number of states and m is the number of inputs.

Settings

Default: 1

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

C

Specify the r-by-n matrix coefficient, where r is the number of outputs and n is the number of states.

Settings

Default: 1

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

D

Specify the r-by-m matrix coefficient, where r is the number of outputs and m is the number of inputs.

Settings

Default: 1

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Initial conditions

Specify the initial state vector.

Settings

Default: `0`

The initial conditions of this block cannot be `inf` or `NaN`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Absolute tolerance

Specify the absolute tolerance for computing the block output.

Settings

Default: `auto`

- You can enter `auto` or a numeric value.
- If you enter `auto`, Simulink uses the absolute tolerance value in the Configuration Parameters dialog box (see "Solver Pane") to compute the block output.
- If you enter a numeric value, that value overrides the absolute tolerance in the Configuration Parameters dialog box.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

State Name (e.g., 'position')

Assign a unique name to each state.

Settings

Default: `' '`

If this field is blank, no name assignment occurs.

Tips

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'`.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, `{'a', 'b', 'c'}`. Each name must be unique.
- The state names apply only to the selected block.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

  For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.
- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell array, or structure.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Direct Feedthrough	Only if $\mathbf{D} \neq 0$
Sample Time	Continuous
Scalar Expansion	Yes, of the initial conditions

States	Depends on the size of **A**
Dimensionalized	Yes
Zero-Crossing Detection	No

See Also

Discrete State-Space

Step

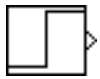
Purpose

Generate step function

Library

Sources

Description



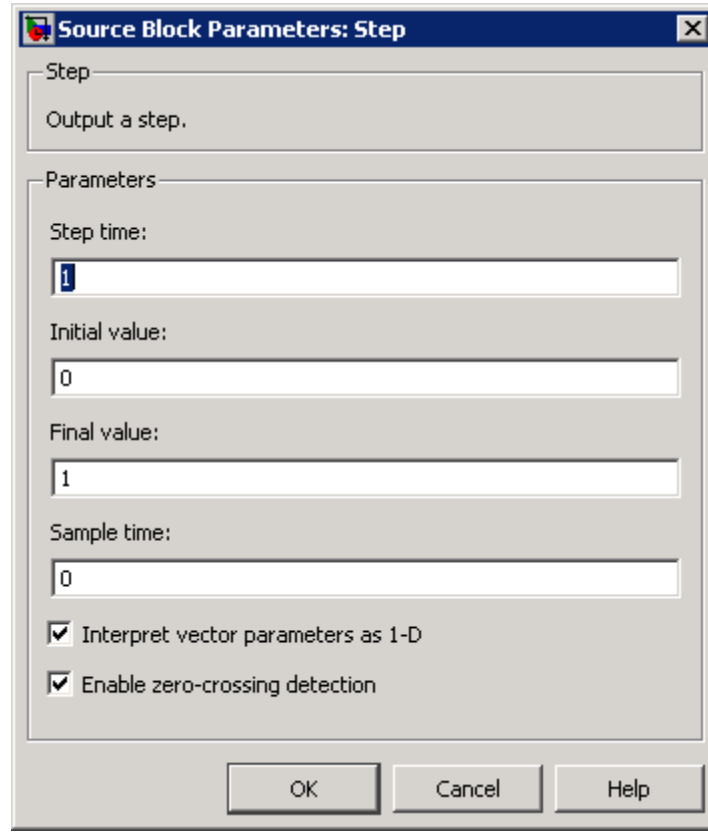
The Step block provides a step between two definable levels at a specified time. If the simulation time is less than the **Step time** parameter value, the block's output is the **Initial value** parameter value. For simulation time greater than or equal to the **Step time**, the output is the **Final 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



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

Step time

The time, in seconds, when the output jumps from the **Initial value** parameter to the **Final value** parameter. The default is 1 second.

Initial value
The block output until the simulation time reaches the **Step time** parameter. The default is `0`.

Final value
The block output when the simulation time reaches and exceeds the **Step time** parameter. The default is `1`.

Sample time
Sample rate of step. See "How to Specify the 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. See "Determining the Output Dimensions of Source Blocks" in the "Working with Signals" chapter of the Simulink documentation.

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

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled.

Purpose

Stop simulation when input is nonzero

Library

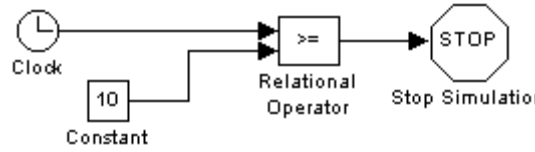
Sinks

Description

The Stop Simulation block stops the simulation when the input is nonzero.

The simulation completes the current time step before terminating. If the block input is a vector, any nonzero vector element causes the simulation to stop.

You can use this block with the Relational Operator block to control when the simulation stops. For example, this model stops the simulation when the input signal reaches 10.



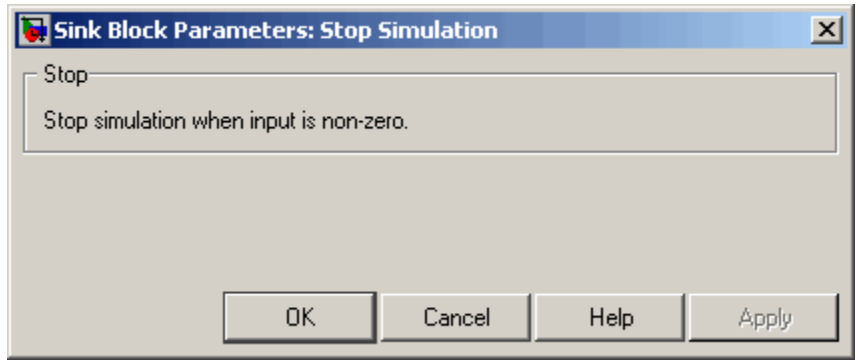
You cannot use the Stop Simulation block to pause the simulation. To create a block that pauses the simulation, see "Creating Pause Blocks" in the Simulink documentation for more information.

Data Type Support

The Stop Simulation block accepts real signals of type `double` or `Boolean`.

Stop Simulation

Parameters and Dialog Box



Characteristics

Sample Time	Inherited from driving block
Dimensionalized	Yes

Subsystem, Atomic Subsystem, Nonvirtual Subsystem, CodeReuse Subsystem

Purpose

Represent system within another system

Library

Ports & Subsystems

Description

A Subsystem block represents a subsystem of the system that contains it. The Subsystem block can represent a virtual subsystem or a nonvirtual subsystem. The primary difference is that nonvirtual subsystems provide the ability to control when the contents of the subsystem are evaluated. Nonvirtual subsystems are executed as a single unit (atomic execution) by the Simulink engine. A subsystem is virtual unless the block is conditionally executed and/or you have selected the block **Treat as atomic unit** check box.

Tip To determine if a subsystem is virtual, use the `get_param` function to obtain the value of the `IsSubsystemVirtual` property for the block. This property returns a read-only boolean value for the block. See "Block-Specific Parameters" on page 8-100 for the subsystem block that interests you.

An Atomic Subsystem block is a Subsystem block that has its **Treat as atomic unit** parameter selected by default. You can create conditionally executed nonvirtual subsystems that are executed only when a transition occurs on a triggering, function-call, action, or enabling input (see "Creating Conditional Subsystems").

You can create a subsystem in these ways:

- Copy the Subsystem (or Atomic Subsystem) block from the Ports & Subsystems library into your model. You can then add blocks to the subsystem by opening the Subsystem block and copying blocks into its window.
- Select the blocks and lines that are to make up the subsystem using a bounding box, then choose **Create Subsystem** from the **Edit** menu. Simulink software replaces the blocks with a Subsystem block. When you open the block, the window displays the blocks you

selected, adding Inport and Outport blocks to reflect signals entering and leaving the subsystem.

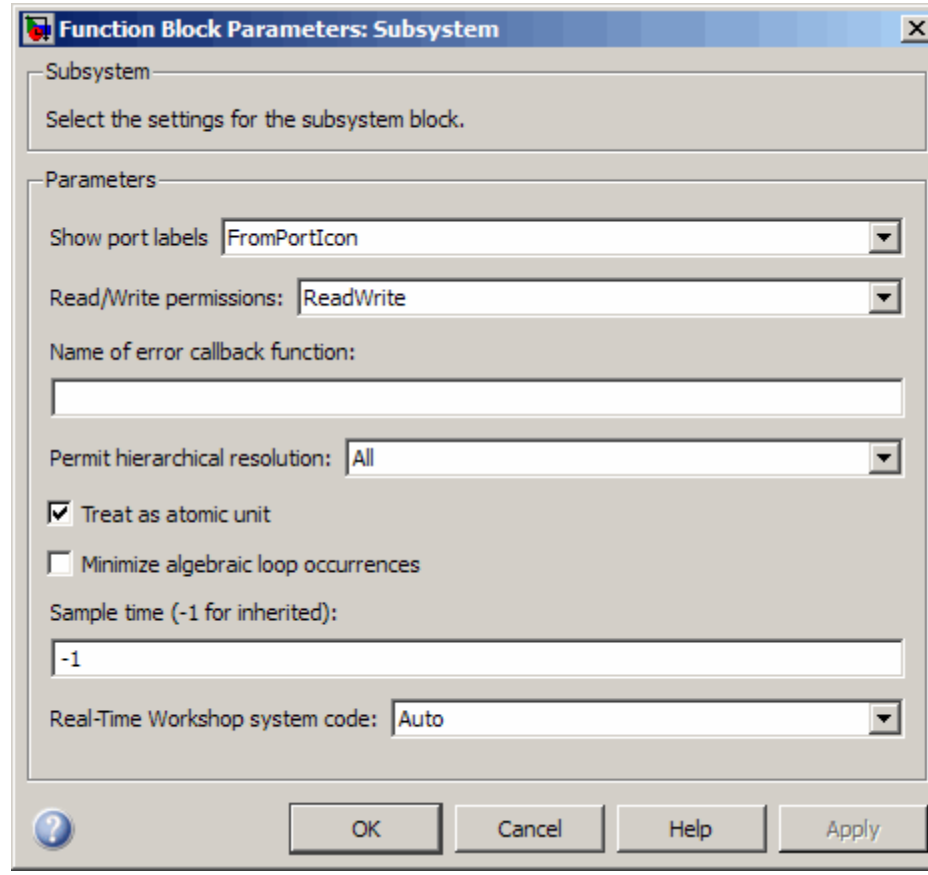
The number of input ports drawn on the Subsystem block's icon corresponds to the number of Inport blocks in the subsystem. Similarly, the number of output ports drawn on the block corresponds to the number of Outport blocks in the subsystem.

See "Creating Subsystems" in the Simulink User's Guide for more information about subsystems.

Data Type Support

See Inport for information on the data types accepted by a subsystem's input ports. See Outport for information on the data types output by a subsystem's output ports.

Parameters and Dialog Box



Show port labels

Cause Simulink software to display labels for the subsystem's ports on the subsystem's icon.

Settings

Default: `FromPortIcon`

`none`
: Does not display port labels on the subsystem block.

`FromPortIcon`
: If the corresponding port icon displays a signal name, display the signal name on the subsystem block. Otherwise, display the port block's name.

`FromPortBlockName`
: Display the name of the corresponding port block on the subsystem block.

`SignalName`
: If a name exists, display the name of the signal connected to the port on the subsystem block; otherwise, the name of the corresponding port block.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Read/Write permissions

Control user access to the contents of the subsystem.

Settings

Default: `ReadWrite`

`ReadWrite`
: Enables opening and modification of subsystem contents.

`ReadOnly`
: Enables opening but not modification of the subsystem. If the subsystem resides in a block library, you can create and open links to the subsystem and can make and modify local copies of the subsystem but cannot change the permissions or modify the contents of the original library instance.

`NoReadOrWrite`
: Disables opening or modification of subsystem. If the subsystem resides in a library, you can create links to the subsystem in a model but cannot open, modify, change permissions, or create local copies of the subsystem.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Name of error callback function

Enter name of a function to be called if an error occurs while Simulink software is executing the subsystem.

Settings

Default: `' '`

Simulink software passes two arguments to the function: the handle of the subsystem and a string that specifies the error type. If no function is specified, Simulink software displays a generic error message if executing the subsystem causes an error.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Permit hierarchical resolution

Specify whether to resolve names of workspace variables referenced by this subsystem.

Settings

Default: `All`

`All`
: Resolve all names of workspace variables used by this subsystem, including those used to specify block parameter values and Simulink data objects (for example, `Simulink.Signal` objects).

`ExplicitOnly`
: Resolve only names of workspace variables used to specify block parameter values, data store memory (where no block exists), signals, and states marked as "must resolve".

`None`
: Do not resolve any workspace variable names.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Treat as atomic unit

Causes Simulink software to treat the subsystem as a unit when determining the execution order of block methods.

Settings

Default: Off

 On

Cause Simulink software to treat the subsystem as a unit when determining the execution order of block methods. For example, when it needs to compute the output of the subsystem, Simulink software invokes the output methods of all the blocks in the subsystem before invoking the output methods of other blocks at the same level as the subsystem block.

 Off

Cause Simulink software to treat all blocks in the subsystem as being at the same level in the model hierarchy as the subsystem when determining block method execution order. This can cause execution of methods of blocks in the subsystem to be interleaved with execution of methods of blocks outside the subsystem.

Dependencies

This parameter enables:

- **Minimize algebraic loop occurrences**.
- **Sample time**
- **Real-Time Workshop system code** (requires a Real-Time Workshop license)

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Minimize algebraic loop occurrences

Try to eliminate any algebraic loops that include the subsystem.

Settings

Default: Off

 On

Try to eliminate any algebraic loops that include the subsystem.

 Off

Does not try to eliminate any algebraic loops that include the subsystem.

Dependency

Treat as atomic unit enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Propagate execution context across subsystem boundary

Enable execution context propagation across the boundary of this subsystem.

Settings

Default: Off

 On

Enables execution context propagation across this subsystem's boundary.

 Off

Does not enable execution context propagation across this subsystem's boundary.

Dependency

Conditional execution of the subsystem enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Warn if function-call inputs are context-specific

Simulink software displays a warning if it has to compute any of this function-call subsystem's inputs directly or indirectly during execution of a function-call.

Settings

Default: Off

 On

Simulink software displays a warning if it has to compute any of this function-call subsystem's inputs directly or indirectly during execution of a function-call.

 Off

Simulink software does not display a warning if it has to compute any of this function-call subsystem's inputs directly or indirectly during execution of a function-call.

Dependency

Use of a function-call subsystem enables this parameter.

The option is effective only if the **Context-dependent inputs** diagnostic on the **Configuration Parameters > Connectivity** dialog box is set to `Use local settings`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Specify whether all blocks in this subsystem must run at the same rate or can run at different rates.

Settings

Default: `-1`

- `-1`

  Specify the inherited sample time. Use this sample time if the blocks in the subsystem can run at different rates.

- `inf`

  Specify constant sample time

- `[Ts 0]`

  Specify periodic sample time.

Tips

- If the blocks in the subsystem can run at different rates, specify the subsystem's sample time as inherited (`-1`).
- If all blocks must run at the same rate, specify the sample time corresponding to this rate as the value of the subsystem's **Sample time** parameter.
- If any of the blocks in the subsystem specify a different sample time (other than `-1` or `inf`), Simulink software displays an error message when you update or simulate the model. For example, suppose all the blocks in the subsystem must run `5` times a second. To ensure this, specify the sample time of the subsystem as `0.2`. In this example, if any of the blocks in the subsystem specify a sample time other than `0.2`, `-1`, or `inf`, Simulink software displays an error when you update or simulate the model.

Dependencies

Treat as atomic unit enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Real-Time Workshop system code

Specify the code format to be generated for an atomic (nonvirtual) subsystem.

Settings

Default: `Auto`

`Auto`
: Real-Time Workshop software chooses the optimal format for you based on the type and number of instances of the subsystem that exist in the model.

`Inline`
: Real-Time Workshop software inlines the subsystem unconditionally.

`Function`
: Real-Time Workshop software generates a separate, non-reentrant function with no arguments, and optionally place the subsystem code in a separate file.

`Reusable function`
: Real-Time Workshop software generates a function with arguments that allows the subsystem's code to be shared by other instances of it in the model.

Tip

If you select `Reusable function` while your generated code is under source control, set **Real-Time Workshop file name options** to `Use subsystem name`, `Use function name`, or `User specified`. Otherwise, the names of your code files change whenever you modify your model, which prevents source control on your files.

Dependencies

- This parameter requires a Real-Time Workshop license.
- **Treat as atomic unit** enables this parameter.
- Setting this parameter to `Function` or `Reusable function` enables the following parameters:

- **Real-Time Workshop function name options**
- **Real-Time Workshop file name options**
- **Function with separate data** (requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file)
- **Memory section for initialize/terminate functions** (requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file)
- **Memory section for execution functions** (requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file)

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Real-Time Workshop function name options

Specify how Real-Time Workshop software is to name the function it generates for the subsystem.

Settings

Default: `Auto`

`Auto`
: Assign a unique function name using the default naming convention, *`model_subsystem`*`()`, where *`model`* is the name of the model and *`subsystem`* is the name of the subsystem (or that of an identical one when code is being reused).

`Use subsystem name`
: Use the subsystem name as the function name.

`User specified`
: Assign a unique, valid C or C++ function name that you specify.

Tip

If you specify `Use subsystem name` and the subsystem is a library block, Real-Time Workshop software names the function (and filename) with the name of the library block, regardless of the names used for that subsystem in the model.

Dependencies

- This parameter requires a Real-Time Workshop license.
- Selecting `User specified` enables the **Real-Time Workshop function name option** parameter.
- **Treat as atomic unit** enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Real-Time Workshop function name

Specify a unique, valid C or C++ function name for subsystem code.

Settings

Default: `' '`

Use this parameter if you want to give the function a specific name instead of allowing the Real-Time Workshop code generator to assign its own autogenerated name or use the subsystem name.

Dependencies

Setting **Real-Time Workshop function name options** to `User specified` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Real-Time Workshop file name options

Specify how Real-Time Workshop software is to name the separate file for the function it generates for the subsystem.

Settings

Default: `Auto`

`Auto`
: Generate the function code within the module generated from the subsystem's parent system, or, if the subsystem's parent is the model itself, within the file `model.c` or `model.cpp`

`Use subsystem name`
: Generate a separate file and name it with the name of the subsystem or library block

`Use function name`
: Generate a separate file and name it with the function name you specify for **Real-Time Workshop function name options**

`User specified`
: Assign a unique, valid C or C++ function name that you specify

Tip

If you specify `Use subsystem name`, the subsystem filename is mangled if the model contains Model blocks, or if a model reference target is being generated for the model. In these situations, the filename for the subsystem consists of the subsystem name prefixed by the model name.

Dependencies

- This parameter requires a Real-Time Workshop license.
- Setting this parameter to `User specified` enables the **Real-Time Workshop filename (no extension)** parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Real-Time Workshop file name (no extension)

Specify how Real-Time Workshop software is to name the file for the function it generates for the subsystem.

Settings

Default: `' '`

- The filename that you specify does not have to be unique. However, avoid giving non-unique names that result in cyclic dependencies (for example, `sys_a.h` includes `sys_b.h`, `sys_b.h` includes `sys_c.h`, and `sys_c.h` includes `sys_a.h`).

Dependencies

- This parameter requires a Real-Time Workshop license.
- Setting **Real-Time Workshop file name options** to `User specified` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Function with separate data

Generate subsystem function code in which the internal data for an atomic subsystem is separated from its parent model and is owned by the subsystem.

Settings

Default: Off

 On

Generate subsystem function code in which the internal data for an atomic subsystem is separated from its parent model and is owned by the subsystem. As a result, the generated code for the atomic subsystem is easier to trace and test. The data separation also tends to reduce the size of data structures throughout the model.

 Off

Do not generate subsystem function code in which the internal data for an atomic subsystem is separated from its parent model and is owned by the subsystem.

Dependencies

- This parameter requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file.
- Setting **Real-Time Workshop system code** to `Function` enables this parameter.
- Selecting this check box enables these parameters:
  - **Memory section for constants**
  - **Memory section for internal data**
  - **Memory section for parameters**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Memory section for initialize/terminate functions

Indicate how the Real-Time Workshop Embedded Coder software is to apply memory sections to the subsystem's initialization and termination functions.

Settings

Default: `Inherit from model`

`Inherit from model`
: Apply the root model's memory sections to the subsystem's function code

`Default`
: Not apply memory sections to the subsystem's system code, overriding any model-level specification

The memory section of interest
: Apply one of the model's memory sections to the subsystem

Tips

- The possible values vary depending on what (if any) package of memory sections you have set for the model's configuration. See "Inserting Comments and Pragmas in Generated Code", "Configuring Memory Sections", and "Real-Time Workshop Pane: Memory Sections" in the Real-Time Workshop Embedded Coder documentation.
- If you have not configured the model with a package, `Inherit from model` is the only value that appears. Otherwise, the list includes `Default` and all memory sections the model's package contains.
- These options can be useful for overriding the model's memory section settings for the given subsystem.

Dependencies

- This parameter requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file.
- Setting **Real-Time Workshop system code** to `Function` enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Memory section for execution functions

Indicate how the Real-Time Workshop Embedded Coder software is to apply memory sections to the subsystem's execution functions.

Settings

Default: `Inherit from model`

`Inherit from model`
: Apply the root model's memory sections to the subsystem's function code

`Default`
: Not apply memory sections to the subsystem's system code, overriding any model-level specification

The memory section of interest
: Apply one of the model's memory sections to the subsystem

Tips

- The possible values vary depending on what (if any) package of memory sections you have set for the model's configuration. See "Inserting Comments and Pragmas in Generated Code", "Configuring Memory Sections", and "Real-Time Workshop Pane: Memory Sections" in the Real-Time Workshop Embedded Coder documentation.
- If you have not configured the model with a package, `Inherit from model` is the only value that appears. Otherwise, the list includes `Default` and all memory sections the model's package contains.
- These options can be useful for overriding the model's memory section settings for the given subsystem.

Dependencies

- This parameter requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file.
- Setting **Real-Time Workshop system code** to `Function` enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Memory section for constants

Indicate how the Real-Time Workshop Embedded Coder software is to apply memory sections to the subsystem's data.

Settings

Default: `Inherit from model`

`Inherit from model`
: Apply the root model's memory sections to the subsystem's data

`Default`
: Not apply memory sections to the subsystem's data, overriding any model-level specification

The memory section of interest
: Apply one of the model's memory sections to the subsystem

Tips

- Can be useful for overriding the model's memory section settings for the given subsystem.
- The possible values vary depending on what (if any) package of memory sections you have set for the model's configuration. See "Configuring Memory Sections" in the Real-Time Workshop Embedded Coder User's Guide.
- If you have not configured the model with a package, `Inherit from model` is the only value that appears. Otherwise, the list includes `Default` and all memory sections the model's package contains.

Dependencies

- This parameter requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file.
- Setting **Real-Time Workshop system code** to `Function` and selecting the **Function with separate data** check box enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Memory section for internal data

Indicate how the Real-Time Workshop Embedded Coder software is to apply memory sections to the subsystem's data.

Settings

Default: `Inherit from model`

`Inherit from model`
: Apply the root model's memory sections to the subsystem's data

`Default`
: Not apply memory sections to the subsystem's data, overriding any model-level specification

The memory section of interest
: Apply one of the model's memory sections to the subsystem

Tips

- Can be useful for overriding the model's memory section settings for the given subsystem.
- The possible values vary depending on what (if any) package of memory sections you have set for the model's configuration. See "Configuring Memory Sections" in the Real-Time Workshop Embedded Coder User's Guide.
- If you have not configured the model with a package, `Inherit from model` is the only value that appears. Otherwise, the list includes `Default` and all memory sections the model's package contains.

Dependencies

- This parameter requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file.
- Setting **Real-Time Workshop system code** to `Function` and selecting the **Function with separate data** check box enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Memory section for parameters

Indicate how the Real-Time Workshop Embedded Coder software is to apply memory sections to the subsystem's data.

Settings

Default: `Inherit from model`

`Inherit from model`
: Apply the root model's memory sections to the subsystem's function code

`Default`
: Not apply memory sections to the subsystem's system code, overriding any model-level specification

The memory section of interest
: Apply one of the model's memory sections to the subsystem

Tips

- Can be useful for overriding the model's memory section settings for the given subsystem.
- The possible values vary depending on what (if any) package of memory sections you have set for the model's configuration. See "Configuring Memory Sections" in the Real-Time Workshop Embedded Coder User's Guide.
- If you have not configured the model with a package, `Inherit from model` is the only value that appears. Otherwise, the list includes `Default` and all memory sections the model's package contains.

Dependencies

- This parameter requires a Real-Time Workshop Embedded Coder license and an ERT-based system target file.
- Setting **Real-Time Workshop system code** to `Function` and selecting the **Function with separate data** check box enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Characteristics

Sample Time	Depends on the blocks in the subsystem
Dimensionalized	Depends on the blocks in the subsystem
Multidimensionalized	Depends on the blocks in the subsystem
Virtual	Yes, if the read-only property `IsSubsystemVirtual` is `on`
Zero Crossing	Yes, for enable and trigger ports if present

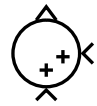
Purpose

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 signal.

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 + and - 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.
- For a round Sum block, the first input port is the port closest to the 12 o'clock position going in a counterclockwise direction around the block. Similarly, other input ports appear in counterclockwise order around the block.
- 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 input port is required, a single "+" or "-" collapses the element via the specified operation.

The Sum block first converts the input data type(s) to its accumulator data type, then performs the specified operations. The block converts the result to its output data type using the specified rounding and overflow modes.

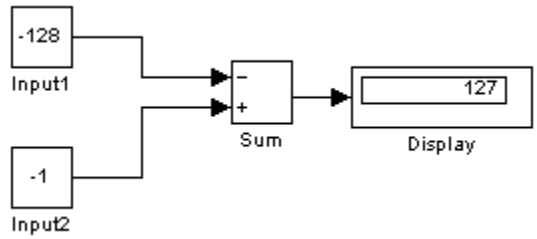
How the Sum Block Reorders Inputs

If you use `-` on the first input port, the Sum block reorders the inputs so that, if possible, the first input uses a `+` operation. For example, in the expression `output = -a-b+c`, the Sum block reorders the inputs so that `output = c-a-b`. To initialize the accumulator, the Sum block uses the first `+` input port.

The block avoids performing a unary minus operation on the first operand `a` because doing so can change the value of `a` for fixed-point data types. In that case, the output value differs from the result of accumulating the values for `a`, `b`, and `c`.

Tip To explicitly specify a unary minus operation for `output = -a-b+c`, you can use the Unary Minus block in the Math Operations library.

Suppose that you have the following model:



The following block parameters apply:

- Both Constant blocks, `Input1` and `Input 2`, use `int8` for the **Output data type**.
- The Sum block uses `int8` for both **Accumulator data type** and **Output data type**.
- The Sum block has **Saturate on integer overflow** turned `on`.

The Sum block reorders the inputs so that the following operations occur and you get the ideal result of 127.

Step	Block Operation
1	Reorders inputs from `( Input1 + Input2)` to `(Input2 Input1)`.
2	Initializes the accumulator by using the first + input port: `Accumulator = int8(-1) = -1`
3	Continues to accumulate values: `Accumulator = Accumulator  int8(-128) = 127`
4	Calculates the block output: `Output = int8(127) = 127`

If the Sum block does *not* reorder the inputs, the following operations occur instead and you get the nonideal result of 126.

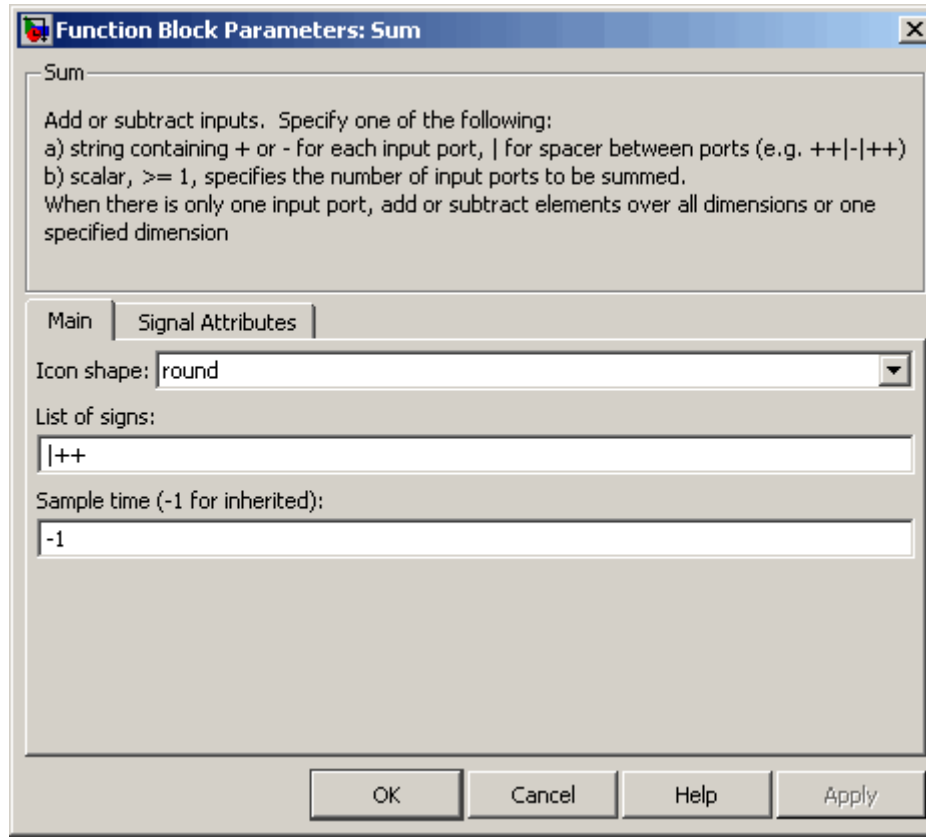
Step	Block Operation
1	Initializes the accumulator by using the first input port: `Accumulator = int8(-(-128)) = 127` Because saturation is on, the initial value of the accumulator saturates at 127 and does not wrap.
2	Continues to accumulate values: `Accumulator = Accumulator + int8(-1) = 126`
3	Calculates the block output: `Output = int8(126) = 126`

Data Type Support

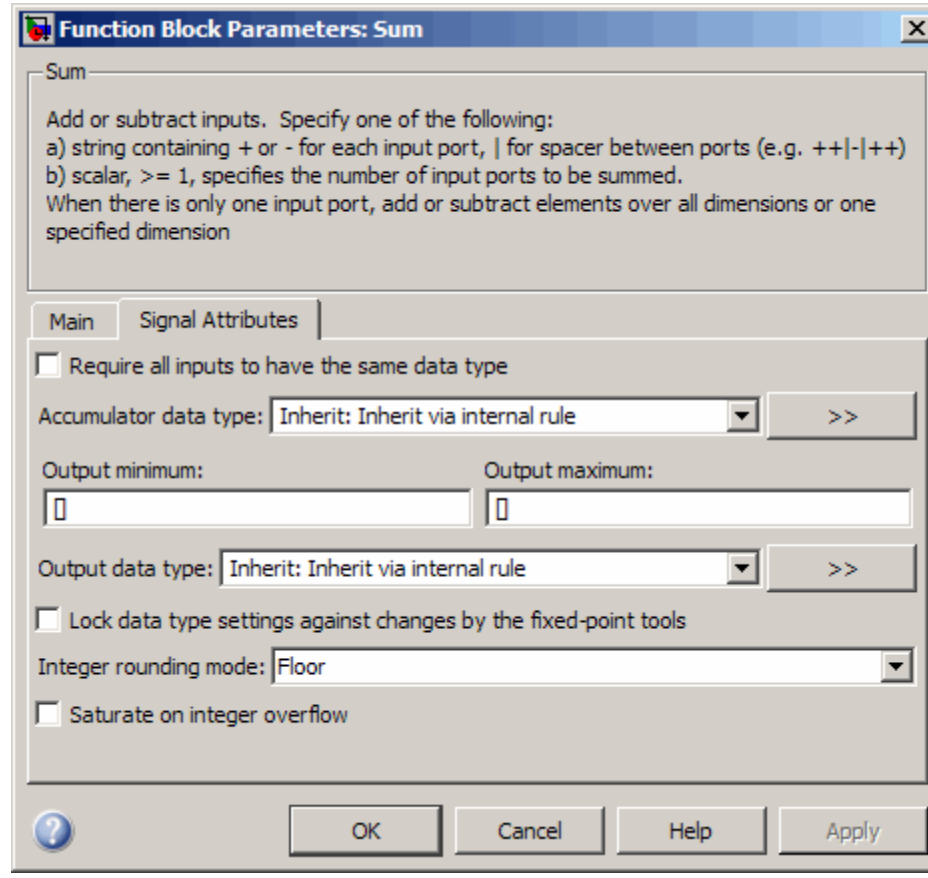
The Sum block accepts real or complex signals of any numeric data type supported by Simulink software, including fixed-point data types. The inputs may be of different data types unless you select the **Require all inputs to have the same data type** parameter.

Parameters and Dialog Box

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



The **Signal Attributes** pane of the Sum block dialog box appears as follows:


Function Block Parameters: Sum
Sum
Add or subtract inputs. Specify one of the following:
a) string containing + or - for each input port, | for spacer between ports (e.g. ++|-|++)
b) scalar, >= 1, specifies the number of input ports to be summed.
When there is only one input port, add or subtract elements over all dimensions or one specified dimension
Main
Signal Attributes
Require all inputs to have the same data type
Accumulator data type:
Inherit: Inherit via internal rule
>>
Output minimum:
Output maximum:
[]
[]
Output data type:
Inherit: Inherit via internal rule
>>
Lock data type settings against changes by the fixed-point tools
Integer rounding mode:
Floor
Saturate on integer overflow
OK
Cancel
Help
Apply

Show data type assistant

Display the **Data Type Assistant**.

Settings

The **Data Type Assistant** helps you set the **Output data type** parameter.

See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Icon shape

Designate the icon shape of the block.

Settings

Default: `round`

`rectangular`
: Designate the icon shape of the block as rectangular.

`round`
: Designate the icon shape of the block as round.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

List of signs

Enter plus (+) and minus (-) characters.

Settings

Default: |++

- 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.
- Enter as many plus (+) and minus (-) characters as there are inputs.

Tips

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.

Dependencies

Entering only one element enables the **Sum over** parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sum over

Select dimension over which to perform the sum over operation.

Settings

Default: `All dimensions`

`All dimensions`
: Sum all input elements, yielding a scalar.

`Specified dimension`
: Display the **Dimension** parameter, where you specify the dimension over which to perform the operation.

Dependencies

Selecting `Specified dimension` enables the **Dimension** parameter.

List of signs (when it has only one element) enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Dimension

Specify the dimension over which to perform the operation.

Settings

Default: `1`

The block follows the same summation rules as the MATLAB `sum` function.

Suppose that you have a 2-by-3 matrix U.

- Setting **Dimension** to `1` results in the output Y being computed as:

$$Y = \sum_{i=1}^{2} U(i, j)$$

- Setting **Dimension** to `2` results in the output Y being computed as:

$$Y = \sum_{j=1}^{3} U(i, j)$$

If the specified dimension is greater than the dimension of the input, an error message appears.

Dependencies

Setting **Sum over** to `Specified dimension` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits or specify another appropriate sample time such as continuous or inherited.

Settings

Default: -1

By default, the block inherits its sample time based upon the context of the block within the model. To set a different sample time, enter a valid sample time based upon the table in "Types of Sample Time".

See also "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Require all inputs to have the same data type

Require that all inputs have the same data type.

Settings

Default: Off

 On

Require that all inputs have the same data type.

 Off

Do not require that all inputs have the same data type.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Lock data type settings against changes by the fixed-point tools

Select to lock data type settings of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off


On

Locks all data type settings for this block.


Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change data type settings for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Floor`

`Ceiling`
: Round both positive and negative numbers toward positive infinity.

`Convergent`
: Round number to the nearest representable value. If a tie occurs, round to the nearest even stored value.

`Floor`
: Round both positive and negative numbers toward negative infinity.

`Nearest`
: Round number to the nearest representable value. If a tie occurs, round toward positive infinity.

`Round`
: Round number to the nearest representable value. If a tie occurs, round positive numbers toward positive infinity and round negative numbers toward negative infinity.

`Simplest`
: This option provides for an optimization of the rounding code for several blocks.

`Zero`
: Round number toward zero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Rounding” in the *Simulink Fixed Point User’s Guide*.

Saturate on integer overflow

Specify whether overflows saturate.

Settings

Default: Off

 On

Overflows saturate.

 Off

Overflows do not saturate.

Tips

- When you select this check box, saturation applies to every internal operation on the block, not just the output or result.
- In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Accumulator data type

Specify the accumulator data type.

Settings

Default: `Inherit:  Inherit via internal rule`

`Inherit:  Inherit via internal rule`
Use internal rule to determine accumulator data type.

`Inherit:  Same as first input`
Use data type of first input signal.

`double`
Accumulator data type is `double`.

`single`
Accumulator data type is `single`.

`int8`
Accumulator data type is `int8`.

`uint8`
Accumulator data type is `uint8`.

`int16`
Accumulator data type is `int16`.

`uint16`
Accumulator data type is `uint16`.

`int32`
Accumulator data type is `int32`.

`uint32`
Accumulator data type is `uint32`.

`fixdt(1,16,0)`
Accumulator data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
Accumulator data type is fixed point `fixdt(1,16,2^0,0)`.

`<data type expression>`
: The name of a data type object, for example `Simulink.NumericType`

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide* for more information.

Mode

Select the category of accumulator data to specify

Settings

Default: `Inherit`

`Inherit`
: Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. Below are the possible values, which can vary by block:

  - `Inherit via internal rule`
  - `Same as first input`

`Built in`
: Built-in data types. Selecting `Built in` enables a second menu/text box to the right. Below are the possible values, which can vary by block:

  - `double`
  - `single`
  - `int8`
  - `uint8`
  - `int16`
  - `uint16`
  - `int32`
  - `uint32`

`Fixed point`
: Fixed-point data types.

`Enumerated`
: Enumerated data types. This option is available on some blocks.

`Expression`

Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text to the right. Below are the possible values, which can vary by block:

- `<data type expression>`

Dependency

Clicking the **Show data type assistant** button for the accumulator data type enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data to be signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data to be signed.

`Unsigned`
: Specify the fixed-point data to be unsigned.

Dependencies

Selecting **Mode** > `Fixed point` for the accumulator data type enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that will hold the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` for the accumulator data type enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` or `Binary point`

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` for the accumulator data type enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` for the accumulator data type enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` for the accumulator data type enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` for the accumulator data type enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Output minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink software uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings

Default: `Inherit: Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block), `Inherit: Inherit from 'Constant value'` (Constant block), `Inherit: Inherit via back propagation` (Data Type Conversion block), `Inherit: Same as input` (Saturation block)

`Inherit: Inherit via internal rule`
: This option appears for some blocks. Simulink software chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the calculated 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 software chooses the output data type without regard to hardware constraints. Otherwise, Simulink software 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 software displays an error message in the Simulation Diagnostics Viewer.

`Inherit: Inherit from 'Constant value'`
: Use data type of **Constant value** parameter. This option appears for some blocks.

`Inherit: Inherit via back propagation`
: Use data type of the driving block.

`Inherit: Same as input`
: Use data type of sole input signal. This option appears for some blocks.

`Inherit: Same as first input`
: Use data type of first input signal. This option appears for some blocks.

`Inherit: Same as accumulator`
: Output data type is the same as accumulator data type. This option appears for some blocks.

`double`
: Output data type is `double`.

`single`
: Output data type is `single`.

`int8`
: Output data type is `int8`.

`uint8`
: Output data type is `uint8`.

`int16`
: Output data type is `int16`.

`uint16`
: Output data type is `uint16`.

`int32`
: Output data type is `int32`.

`uint32`
: Output data type is `unt32`.

`fixdt(1,16,0)`
: Output data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Output data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum: <class name>`
: Use an enumerated data type, for example, `Enum: BasicColors`. This option appears for some blocks.

`<data type expression>`
: Use a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Mode

Select the category of data to specify.

Settings

Default: `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Built in` (Logical Operator, Relational Operator)

`Inherit`

Inheritance rules for data types. Selecting `Inherit` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `Inherit from 'Constant value'` (Constant block default)
- `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
- `Inherit via back propogation` (Data Type Conversion block default)
- `auto` (Inport, Outport block default)
- `Logical (see Configuration Parameters: Optimization)`
- `Same as first input`
- `Same as input` (Saturation block default)
- `Same as accumulator`

`Built in`

Built-in data types. Selecting `Built in` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
- `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
Fixed-point data types.

`Enumerated`
Enumerated data types. This option is available on some blocks. Selecting `Enumerated` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `<class name>`

`Expression`
Expressions that evaluate to data types. Selecting `Expression` enables a second menu/text box to the right. The following list is the possible values, which can vary by block:

- `<data type expression>`

Dependency

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specify the fixed-point data as signed.

`Unsigned`
: Specify the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default: `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch), `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specify binary point location.

`Slope and bias`
: Enter slope and bias.

`Best precision`
: Specify best-precision values. This option appears for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**
- **Calculate Best-Precision Scaling**

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

See “Specifying a Fixed-Point Data Type” in the *Simulink User's Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes
States	0
Dimensionalized	Yes
Multidimensionalized	Yes, only along the specified dimension
Zero Crossing	No

Switch

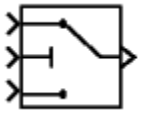
Purpose

Switch output between first input and third input based on value of second input

Library

Signal Routing

Description



The Switch block passes through the first input or the third input based on the value of the second input. The first and third inputs are called *data inputs*. The second input is called the *control input*. See "How to Rotate a Block" in the *Simulink User's Guide* for a description of the port order for various block orientations.

You select the condition under which the block passes the first input with the **Criteria for passing first input** parameter.

Note If the data inputs to the switch are buses, the element names of both buses must be the same. Using the same element names ensures that the output bus has the same element names no matter which input bus the block selects. You can ensure that your model meets this requirement by using a bus object to define the buses with the model **Element name mismatch** diagnostic set to error. See "Connectivity Diagnostics Overview" for more information.

To immediately back propagate a known output data type to the first and third input ports, set the **Output data type** parameter to `Inherit: Inherit via internal rule` and select the **Require all data port inputs to have the same data type** check box.

Data Type Support

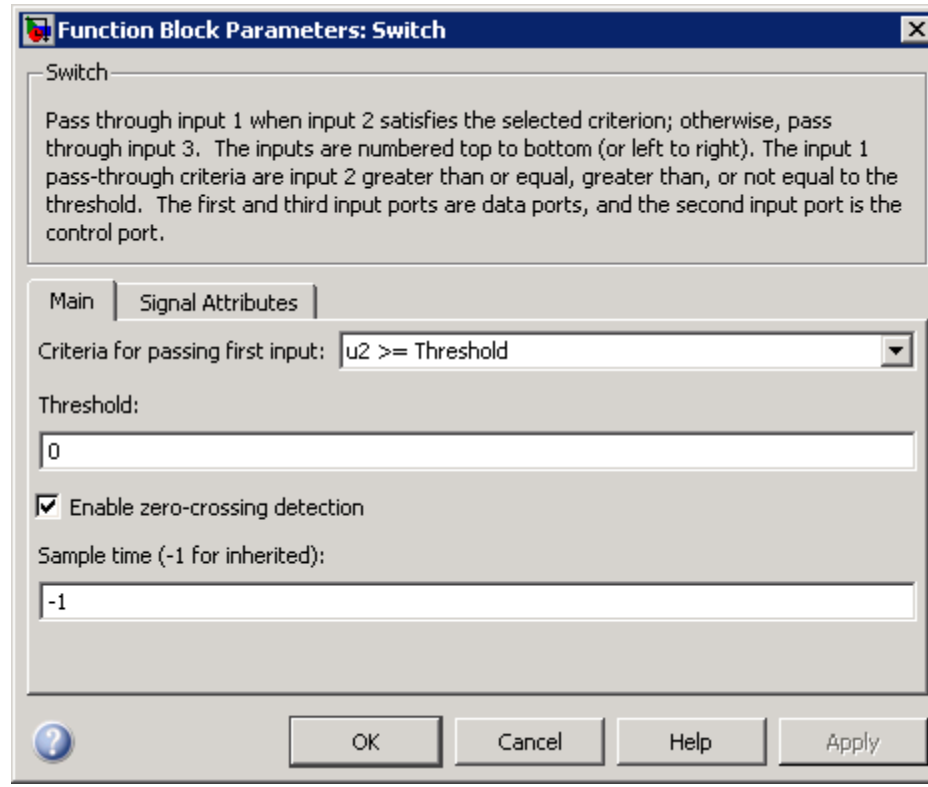
The control signal can be of any data type that Simulink supports, including fixed-point and enumerated types. The control signal cannot be complex. If the control signal is enumerated, the block **Threshold** parameter must be a value of the same enumerated type.

The data signals can be of any data type that Simulink supports. If either data signal is of an enumerated type, the other must be of the same enumerated type.

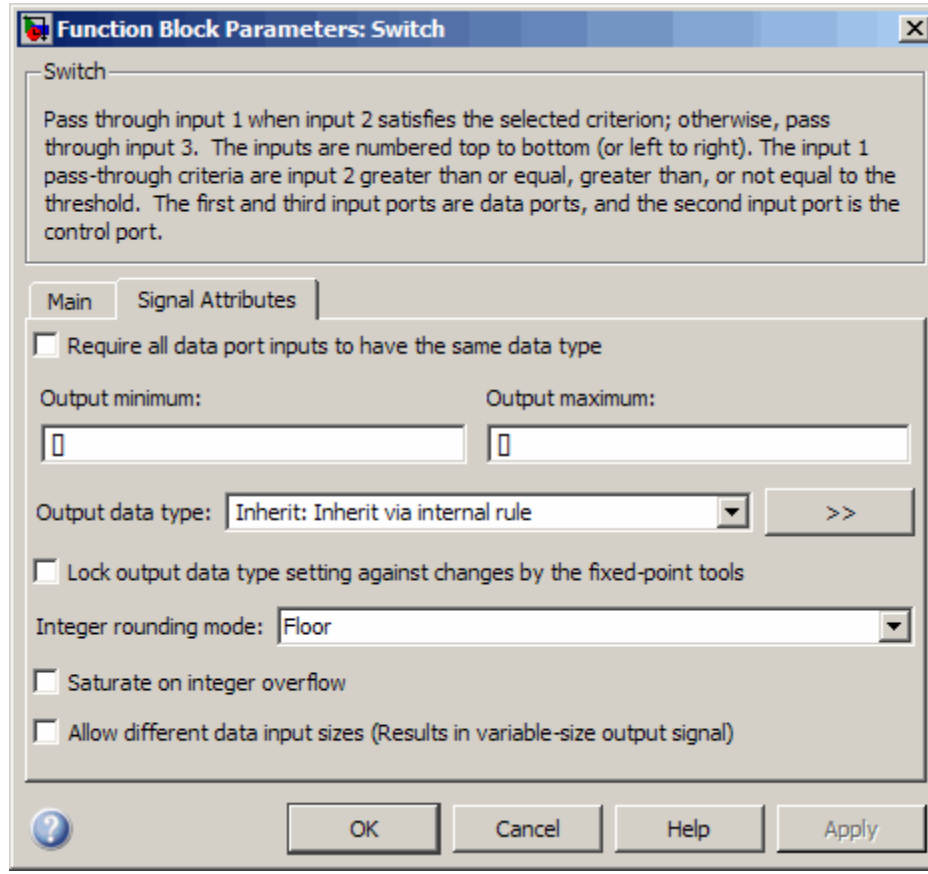
For more information, see “Data Types Supported by Simulink” in the *Simulink User's Guide*.

Parameters and Dialog Box

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



The **Signal Attributes** pane of the Switch block dialog box appears as follows:



- "Criteria for passing first input" on page 2-1298
- "Threshold" on page 2-1300
- "Enable zero-crossing detection" on page 2-1301
- "Sample time (-1 for inherited)" on page 2-1406
- "Require all data port inputs to have the same data type" on page 2-1303
- "Lock output data type setting against changes by the fixed-point tools" on page 2-1304
- "Integer rounding mode" on page 2-1305
- "Saturate on integer overflow" on page 2-1306
- "Allow different data input sizes" on page 2-1307

Criteria for passing first input

Select the condition under which the block passes the first input. If the control input meets the condition set in the **Criteria for passing first input** parameter, the block passes the first input. Otherwise, the block passes the third input.

Settings

Default: `u2 >= Threshold`

`u2 >= Threshold`
: Checks whether the control input is greater than or equal to the threshold value.

`u2 > Threshold`
: Checks whether the control input is greater than the threshold value.

`u2 ~=0`
: Checks whether the control input is nonzero.

> **Note** The Switch block does not support `u2 ~=0` mode for enumerated data types.

Tip

When the control input is a Boolean signal, use one of these combinations of condition and threshold value:

- `u2 >= Threshold`, where the threshold value equals 1
- `u2 > Threshold`, where the threshold value equals 0
- `u2 ~=0`

Otherwise, the Switch block ignores threshold values and uses the Boolean value for signal routing. For a value of 1, the block passes the first input, and for a value of 0, the block passes the third input.

A warning message that describes this behavior also appears in the MATLAB Command Window.

Dependencies

Selecting `u2 ~=0` disables the **Threshold** parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Threshold

Assign the switch threshold that determines which input the block passes to the output.

Settings

Default: `0`

Minimum: value from the **Output minimum** parameter

Maximum: value from the **Output maximum** parameter

Dependencies

Setting **Criteria for passing first input** to `u2 ~=0` disables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Enable zero-crossing detection

Select to enable zero-crossing detection.

Settings

Default: On

 On

Enables zero-crossing detection.

 Off

Disables zero-crossing detection.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits or specify another appropriate sample time such as continuous or inherited.

Settings

Default: -1

By default, the block inherits its sample time based upon the context of the block within the model. To set a different sample time, enter a valid sample time based upon the table in "Types of Sample Time".

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Require all data port inputs to have the same data type

Require all data inputs to have the same data type.

Settings

Default: Off

 On

Requires all data inputs to have the same data type.

 Off

Does not require all data inputs to have the same data type.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Lock output data type setting against changes by the fixed-point tools

Select to lock the output data type setting of this block against changes by the Fixed-Point Tool and the Fixed-Point Advisor.

Settings

Default: Off

 On

Locks the output data type setting for this block.

 Off

Allows the Fixed-Point Tool and the Fixed-Point Advisor to change the output data type setting for this block.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Fixed-Point Tool” and “Fixed-Point Advisor” in the Simulink Fixed Point documentation.

Integer rounding mode

Select the rounding mode for fixed-point operations.

Settings

Default: `Floor`

`Ceiling`
: Rounds both positive and negative numbers toward positive infinity.

`Convergent`
: Rounds number to the nearest representable value. If a tie occurs, round to the nearest even stored value.

`Floor`
: Rounds both positive and negative numbers toward negative infinity.

`Nearest`
: Rounds number to the nearest representable value. If a tie occurs, round toward positive infinity.

`Round`
: Rounds number to the nearest representable value. If a tie occurs, round positive numbers toward positive infinity and round negative numbers toward negative infinity.

`Simplest`
: Provides for an optimization of the rounding code for several blocks.

`Zero`
: Rounds number toward zero.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

See Also

For more information, see “Rounding” in the *Simulink Fixed Point User’s Guide*.

Saturate on integer overflow

Specify whether overflows saturate.

Settings

Default: Off

 On

Specifies overflows saturate.

 Off

Specified overflows do not saturate.

Tips

- When you select this check box, saturation applies to every internal operation on the block, not just the output or result.
- In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Allow different data input sizes

Select this check box to allow input signals with different sizes.

Settings

Default: Off

 On

Allows input signals with different sizes, and propagates the input signal size to the output signal.

 Off

Inputs signals must be the same size.

Command-Line Information

Parameter: `AllowDiffInputSignals`
Type: string
Value: `'on' | 'off'`
Default: `'off'`

Output minimum

Specify the minimum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `-Inf`.

Simulink uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output maximum

Specify the maximum value that the block should output.

Settings

Default: `[]`

The default value, `[]`, is equivalent to `Inf`.

Simulink uses this value to perform:

- Parameter range checking (see "Checking Parameter Values") for some blocks
- Simulation range checking (see "Checking Signal Ranges")
- Automatic scaling of fixed-point data types

Tip

This number must be a double scalar value.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Output data type

Specify the output data type.

Settings
Default:

- `Inherit: Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block)
- `Inherit: Inherit from 'Constant value'` (Constant block)
- `Inherit: Inherit via back propagation` (Data Type Conversion block)
- `Inherit: Same as input` (Saturation block)

`Inherit: Inherit via internal rule`

This option appears only for some blocks. Simulink chooses a combination of output scaling and data type that requires the smallest amount of memory. The choices are consistent with accommodating the calculated output range, maintaining the output precision of the block, and the word size of the targeted hardware implementation specified for the model.

If you set the **Device type** parameter on the **Hardware Implementation** configuration parameters pane 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 you specify `ASIC/FPGA` as the targeted hardware type, the output data type is `sfix24`.

If you select `Unspecified (assume 32-bit Generic)` for a generic 32-bit microprocessor 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.

`Inherit: Inherit from 'Constant value'`
Usees data type of **Constant value** parameter. This option appears only for some blocks.

`Inherit: Inherit via back propagation`
Uses data type of the driving block.

`Inherit: Same as input`
Uses data type of sole input signal. This option appears only for some blocks.

`Inherit: Same as first input`
Uses data type of first input signal. This option appears only for some blocks.

`Inherit: Same as accumulator`
Specifies output data type is the same as accumulator data type. This option appears for some blocks.

`double`
Specifies output data type is `double`.

`single`
Specified output data type is `single`.

`int8`
Specifies output data type is `int8`.

`uint8`
Specifies output data type is `uint8`.

`int16`
Specifies output data type is `int16`.

`uint16`
Specifies output data type is `uint16`.

`int32`
Specifies output data type is `int32`.

`uint32`
: Specifies output data type is `unt32`.

`fixdt(1,16,0)`
: Specifies output data type is fixed point `fixdt(1,16,0)`.

`fixdt(1,16,2^0,0)`
: Specifies output data type is fixed point `fixdt(1,16,2^0,0)`.

`Enum:  <class name>`
: Uses an enumerated data type, for example, `Enum:  BasicColors`. This option appears only for some blocks.

`<data type expression>`
: Uses a data type object, for example, `Simulink.NumericType`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying Block Output Data Types" in the *Simulink User's Guide* for more information.

Mode

Select the category of data to specify.

Settings
Default:

- `Inherit` (Constant, Data Type Conversion, Discrete Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch)
- `Built in` (Logical Operator, Relational Operator)

`Inherit`
: Specifies inheritance rules for data types. Selecting `Inherit` enables a list of possible values, which can vary by block:

  - `Inherit from 'Constant value'` (Constant block default)
  - `Inherit via internal rule` (Discrete-Time Integrator, Gain, Product, Sum, Switch block default)
  - `Inherit via back propogation` (Data Type Conversion block default)
  - `auto` (Inport, Outport block default)
  - `Logical (see Configuration Parameters: Optimization)`
  - `Same as first input`
  - `Same as input` (Saturation block default)
  - `Same as accumulator`

`Built in`
: Specifies built-in data types. Selecting `Built in` enables a list of possible values, which can vary by block:

  - `double` (Constant, Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch block default)
  - `single`

- `int8`
- `uint8`
- `int16`
- `uint16`
- `int32`
- `uint32`
- `boolean` (Logical Operator, Relational Operator block default)

`Fixed point`
: Specifies fixed-point data types.

`Enumerated`
: Specifies enumerated data types. This option is available only on some blocks. Selecting `Enumerated` enables a list of possible values, which can vary by block:

  - `<class name>`

`Expression`
: Specifies expressions that evaluate to data types. Selecting `Expression` enables a list of possible values, which can vary by block:

  - `<data type expression>`

Dependencies

Clicking the **Show data type assistant** button enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Using the Data Type Assistant" in the *Simulink User's Guide*.

Signedness

Specify whether you want the fixed-point data as signed or unsigned.

Settings

Default: `Signed`

`Signed`
: Specifies the fixed-point data as signed.

`Unsigned`
: Specifies the fixed-point data as unsigned.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Word length

Specify the bit size of the word that holds the quantized integer.

Settings

Default: `16`

Minimum: `0`

Maximum: `32`

Large word sizes represent large values with greater precision than small word sizes.

Dependencies

Selecting **Mode** > `Fixed point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Scaling

Specify the method for scaling your fixed-point data to avoid overflow conditions and minimize quantization errors.

Settings

Default:

- `Best precision` (Constant), `Binary point` (Data Type Conversion, Discrete-Time Integrator, Gain, Inport, Outport, Product, Saturation, Sum, Switch)
- `Integer` (Logical Operator, Relational Operator)

`Binary point`
: Specifies binary point location.

`Slope and bias`
: Enters slope and bias.

`Best precision`
: Specifies best-precision values. This option appears only for some blocks.

`Integer`
: Specify integer. This setting has the same result as specifying a binary point location and setting fraction length to 0. This option appears for some blocks.

Dependencies

Selecting `Binary point` enables:

- **Fraction length**
- **Calculate Best-Precision Scaling**

Selecting `Slope and bias` enables:

- **Slope**
- **Bias**

- **Calculate Best-Precision Scaling**

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Fraction length

Specify fraction length for fixed-point data type.

Settings

Default: `0`

Binary points can be positive or negative integers.

Dependencies

Selecting **Scaling** > `Binary point` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Slope

Specify slope for the fixed-point data type.

Settings

Default: `2^0`

Specify any positive real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bias

Specify bias for the fixed-point data type.

Settings

Default: `0`

Specify any real number.

Dependencies

Selecting **Scaling** > `Slope and bias` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

See "Specifying a Fixed-Point Data Type" in the *Simulink User's Guide* for more information.

Bus Support

The Switch block is a bus-capable block. The data inputs can be virtual or nonvirtual bus signals subject to the following restrictions:

- All the buses must be equivalent (same hierarchy with identical names and attributes for all elements).
- All signals in a nonvirtual bus input to a Switch block must have the same sample time. The requirement holds even if the elements of the associated bus object specify inherited sample times.

You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus. See "Using Composite Signals" and Bus-Capable Blocks for more information.

Characteristics

Bus-capable	Yes, with restrictions
Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter

Scalar Expansion	Yes
Dimensionalized	Yes
Multidimensionalized	Yes
Zero-Crossing Detection	Yes, if enabled

See Also

Multiport Switch

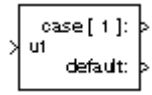
Purpose

Implement C-like `switch` control flow statement

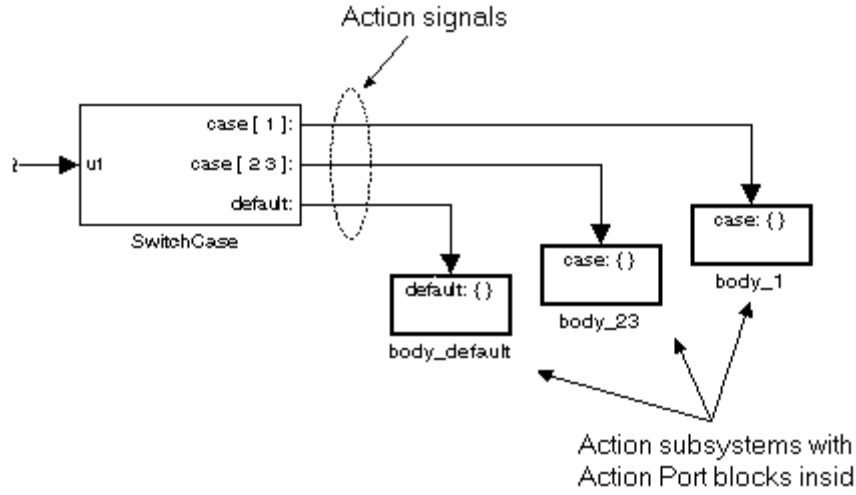
Library

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 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 Switch Case block dialog box and update parameters:
   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.

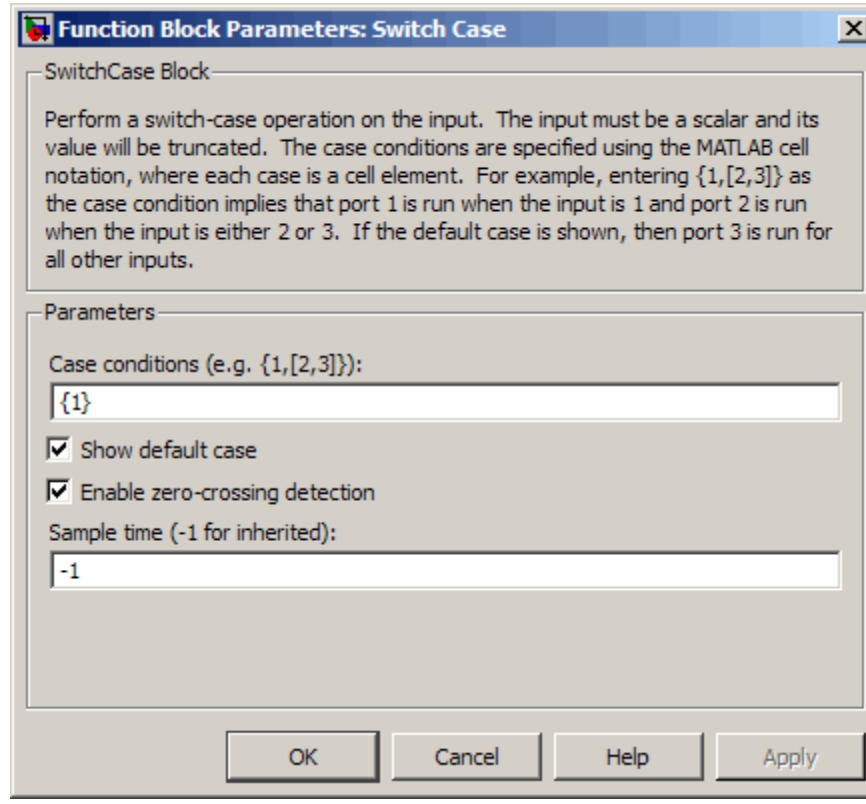
5 In each Switch Case Action subsystem, enter the Simulink logic appropriate to the case it handles. All blocks in a Switch Case Action Subsystem must run at the same rate as the driving Switch Case block. You can achieve this by setting each block's sample time parameter to be either inherited (`-1`) or the same value as the Switch Case block's sample time.

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 numeric data type supported by Simulink software, including a fixed-point data type. The input to `u1` cannot be `Boolean` or have an enumerated type. Noninteger inputs are truncated. Data outputs are action signals to Switch Case Action subsystems that you create 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 zero-crossing detection. For more information, see "Zero-Crossing Detection" in the Simulink documentation.

Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the 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 Detection	Yes, if enabled

Switch Case Action Subsystem

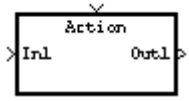
Purpose

Represent subsystem whose execution is triggered by 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.

Note All blocks in a Switch Case Action Subsystem must run at the same rate as the driving Switch Case block. You can achieve this by setting each block's sample time parameter to be either inherited (`-1`) or the same value as the Switch Case block's sample time.

For more information, see the Switch Case block and "Modeling Control Flow Logic" in the "Creating a Model" chapter of the Simulink documentation.

Purpose

Delay scalar signal multiple sample periods and output all delayed versions

Library

Discrete

Description

The Tapped Delay block delays an input by the specified number of sample periods and outputs all the delayed versions. Use this block to discretize a signal in time or resample a signal at a different rate.

The block accepts one scalar input and generates an output vector that contains each delay. Specify the order of the delays in the output vector with the **Order output vector starting with** parameter:

- **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.

Specify the output vector for the first sampling period with the **Initial condition** parameter. Careful selection of this parameter can minimize unwanted output behavior.

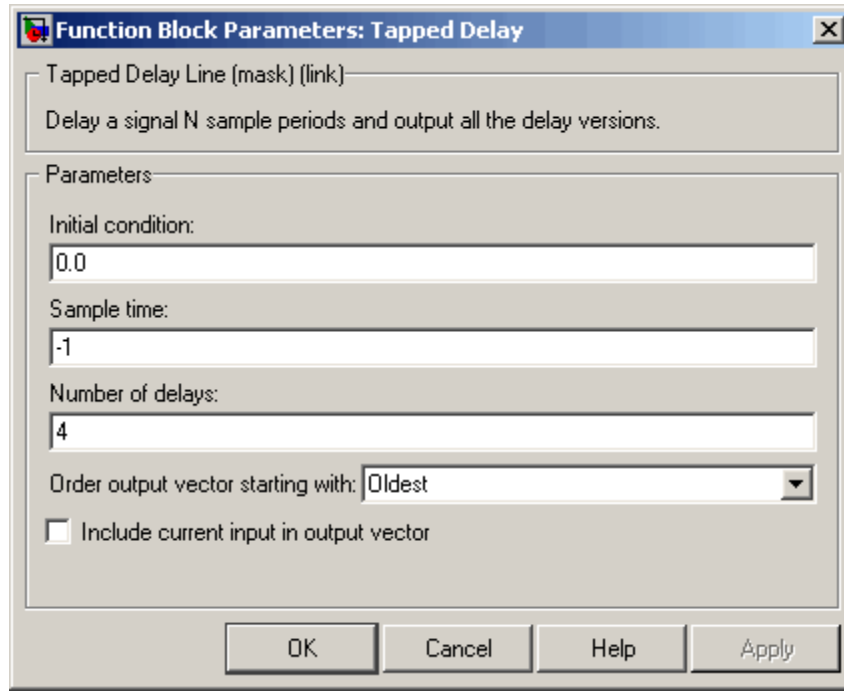
Specify the time between samples with the **Sample time** parameter. Specify the number of delays with the **Number of delays** parameter. A value of `-1` instructs the block to inherit the number of delays by back propagation. Each delay is equivalent to the z^{-1} discrete-time operator, which the Unit Delay block represents.

Data Type Support

The Tapped Delay block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

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 software 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 "How to Specify the Sample Time" in the online Simulink documentation for more information.

Number of delays

Specify the number of discrete-time operators.

Order output vector starting with
Specify whether to output the oldest delay version first, or the newest delay version 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** check box is selected. No, otherwise.
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of initial conditions

Terminator

Purpose

Terminate unconnected output port

Library

Sinks

Description

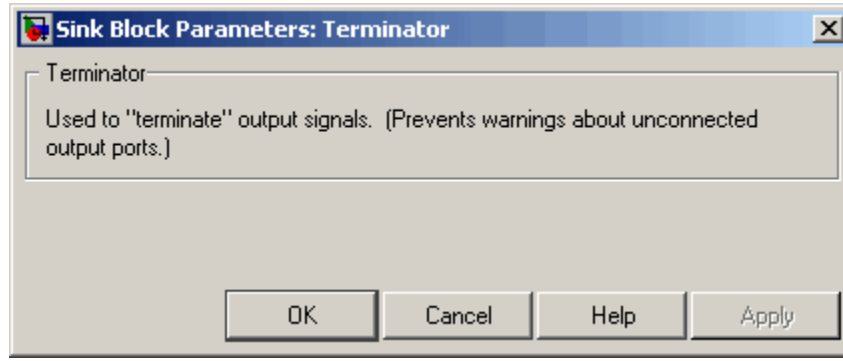
The Terminator block can be used to cap blocks whose output ports are not connected to other blocks. If you run a simulation with blocks having unconnected output ports, Simulink software issues warning messages. Using Terminator blocks to cap those blocks avoids warning messages.

Data Type Support

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

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

Parameters and Dialog Box



Characteristics

Sample Time	Inherited from driving block
Dimensionalized	Yes

Multidimensionalized	Yes
Virtual	Yes For more information, see “Virtual Blocks” in the Simulink documentation.

Time-Based Linearization

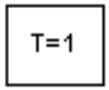
Purpose

Generate linear models in 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 block sets the following model parameters to the indicated values:

- `BufferReuse = 'off'`
- `RTWInlineParameters = 'on'`
- `BlockReductionOpt = 'off'`

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

Field	Description
`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);
```

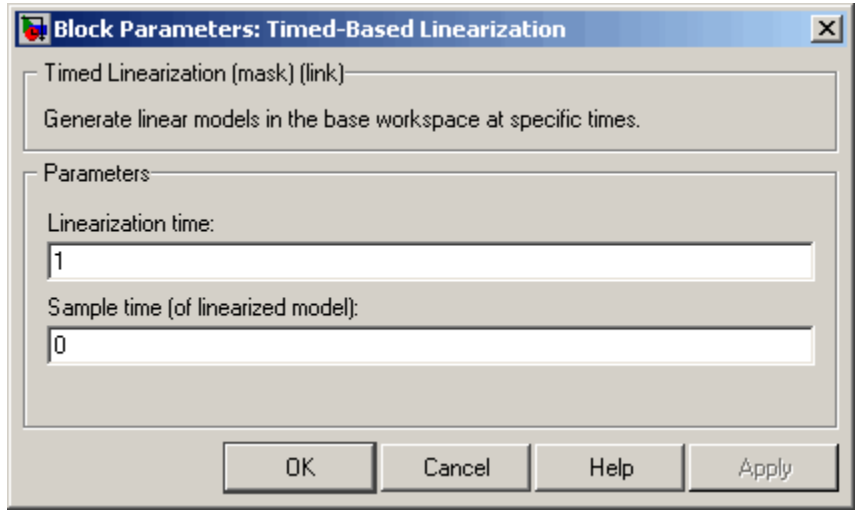
9 Get the state vectors at the operating points.

```
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 4-130).

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	No

See Also

Trigger-Based Linearization

To File

Purpose

Write data to file

Library

Sinks

Description

The To File block inputs a scalar or vector signal of type `double` and writes the signal data to a MAT-file. The block's icon shows the name of the output file. If the output file exists at the time the simulation starts, the block overwrites the file. The block writes to the output file incrementally, so memory overhead during simulation is low. The file automatically closes when simulation is complete.

Note If simulation terminates abnormally, the To File block may have written a MAT-file that contains some data, but the file is unusable.

The To File block writes data as a matrix of two or more rows. The block writes one column to the MAT-file for each time step. The first element of the column contains the simulation time. The remainder of the column contains scalar or vector data for the time shown at the top of the column. The stored matrix has this form:

$$\begin{bmatrix} t_1 & t_2 & \dots t_{final} \\ u1_1 & u1_2 & \dots u1_{final} \\ \dots & & \\ un_1 & un_2 & \dots un_{final} \end{bmatrix}$$

To avoid the overhead of compressing data in real time, the To File block writes an uncompressed MAT-file. To compress the file, load and save it in MATLAB. The resaved file will be smaller because MATLAB software automatically compresses MAT-files.

The MAT-file can contain at most 10^8 elements, where each timestamp and each data value counts as one element. Thus if N is the number of time steps in the file, and W is the width of the data, N(W+1) cannot exceed 10^8. If writing the next column to the file would cause the number of elements to exceed 10^8, the To File block discards the

unwritten column, closes the file, and posts a warning. Simulation continues without interruption, but the To File block discards all subsequent data. The truncated MAT-file is syntactically valid and can be used as input to a subsequent simulation, or loaded into MATLAB for any purpose.

Controlling When Data is Written to the File

For a fixed-step solver, the block's **Decimation** and **Sample Time** parameters control when data is written to the file. For a variable-step solver, the **Configuration Parameters > Data Import/Export Pane > "Output options"** parameter controls the amount of data available to the To File block, and the block's **Decimation** and **Sample Time** parameters control how much of this data the block actually writes.

For example, if you need to ensure that data is written at identical time points over multiple simulations with a varying-step solver, set **Output Options** to `Produce specified output only` and enter the desired time vector in the **Output times** field. See "Importing Data from a Workspace" for guidelines on choosing time vectors for discrete systems.

Saving Data for Use by a From File Block

The From File block can use data written by a To File block without any modifications to the data or other special provisions.

Saving Data for Use by a From Workspace Block

The From Workspace block requires data that is the transposition of the data written by the To File block. To provide the required format, use MATLAB commands to load and transpose the data from MAT-file. You will then be able to use a From Workspace block to access the data.

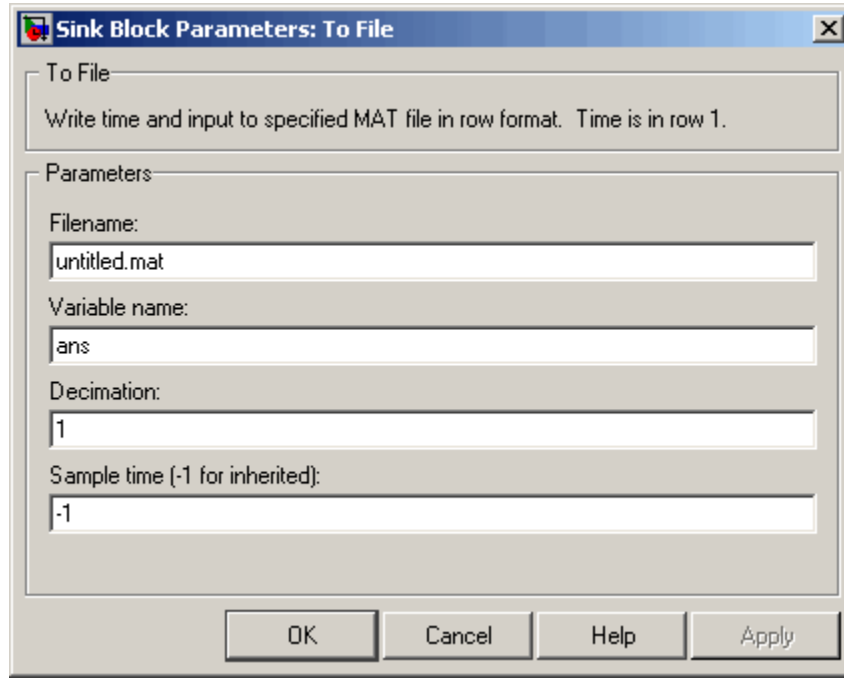
Limitation on To File block in a Referenced Model

When a To File block exists in a referenced model, that model must be a single-instance model. Only one instance of such a model can exist in a model hierarchy. Otherwise, the corresponding To File blocks in different instances of the referenced model would all try to write to the same data file, which could not provide a useful result. See "General Reusability Limitations" for more information.

Data Type Support

The To File block accepts only vector and scalar data, and all data must be of type `double`. The To File block does not accept matrix signals or complex data. To save other kinds of data, use a To Workspace block to save the data to the MATLAB workspace. You can then save the data to a file.

Parameters and Dialog Box



Filename

The pathname or filename of the MAT-file in which to store the output. On UNIX systems, the pathname can start with a tilde (~) character signifying your home directory. The default filename is `untitled.mat`. If you specify a filename without path information, Simulink software stores the file in the MATLAB working directory. (To determine the working directory, type

`pwd` at the MATLAB command line.) If the file already exists, Simulink software overwrites it.

Variable name
The name of the matrix contained in the named file. The default name is `ans`.

Decimation
Specifies writing data at every `n`th sample, where `n` is the decimation factor. The default decimation is `1`, which writes data at every time step.

Sample time
Specifies the sample period and offset 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 constant. The default is `-1`, which inherits the sample time from the driving block when determining the points to write. See "How to Specify the Sample Time" for more information.

Characteristics

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

See Also

"Importing and Exporting Data", From File, From Workspace, To Workspace

To Workspace

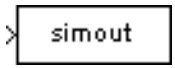
Purpose

Write data to MATLAB workspace

Library

Sinks

Description



The To Workspace block inputs a signal and writes the signal data to the MATLAB workspace. The block writes the data 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.

For variable-step solvers, the **Output options** found on the **Data Import/Export** pane of the Configuration Parameters dialog box determine the amount of data available to the To Workspace block. For example, if you need to ensure that data is written at identical time points over multiple simulations, select the `Produce specified output only` option in the Configuration Parameters dialog box and enter the desired time vector.

Block parameters then control when and how much of this data the To Workspace block writes:

- 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 "How to Specify the 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. This is the only format output by a To

Workspace block that can be read directly by a From Workspace block. Structure with Time format does not support frame-based signals: use Array or Structure format instead.

Examples

In a simulation where the start time is 0, the **Limit data points to last** 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 **Limit data points to last** 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).

Saving Data for Use by a From File Block

The From File block can read data written by a To Workspace block subject to the following requirements:

- 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 “Data Import/Export Pane” for more information.
- The data must be the transposition of the data saved to the workspace by the To Workspace block. Before saving the data to a MAT-file, transpose it to the form expected by the From File block.
- The data in the file must be scalar or vector data of type `double`.

Saving Data for Use by a From Workspace Block

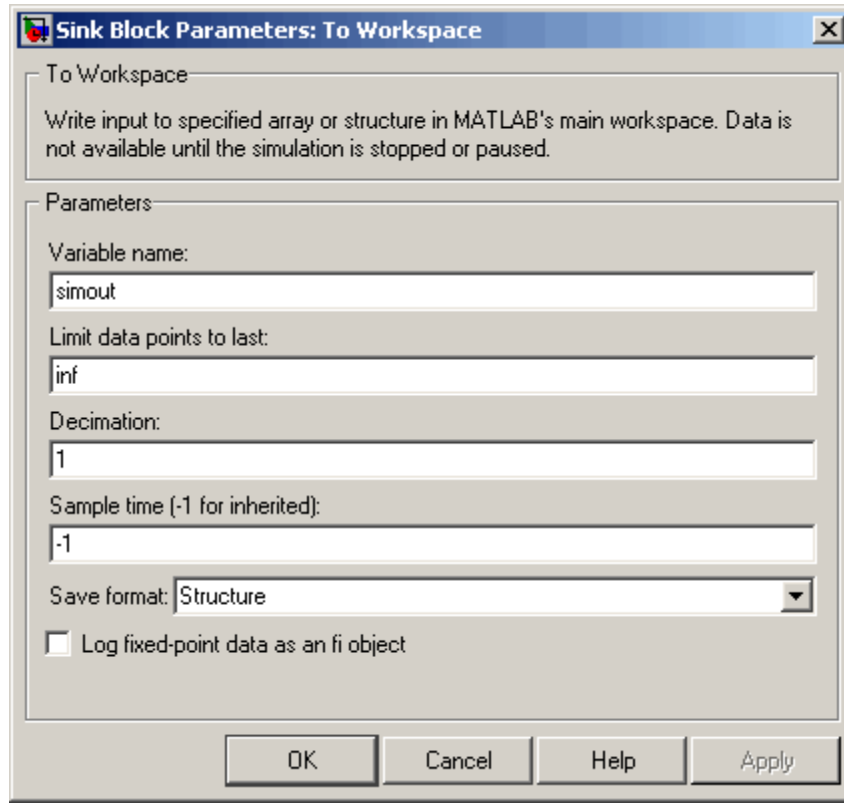
In a To Workspace block, use the `Structure with Time` format to save sample-based data if you intend to use a From Workspace block to play back the data in another simulation.

Data Type Support

The To Workspace block can save real or complex inputs of any data type supported by Simulink software, including fixed-point and enumerated data types, to the MATLAB workspace.

For a discussion on the data types supported by Simulink software, 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 `inf` samples.

Decimation
A decimation factor. The default is `1`.

Sample time
The sample time at which to collect points. See "How to Specify the Sample Time" 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
Multidimensionalized	Yes

See Also

"Importing and Exporting Data", From File, From Workspace, To File

Transfer Fcn

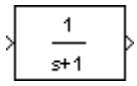
Purpose

Model linear system by transfer function

Library

Continuous

Description



The Transfer Fcn block models a linear system by a transfer function of the Laplace-domain variable *s*. The block can model single-input single-output (SISO) and single-input multiple output (SIMO) systems.

Conditions for Using This Block

The Transfer Fcn block assumes the following conditions:

- The transfer function has the form

$$H(s) = \frac{y(s)}{u(s)} = \frac{num(s)}{den(s)} = \frac{num(1)s^{nn-1} + num(2)s^{nn-2} + \ldots + num(nn)}{den(1)s^{nd-1} + den(2)s^{nd-2} + \ldots + den(nd)}$$

  where *u* and *y* are the system input and outputs, respectively, and *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*.

- The order of the denominator must be greater than or equal to the order of the numerator.
- For a multiple-output system, all transfer functions have the same denominator and all numerators have the same order.

Modeling a Single-Output System

For a single-output system, the input and output of the block are scalar time-domain signals. To model this system:

1. Enter a vector for the numerator coefficients of the transfer function in the **Numerator coefficients** field.
2. Enter a vector for the denominator coefficients of the transfer function in the **Denominator coefficients** field.

Modeling a Multiple-Output System

For a multiple-output system, the block input is a scalar and the output is a vector, where each element is an output of the system. To model this system:

1. Enter a matrix in the **Numerator coefficients** field.

   Each *row* of this matrix contains the numerator coefficients of a transfer function that determines one of the block outputs.

2. Enter a vector of the denominator coefficients common to all transfer functions of the system in the **Denominator coefficients** field.

Specifying Initial Conditions

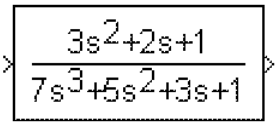
Initial conditions are preset to zero. To specify initial conditions, convert to state-space form using `tf2ss` and use the State-Space block. The `tf2ss` utility provides the A, B, C, and D matrices for the system. For more information, type `help tf2ss` or see the Control System Toolbox™ documentation.

Transfer Function Display on the Block

The Transfer Fcn block displays the transfer function depending on how you specify the numerator and denominator parameters.

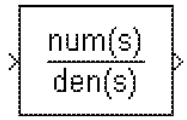
- If you specify each parameter as an expression or a vector, the block shows the transfer function with the specified coefficients and powers of *s*. If you specify a variable in parentheses, the block evaluates the variable.

  For example, if you specify **Numerator coefficients** as `[3,2,1]` and **Denominator coefficients** as `(den)`, where `den` is `[7,5,3,1]`, the block looks like this:



- If you specify each parameter as a variable, the block shows the variable name followed by `(s)`.

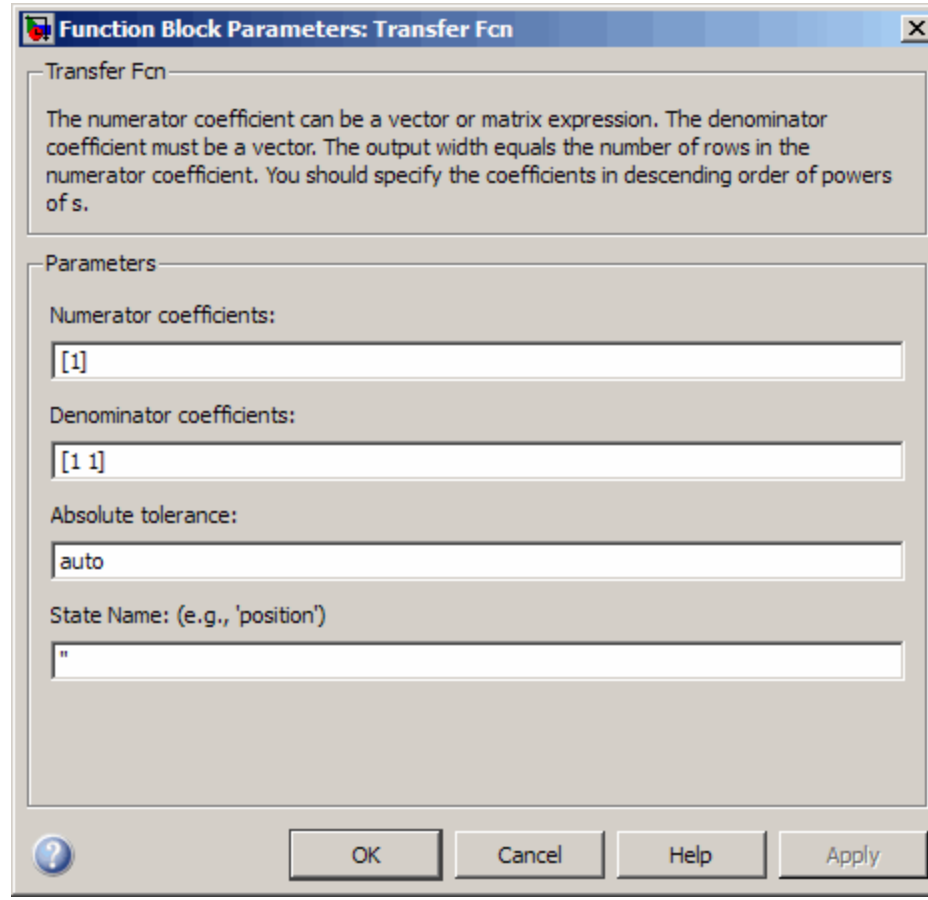
  For example, if you specify **Numerator coefficients** as `num` and **Denominator coefficients** as `den`, the block looks like this:



Data Type Support

The Transfer Fcn block accepts and outputs signals of type `double`.

Parameters and Dialog Box



Numerator coefficients

Define the row vector of numerator coefficients.

Settings

Default: `[1]`

Tips

- For a single-output system, enter a vector for the numerator coefficients of the transfer function.
- For a multiple-output system, enter a matrix. Each row of this matrix contains the numerator coefficients of a transfer function that determines one of the block outputs.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Denominator coefficients

Define the row vector of denominator coefficients.

Settings

Default: `[1 1]`

Tips

- For a single-output system, enter a vector for the denominator coefficients of the transfer function.
- For a multiple-output system, enter a vector containing the denominator coefficients common to all transfer functions of the system.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Absolute tolerance

Specify the absolute tolerance for computing the block output.

Settings

Default: `auto`

- You can enter `auto` or a numeric value.
- If you enter `auto`, Simulink uses the absolute tolerance value in the Configuration Parameters dialog box (see "Solver Pane") to compute the block output.
- If you enter a numeric value, that value overrides the absolute tolerance in the Configuration Parameters dialog box.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

State Name (e.g., 'position')

Assign a unique name to each state.

Settings

Default: `' '`

If this field is blank, no name assignment occurs.

Tips

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'`.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, `{'a', 'b', 'c'}`. Each name must be unique.
- The state names apply only to the selected block.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

  For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.
- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell array, or structure.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Direct Feedthrough	Only if the lengths of the **Numerator coefficients** and **Denominator coefficients** parameters are equal
Sample Time	Continuous
Scalar Expansion	No
States	Length of **Denominator coefficients** -1
Dimensionalized	Yes, the block expands scalar input into vector output when the transfer function numerator is a matrix.
Zero-Crossing Detection	No

See Also

Discrete Transfer Fcn

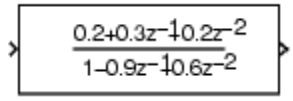
Purpose

Implement Direct Form II realization of transfer function

Library

Additional Math & Discrete / Additional Discrete

Description



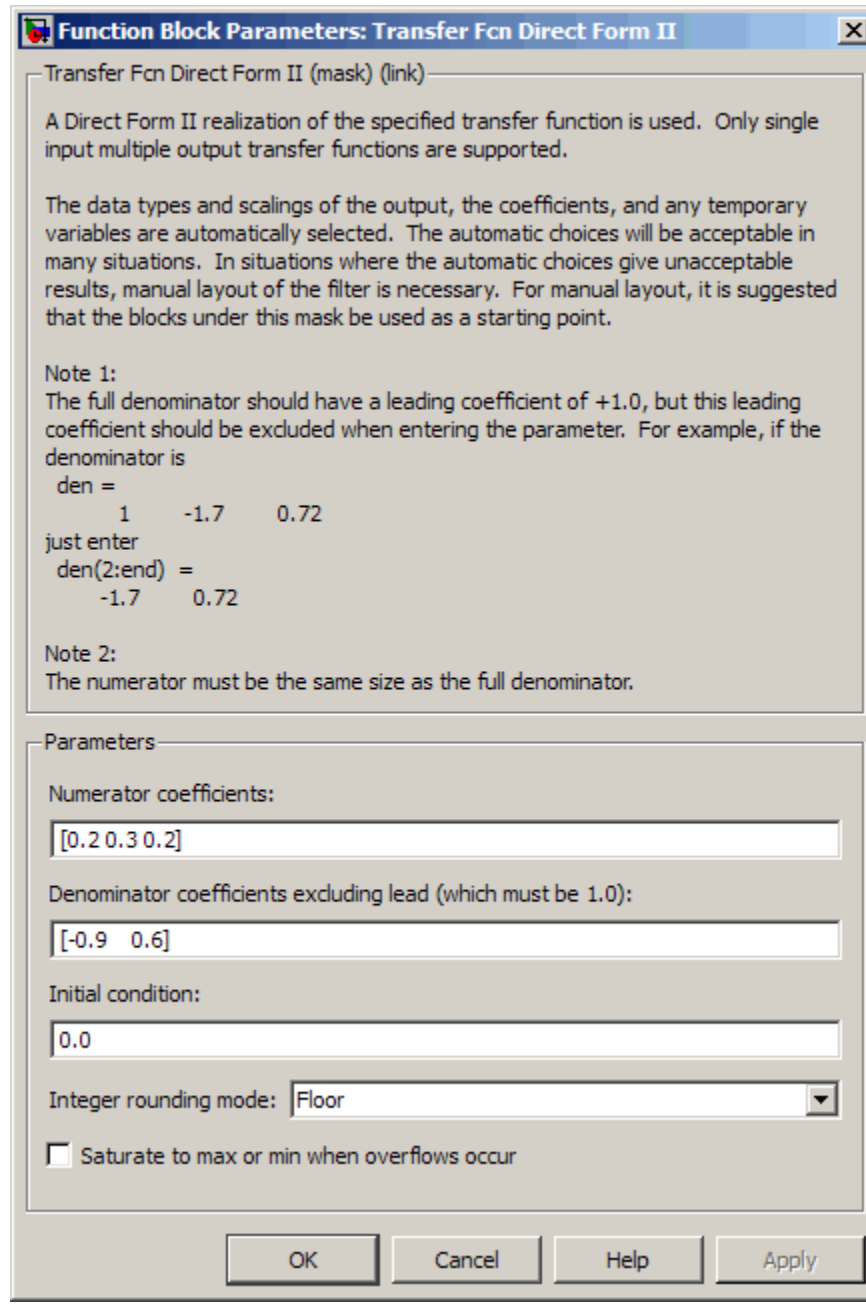
The Transfer Fcn Direct Form II block implements a Direct Form II realization of the transfer function specified by the **Numerator coefficients** and the **Denominator coefficients excluding lead** parameters. The block only supports single input-single output transfer functions.

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 numeric data type supported by Simulink software, 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.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

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

Transfer Fcn Direct Form II Time Varying

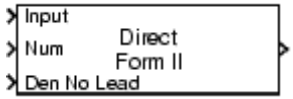
Purpose

Implement time varying Direct Form II realization of 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 supports only single input-single output (SISO) transfer functions.

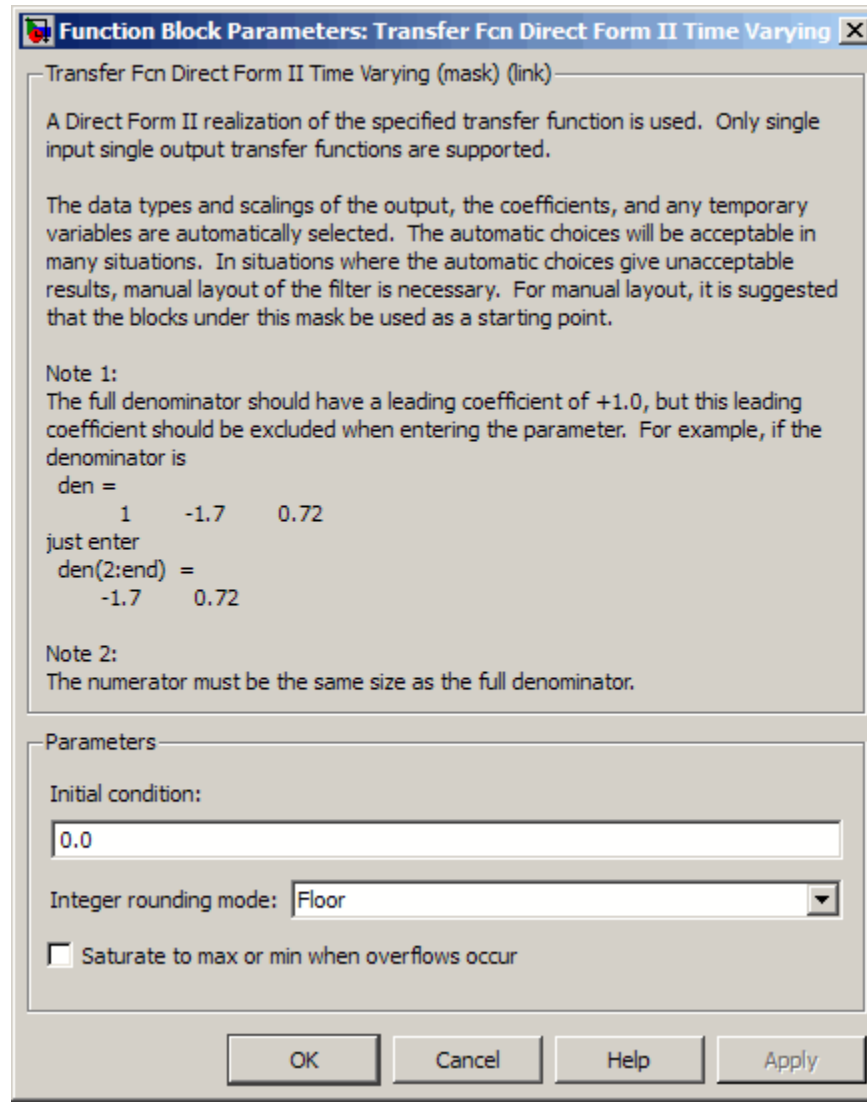
The input signal labeled `Den No Lead` contains the denominator coefficients of the transfer function. The full denominator has a leading coefficient of one, but it is excluded from the input signal. For example, a denominator of [1 -1.7 0.72] is represented by a signal with the value [-1.7 0.72]. The input signal labeled `Num` contains the numerator coefficients. The data types of the numerator and denominator coefficients can be different, but the length of the numerator vector and the full denominator vector must be the same. Pad the numerator vector with zeros, if needed.

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 numeric data type supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box



Initial condition
Set the initial condition.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

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

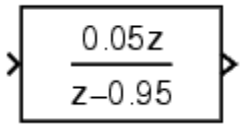
Purpose

Implement discrete-time first order transfer function

Library

Discrete

Description

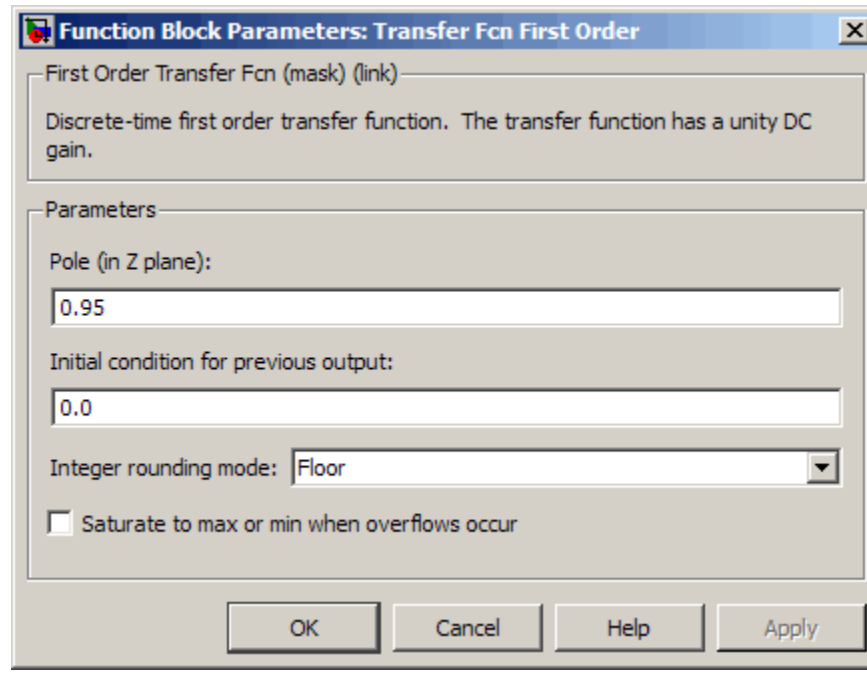


The Transfer Fcn First Order block implements a discrete-time first order transfer function of the input. The transfer function has a unity DC gain.

Data Type Support

The Transfer Fcn First Order block accepts signals of any numeric data type supported by Simulink software, 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.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

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

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

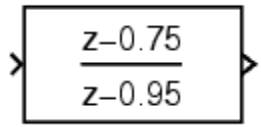
Purpose

Implement discrete-time lead or lag compensator

Library

Discrete

Description



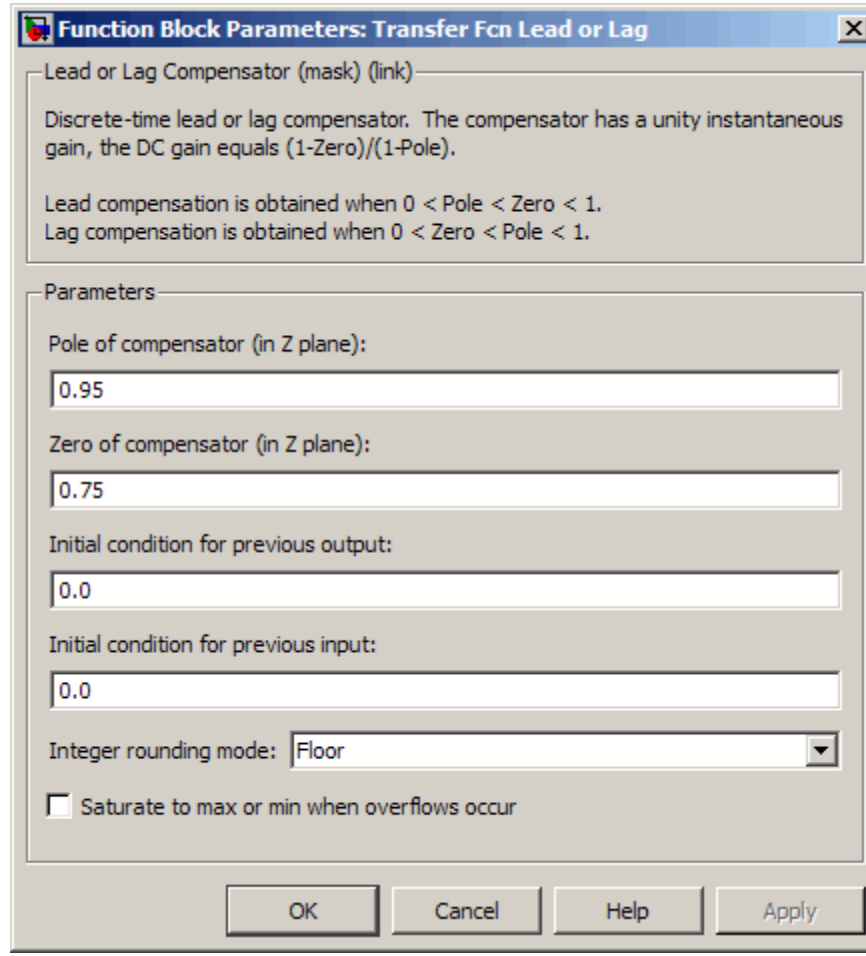
The Transfer Fcn Lead or Lag block implements a discrete-time lead or lag compensator of the input. The instantaneous gain of the compensator is one, and the DC gain is equal to (1-z)/(1-p), where z is the zero and p is the pole of the compensator.

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 numeric data type supported by Simulink software, 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.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

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

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

Transfer Fcn Real Zero

Purpose

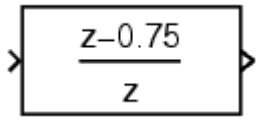
Implement discrete-time transfer function that has real zero and no pole

Library

Discrete

Description

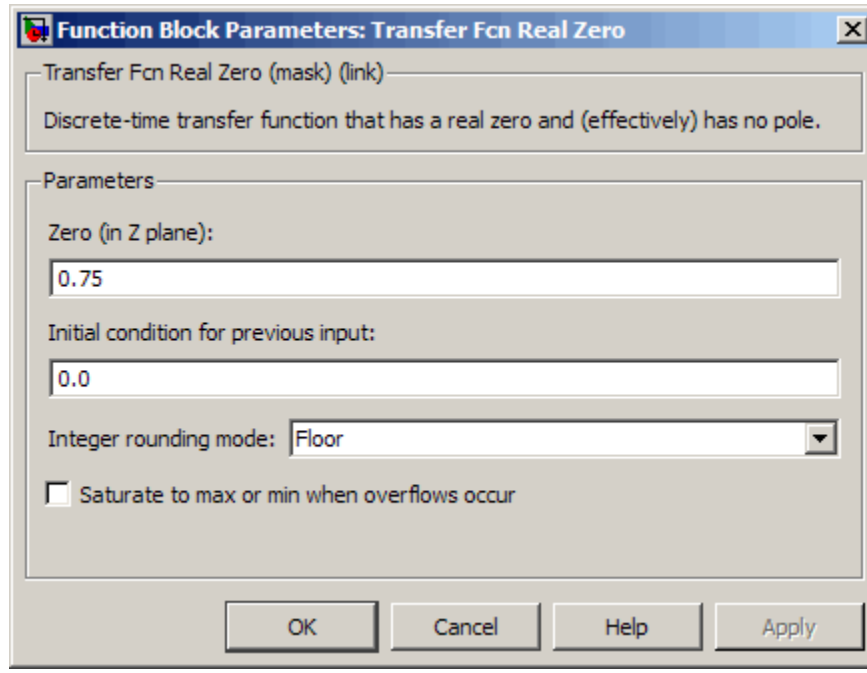
The Transfer Fcn Real Zero block implements a discrete-time transfer function that has a real zero and effectively has no pole.



Data Type Support

The Transfer Fcn Real Zero block accepts signals of any numeric data type supported by Simulink software, 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.

Integer rounding mode
Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

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

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

Transport Delay

Purpose

Delay input by given amount of time

Library

Continuous

Description

The Transport Delay block delays the input by a specified amount of time. You can use this block to simulate a time delay.

At the start of simulation, the block outputs the **Initial output** parameter until the simulation time exceeds the **Time delay** parameter. Then, the block begins generating the delayed input. During simulation, the block stores input points and simulation times in a buffer. You specify this size with the **Initial buffer size** parameter.

When you want output at a time that does not correspond to 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 an output value. 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 must 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, the block returns the discrete value at the required time.

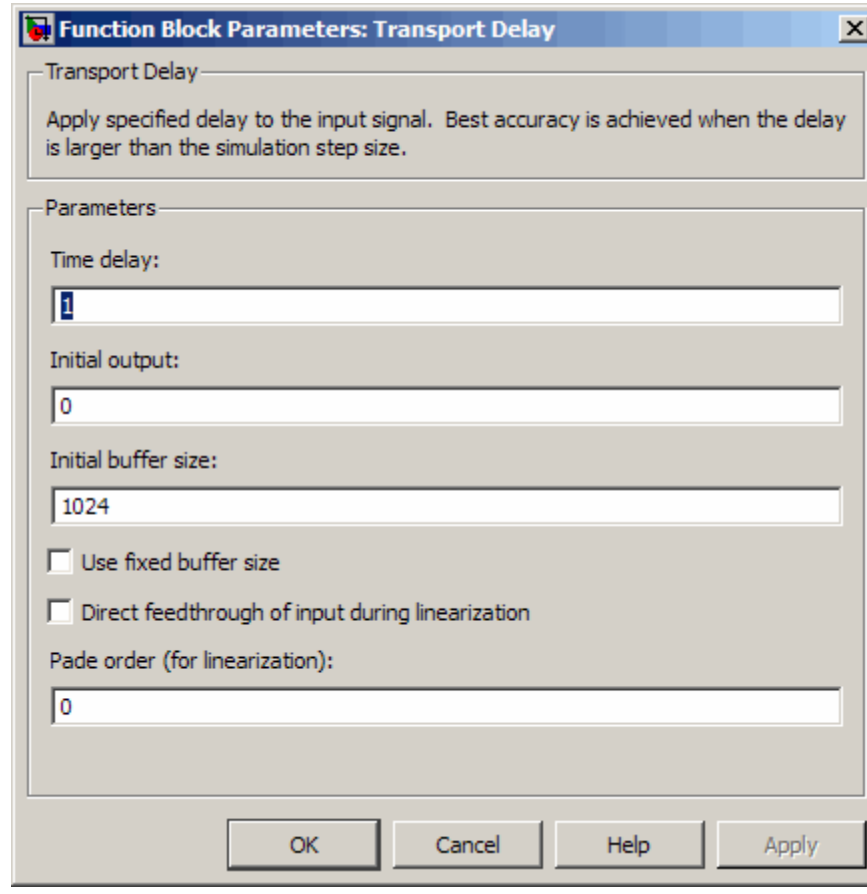
This block differs from the Unit Delay block, which delays and holds the output on sample hits only.

Tip Avoid using `linmod` to linearize a model that contains a Transport Delay block. For more information, see "Linearizing Models" in the Simulink documentation.

Data Type Support

The Transport Delay block accepts and outputs real signals of type `double`.

Parameters and Dialog Box



Time delay

Specify the amount of simulation time to delay the input signal before propagation to the output.

Settings

Default: 1

This value must be nonnegative.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Initial output

Specify the output that the block generates until the simulation time first exceeds the time delay input.

Settings

Default: `0`

The initial output of this block cannot be `inf` or `NaN`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Initial buffer size

Define the initial memory allocation for the number of input points to store.

Settings

Default: `1024`

- If the number of input points exceeds the initial buffer size, the block allocates additional memory.
- After simulation ends, a message shows the total buffer size needed.

Tips

- Because allocating memory slows down simulation, choose this value carefully if simulation speed is an issue.
- For long time delays, this block can use a large amount of memory, particularly for dimensionalized input.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Use fixed buffer size

Specify use of a fixed-size buffer to save input data from previous time steps.

Settings

Default: Off

 On

The block uses a fixed-size buffer.

 Off

The block does not use a fixed-size buffer.

The **Initial buffer size** parameter specifies the size of the buffer. If the buffer is full, new data replaces data already in the buffer. Simulink software uses linear extrapolation to estimate output values that are not in the buffer.

Note ERT or GRT code generation uses a fixed-size buffer even if you do not select this check box.

Tips

- If the input data is linear, selecting this check box can save memory.
- If the input data is nonlinear, do not select this check box. Doing so can yield inaccurate results.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Direct feedthrough of input during linearization

Cause the block to output its input during linearization and trim, which sets the block mode to direct feedthrough.

Settings

Default: Off

 On

Enables direct feedthrough of input.

 Off

Disables direct feedthrough of input.

Tips

- Selecting this check box can cause a change in the ordering of states in the model when you use the functions `linmod`, `dlinmod`, or `trim`. To extract this new state ordering:

  1. Compile the model using the following command, where `model` is the name of the Simulink model.

     ```
     [sizes, x0, x_str] = model([],[],[],'lincompile');
     ```

  2. 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 you pass a vector of states as input to the `linmod`, `dlinmod`, or `trim` functions, the state vector must use this new state ordering.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Pade order (for linearization)

Set the order of the Pade approximation for linearization routines.

Settings

Default: `0`

- 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.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Direct Feedthrough	No
Sample Time	Continuous
Scalar Expansion	Yes, of input and all parameters except **Initial buffer size**
Dimensionalized	Yes
Zero-Crossing Detection	No

See Also

Variable Time Delay

Trigger

Purpose

Add trigger port to subsystem or function-call model

Library

Ports & Subsystems

Description

Adding a Trigger block to a subsystem or a model allows an external signal to trigger its execution. You can configure the Trigger block to execute the subsystem:

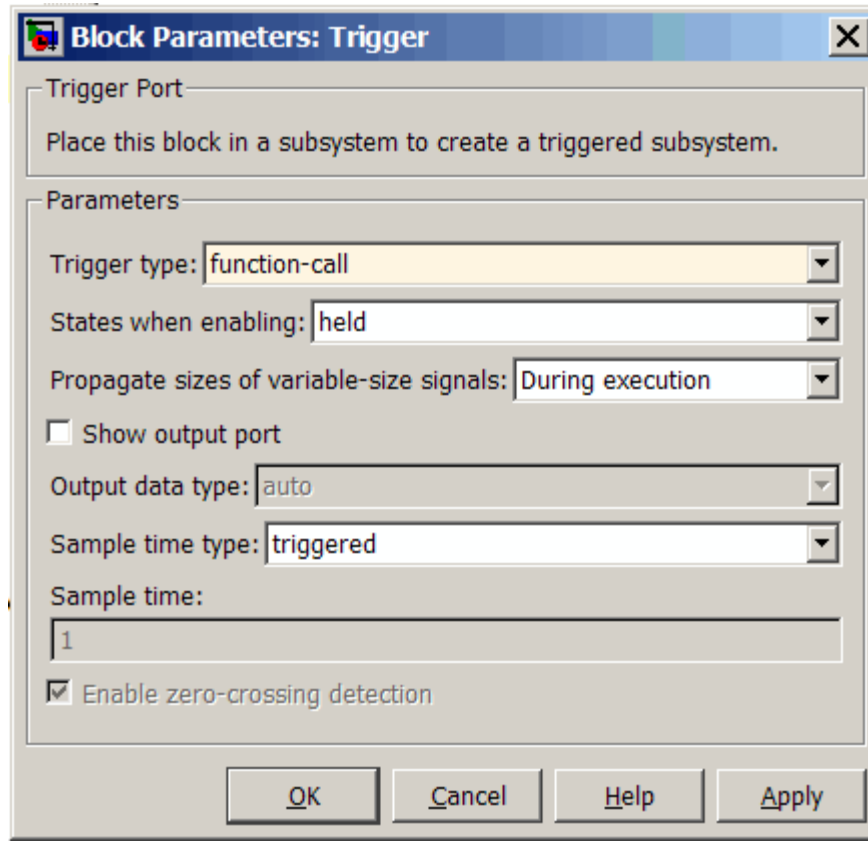
- Once on each integration step when the value of the external signal changes in a specifiable way.
- Multiple times during a time step when the external signal is a function-call from a Function-Call Generator block or S-function.

A subsystem or model can contain only one Trigger block. For more information, see “Defining Function-Call Models”, “Function-Call Subsystems”, and “Triggered Subsystems”.

Data Type Support

The Trigger block accepts signals of any numeric data type supported by Simulink software, including fixed-point data types. For more information, see “Data Types Supported by Simulink” in the Simulink documentation.

Parameters and Dialog Box



- "Trigger type" on page 2-1381
- "States when enabling" on page 2-1382
- "Propagate sizes of variable-size signals" on page 2-1384
- "Show output port" on page 2-1385
- "Output data type" on page 2-1386
- "Sample time type" on page 2-1387

Trigger type

Select the type of event that triggers execution of the subsystem.

Settings

Default: `rising`

`rising`
: Triggers execution of the subsystem when the control signal rises from a negative or zero value to a positive value. If the initial value is negative, rising to zero triggers execution.

`falling`
: Triggers execution of the subsystem when the control signal falls from a positive or a zero value to a negative value. If the initial value is positive, falling to zero triggers execution.

`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. The **Trigger type** must be `function-call` for Trigger ports at the root-level of a model. Only function-call signals can trigger execution of a model.

Command-Line Information

Parameter: `TriggerType`
Type: string
Value: `'rising' | 'falling' | 'either' | 'function-call'`
Default: `'rising'`

States when enabling

Specify the state values when triggered by a function-call.

Settings

Default: `held`

`held`
: Leaves the states at their current values.

`reset`
: Resets the states.

`inherit`
: Uses the `held`/`reset` setting from the parent subsystem initiating the function-call. If the parent of the initiator is the model root, the inherited setting is `held`. If the trigger has multiple initiators, set the parents of all initiators to either `held` or `reset`.

Dependencies

If you select `function-call` as the block trigger type, the dialog enables this parameter. The parameter setting applies only if the model explicitly enables and disables the function-call subsystem. For example:

- The function-call subsystem resides in an enabled subsystem. In this case, the model enables and disables the function-call subsystem along with the parent subsystem.
- The function-call initiator that controls the function-call subsystem resides in an enabled subsystem. In this case, the model enables and disables the function-call subsystem along with the enabled subsystem containing the function-call initiator.
- The function-call initiator is a Stateflow event bound to a particular state. See "Using Bind Actions to Control Function-Call Subsystems" in the Stateflow documentation.
- The function-call initiator is an S-function that explicitly enables and disables the function-call subsystem. See `ssEnableSystemWithTid` for an example.

Command-Line Information

Parameter: `StatesWhenEnabling`
Type: string
Value: `'held' | 'reset' | 'inherit'`
Default: `'held'`

Propagate sizes of variable-size signals

Specify when to propagate a variable-size signal.

Settings

Default: `During execution`

`Only when enabling`
: Propagates variable-size signals only when enabling the subsystem containing the Trigger Port block.

`During execution`
: Propagates variable-size signals at each time step.

Dependencies

Selecting `Function-call` from the **Trigger type** list enables this parameter.

Command-Line Information

Parameter: `PropagateVarSize`
Type: string
Value: `'Only when enabling' | 'During execution'`
Default: `'During execution'`

Show output port

Select this check box to output the trigger signal.

Settings

Default: On

 On

Shows the Trigger block output port and outputs the trigger signal if this block is in a subsystem. This action 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

 Off

Removes the output port.

Dependencies

This dialog box disables this parameter for function-call Trigger blocks residing at the root level of a model.

Command-Line Information

Parameter: `ShowOutputPort`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

Output data type

Specify the trigger output data type.

Settings

Default: Auto

 Auto

Specifies the data type is the same as the port connected to output.

 double

Sets the date type to double.

 int8

Sets the data type to integer.

Dependencies

Selecting `Function-call` from the **Trigger type** list and selecting the **Show output port** check box enables this parameter.

The Trigger block ignores the **Data type override** setting of the Fixed-Point Tool.

Sample time type

Specify the calling frequency of a subsystem.

Settings

Default: `triggered`

`triggered`
: Applies to applications that do not have a periodic calling frequency.

`periodic`
: Applies if the caller of the parent function-call subsystem calls the subsystem once per time step when the subsystem is active (enabled). A Stateflow chart is an example of a caller.

Dependencies

Selecting `Function-call` from the **Trigger type** list enables this parameter.

Command-Line Information

Parameter: `SampleTimeType`
Type: string
Value: `'triggered' | 'periodic'`
Default: `'triggered'`

Sample time

Specify the sample time.

Settings

Default: `1`

Set this parameter to the sample time you expect for the calling rate of the function-call subsystem containing this Trigger block. If the actual calling rate for the subsystem differs from the rate that this parameter specifies, Simulink software displays an error.

Dependencies

Setting **Trigger type** to `function-call` and **Sample time type** to `periodic` enables this parameter.

Command-Line Information

Parameter: `SampleTime`
Type: string
Value: `1`
Default: `1`

Enable zero-crossing detection

Select to enable zero-crossing detection.

Settings

Default: On

 On

Detects zero-crossings.

 Off

Does not detect zero-crossings.

Command-Line Information

Parameter: `ZeroCross`
Type: string
Value: `'on' | 'off'`
Default: `'on'`

Characteristics

Sample Time	Specified by the **Sample time** parameter if: • **Trigger type** is `function-call` • **Sample time type** is `periodic` Otherwise, specified by the signal at the trigger port.
Dimensionalized	Yes
Virtual	Yes, when the output port is *not* present For more information, see "Virtual Blocks" in the Simulink documentation.
Zero-Crossing Detection	Yes, if enabled

See Also

Function-Call Subsystem

Trigger-Based Linearization

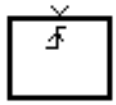
Purpose

Generate linear models in 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 block sets the following model parameters to the indicated values:

- `BufferReuse = 'off'`
- `RTWInlineParameters = 'on'`
- `BlockReductionOpt = 'off'`

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

Field	Description
`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.

You can use state and simulation time logging to extract the model states at operating points. For example, suppose that you want to get the states of the vdp demo model when the signal `x1` triggers the Trigger-Based Linearization block on a rising edge.

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. Connect the block's trigger port to the signal labeled `x1`.
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: `vdp_Trigger_Based_Linearization`, `tout`, and `xout`.
8. Get the index to the first operating point time by entering the following at the MATLAB command line:

```
ind1 = find(vdp_Trigger_Based_Linearization(1).OperPoint.t==tout);
```

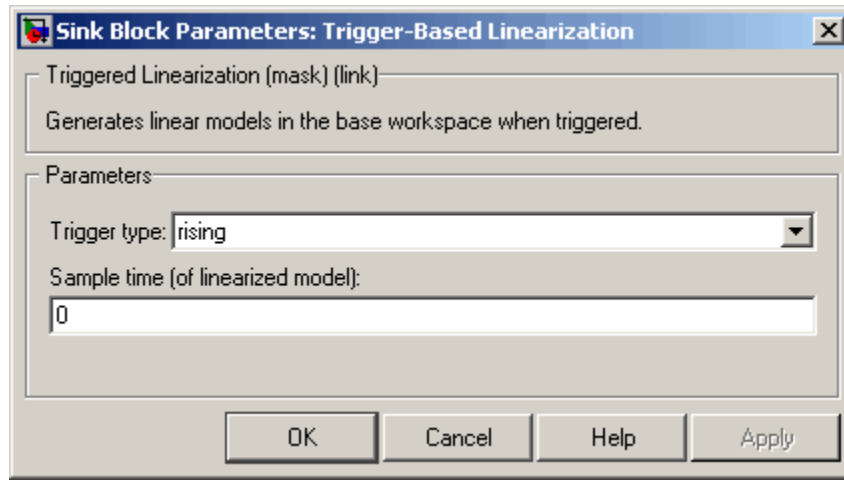
9 Get the state vector at this operating point.

```
x1 = xout(ind1,:);
```

Data Type Support

The trigger port accepts signals of any numeric data type supported by Simulink software.

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 4-130).

Characteristics

Sample Time	Specified in the **Sample time** parameter
Dimensionalized	No

See Also

Time-Based Linearization

Triggered Subsystem

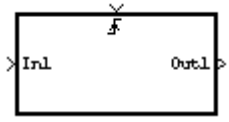
Purpose

Represent 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 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`, `atanh`, and `sincos`. The name of the function appears on the block. Each block output is the result of the function operating on one or more inputs.

If you select the `atan2` function, the block displays two inputs. The first input is the *y*-axis or complex part of the function argument. The second input is the *x*-axis or real part of the function argument. (See "How to Rotate a Block" in the Simulink User's Guide for a description of the port order for various block orientations.)

If you select the `sincos` function, the block displays two outputs. The first output is the sine of the function argument, and the second output is the cosine of the function argument.

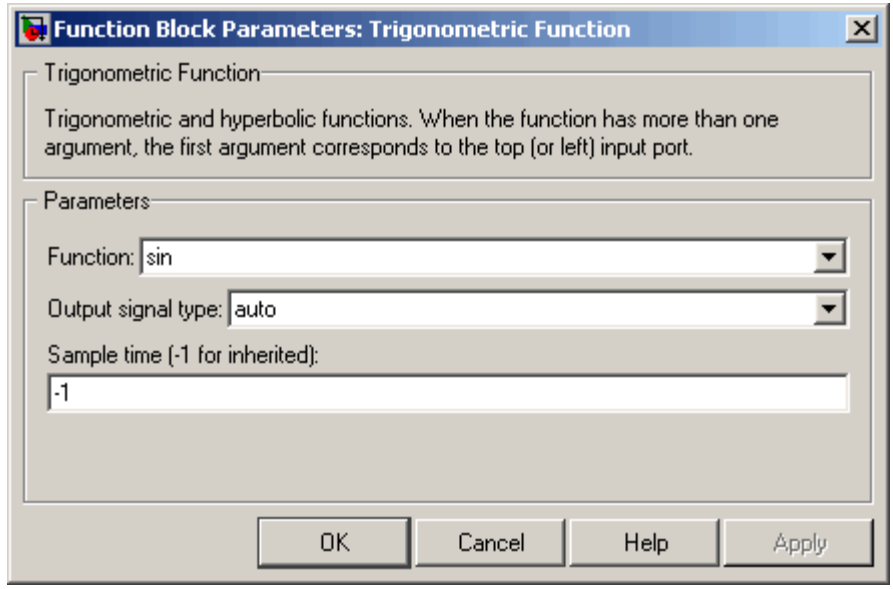
Use the Trigonometric Function block instead of the Fcn block when you want dimensionalized output, because the Fcn block can produce only scalar output.

Note If you use a compiler that does not support the `asinh`, `acosh`, and `atanh` functions, the Real-Time Workshop software issues a warning for the Trigonometric Function block and the generated code fails to link. For more information, see "Simulink Built-In Blocks That Support Code Generation" in the Real-Time Workshop documentation.

Data Type Support

The Trigonometric Function block accepts and outputs real or complex signals of type `single` or `double`.

Parameters and Dialog Box



Function
The trigonometric function.

Output signal type
Type of signal (complex or real) to output.

> **Note** The Trigonometric Function block cannot output complex signals in Real-Time Workshop generated code. The imaginary part of a complex signal is always set to zero in the generated code.

Sample time (-1 for inherited)
Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the Sample Time" in the Simulink User's Guide 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
Multidimensionalized	Yes
Zero Crossing	No

Unary Minus

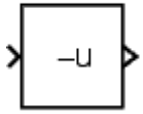
Purpose

Negate input

Library

Math Operations

Description



The Unary Minus block negates the input. The block accepts only signed data types.

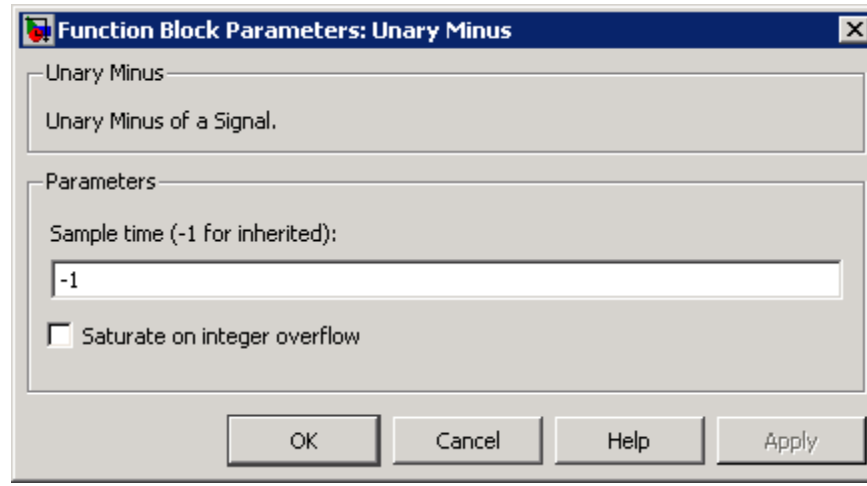
For signed data types, you cannot accurately negate the most negative value because the data type cannot represent the result. In this case, the **Saturate on integer overflow** check box controls the behavior of the block. If you select the check box, the most negative value of the data type wraps to the most positive value. If you do not select the check box, the operation has no effect. If an overflow occurs, a warning appears at 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 on integer overflow** check box, the negation of -128 is 127. If you do not select the check box, the negation of -128 remains at -128.

Data Type Support

The Unary Minus block accepts signals of any numeric data type that Simulink supports, including fixed-point data types, but not unsigned integer types.

Parameters and Dialog Box



Sample time (-1 for inherited)

Specify the time interval between samples. To inherit the sample time, set this parameter to -1. See "How to Specify the Sample Time" in the Simulink User's Guide for more information.

Saturate on integer overflow

If selected, fixed-point overflows saturate. Otherwise, they wrap.

Characteristics

Direct Feedthrough	No
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes, of input or initial conditions
Multidimensionalized	Yes

Uniform Random Number

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.

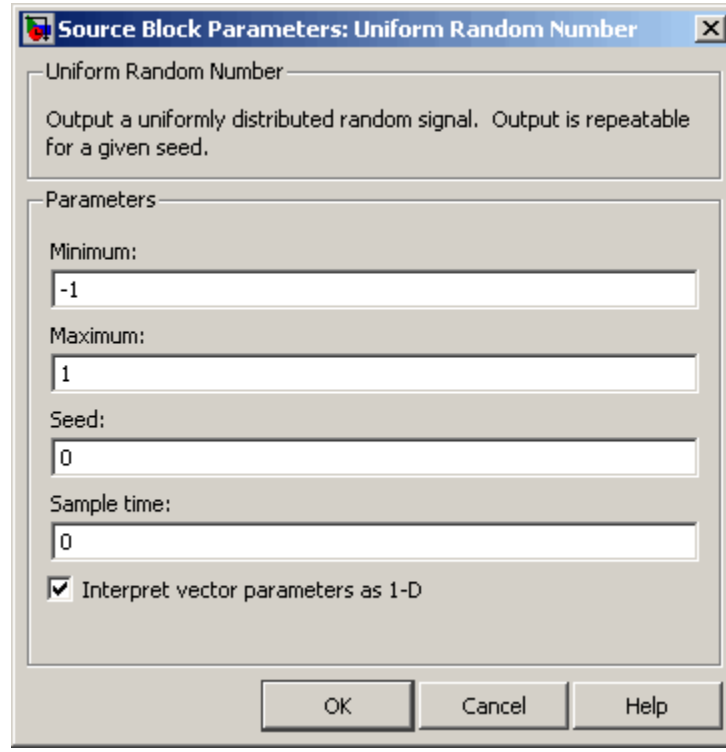
Avoid integrating a random signal, because solvers are meant to integrate relatively smooth signals. Instead, use the Band-Limited White Noise block.

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.

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 During Simulation" 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`.

Seed

The starting seed for the random number generator. The default is `0`.

Sample time

The sample period. The default is `0`. See "How to Specify the Sample Time" in the online documentation for more information.

Interpret vector parameters as 1-D

If selected, column or row matrix values for the Uniform Random Number 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. See "Determining the Output Dimensions of Source Blocks" in the "Working with Signals" chapter of the Simulink documentation.

Characteristics

Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	Yes
Multidimensionalized	Yes
Zero Crossing	No

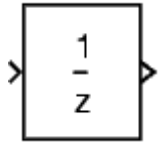
Purpose

Delay 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.

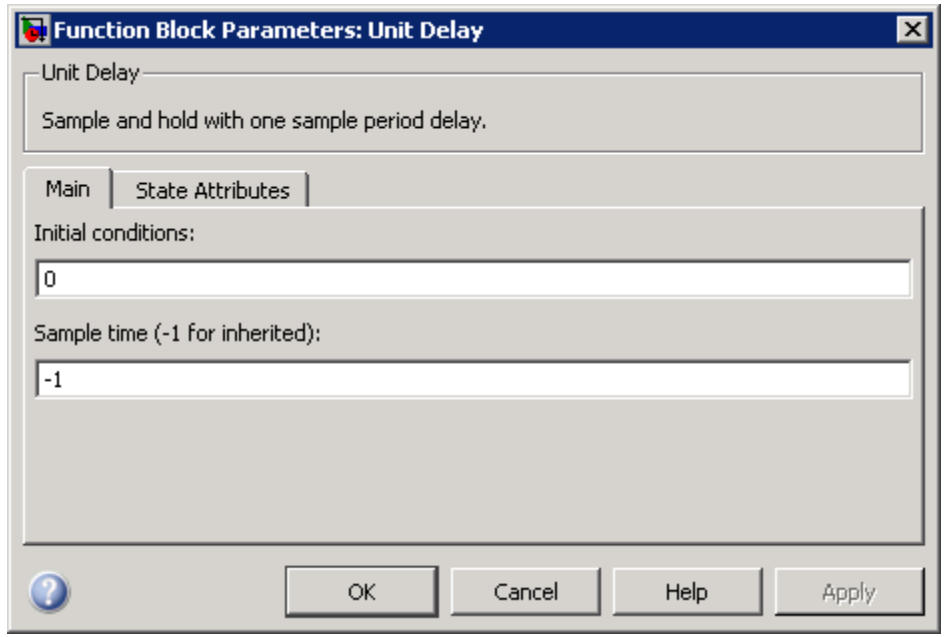
Note Do not use the Unit Delay block to create a slow-to-fast transition between blocks operating at different sample rates. Instead, use the Rate Transition block.

Data Type Support

The Unit Delay block accepts real or complex signals of any data type supported by Simulink software, including fixed-point and enumerated 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 software, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box



During simulation, the block uses the following values:

- The initial value of the signal object to which the state name is resolved
- Min and Max values of the signal object

See "Block State Storage and Interfacing Considerations" in the Real-Time Workshop documentation for more information.

Initial conditions

Specify the output of the simulation for the first sampling period, during which the output of the Unit Delay block is otherwise undefined.

Settings

Default: `0`

The **Initial conditions** parameter is converted from a double to the input data type offline using round-to-nearest and saturation.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Sample time (-1 for inherited)

Enter the discrete interval between sample time hits or specify another appropriate sample time such as continuous or inherited.

Settings

Default: -1

By default, the block inherits its sample time based upon the context of the block within the model. To set a different sample time, enter a valid sample time based upon the table in "Types of Sample Time".

See also "How to Specify the Sample Time" in the online documentation for more information.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

State name

Use this to assign a unique name to each state.

Settings

Default: `' '`

- If left blank, no name is assigned.

Tips

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'` .
- The state names apply only to the selected block.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces. For example, `{'a', 'b', 'c'}` . Each name must be unique.
- The number of states must be evenly divided by the number of state names. There can be fewer names than states, but there cannot be more names than states.
- For example, you can specify two names in a system with four states. Simulink software will assign the first name to the first two states and the second name to the last two.
- To assign state names with a variable that has been defined in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell, or structure.

Dependencies

This parameter enables **State name must resolve to Simulink signal object** when you click the **Apply** button.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

State name must resolve to Simulink signal object

Require that state name resolve to Simulink signal object.

Settings

Default: Off

 On

Require that state name resolve to Simulink signal object.

 Off

Do not require that state name resolve to Simulink signal object.

Dependencies

State name enables this parameter.

This parameter enables **Real-Time Workshop storage class**.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Real-Time Workshop storage class

Select state storage class.

Settings

Default: `Auto`

`Auto`
: Auto is the appropriate storage class for states that you do not need to interface to external code.

`ExportedGlobal`
: State is stored in a global variable

`ImportedExtern`
: *`model`*`_private.h` declares the state as an extern variable.

`ImportedExternPointer`
: *`model`*`_private.h` declares the state as an extern pointer.

Dependencies

State name enables this parameter.

Setting this parameter to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables **Real-Time Workshop storage type qualifier**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

See Also

"Block State Storage Classes" in the *Real-Time Workshop User's Guide*.

Real-Time Workshop storage type qualifier

Specify Real-Time storage type qualifier.

Settings

Default: `' '`

If left blank, no qualifier is assigned.

Dependencies

Setting **Real-Time Workshop storage class** to `ExportedGlobal`, `ImportedExtern`, or `ImportedExternPointer` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Bus Support

The Unit Delay block is a bus-capable block. The input can be a virtual or nonvirtual bus signal subject to the following restrictions:

- **Initial conditions** must be zero or a non-zero scalar.
- If **Initial conditions** is zero and a **State name** is specified, the input cannot be a virtual bus.
- If **Initial conditions** is a non-zero scalar, no **State name** can be specified.

All signals in a nonvirtual bus input to a Unit Delay block must have the same sample time, even if the elements of the associated bus object specify inherited sample times. You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus. See "Using Composite Signals" and Bus-Capable Blocks for more information.

Characteristics

Bus-capable	Yes, with restrictions as noted in "Bus Support" on page 2-1410
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
Multidimensionalized	Yes
Zero-Crossing Detection	No

See Also

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

Unit Delay Enabled

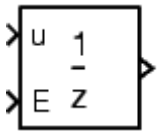
Purpose

Delay signal one sample period, if 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 of the **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.

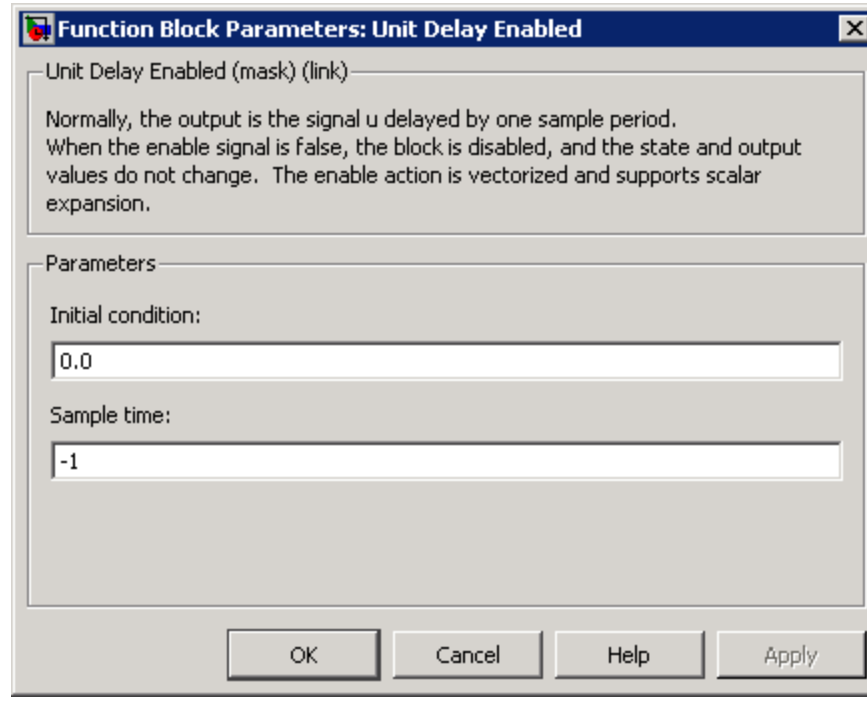
You input the sample time with the **Sample time** parameter. A setting of `-1` means that the block inherits the **Sample time**.

Data Type Support

The Unit Delay Enabled block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

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 "How to Specify the Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	No
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	Yes

Unit Delay Enabled

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

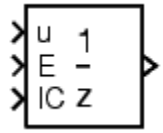
Purpose

Delay signal one sample period, if external enable signal is on, with 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 block inherits the **Sample time**.

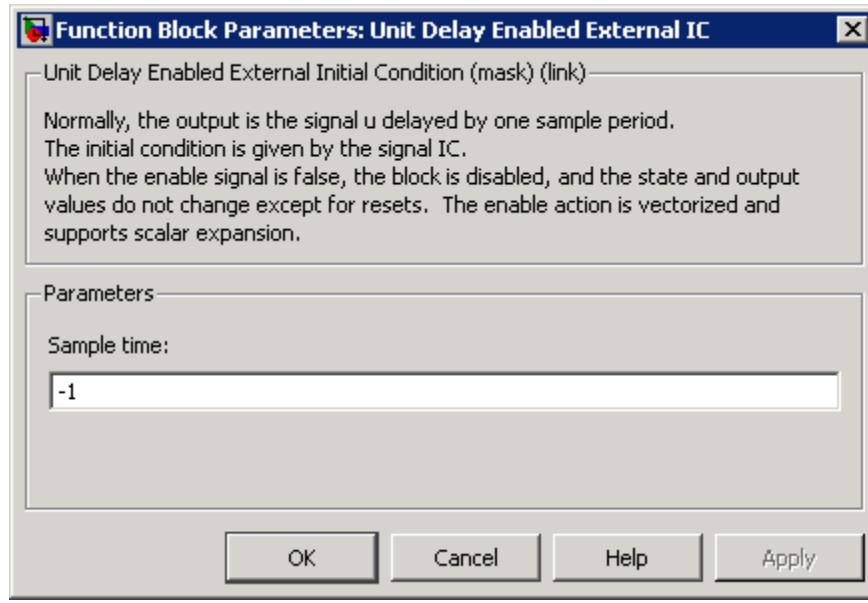
Data Type Support

The Unit Delay Enabled External IC block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

Unit Delay Enabled External IC

Parameters and Dialog Box



Sample time

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

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

Unit Delay Enabled Resettable

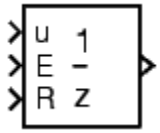
Purpose

Delay signal one sample period, if external enable signal is on, with 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.

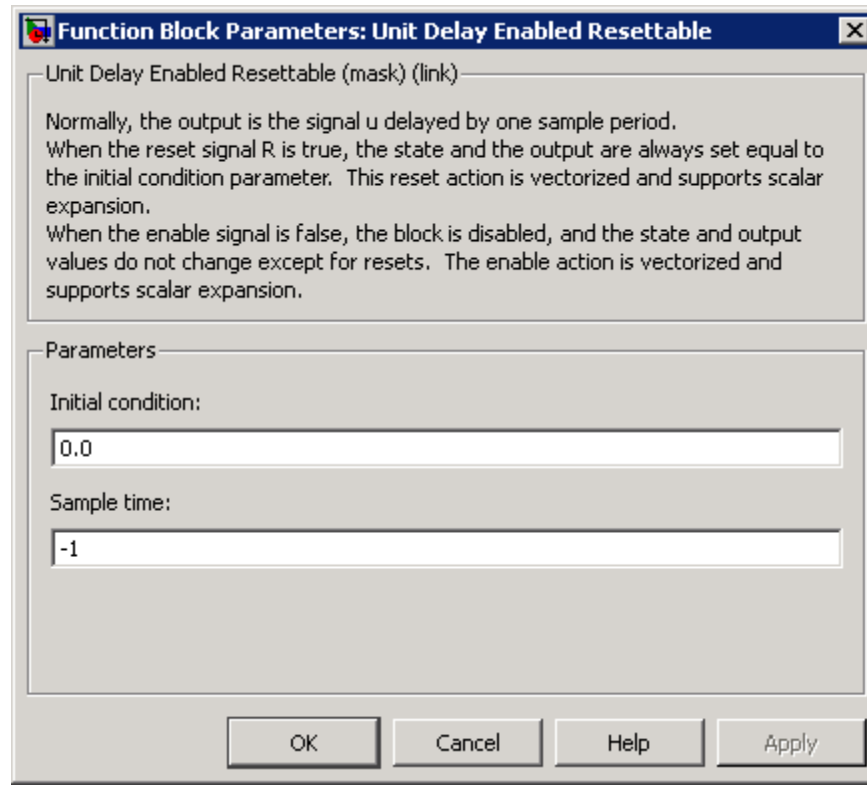
You specify the time between samples with the **Sample time** parameter. A setting of `-1` means that the block inherits the **Sample time**.

Data Type Support

The Unit Delay Enabled Resettable block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

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 "How to Specify the Sample Time" in the online documentation for more information.

Unit Delay Enabled Resettable

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

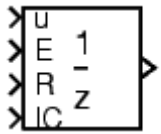
Purpose

Delay signal one sample period, if external enable signal is on, with 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 that the block inherits the **Sample time**.

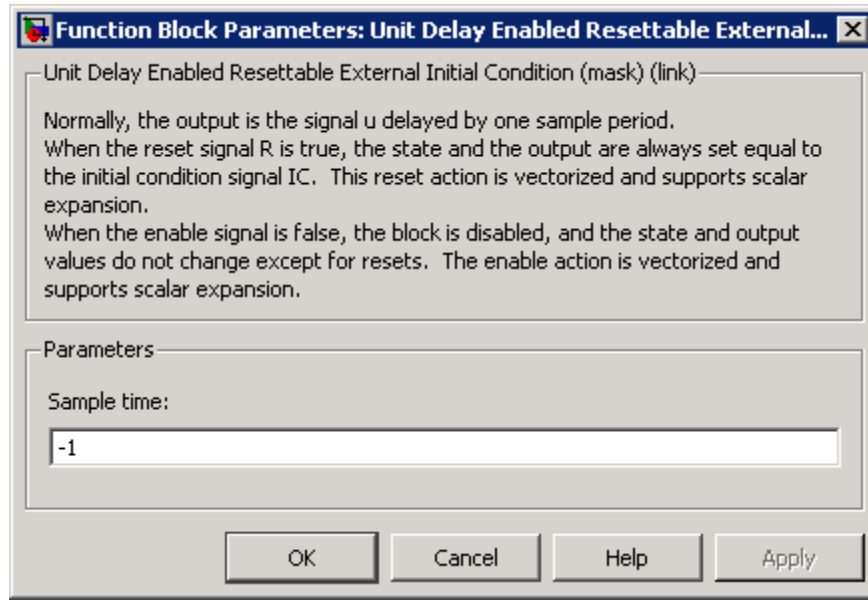
Data Type Support

The Unit Delay Enabled Resettable External IC block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

Unit Delay Enabled Resettable External IC

Parameters and Dialog Box



Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the 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

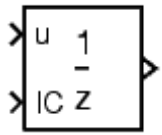
Purpose

Delay signal one sample period, with 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.

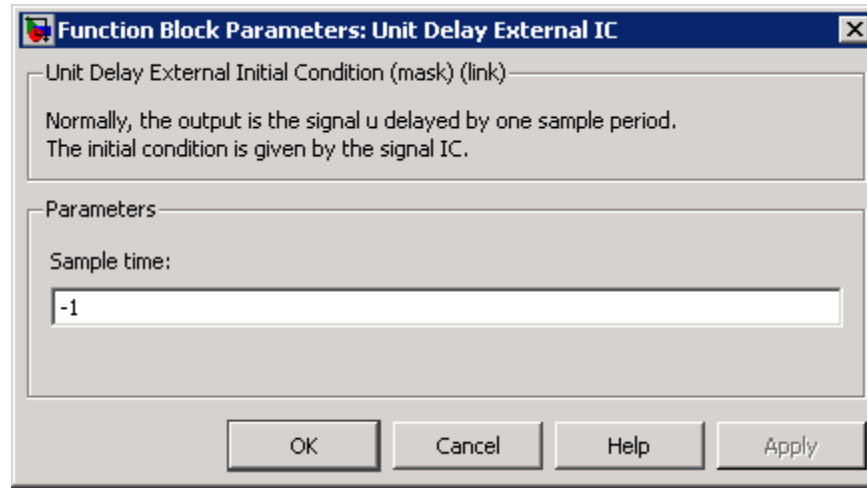
You specify the time between samples with the **Sample time** parameter. A setting of `-1` means that the block inherits the **Sample time**.

Data Type Support

The Unit Delay External IC block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

Parameters and Dialog Box



Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the 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

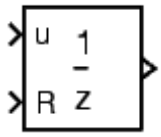
Purpose

Delay signal one sample period, with 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 both its state and output 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`.

At the start of simulation, the block's **Initial condition** parameter determines its initial output. During simulation, 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 and its output to the **Initial condition**.

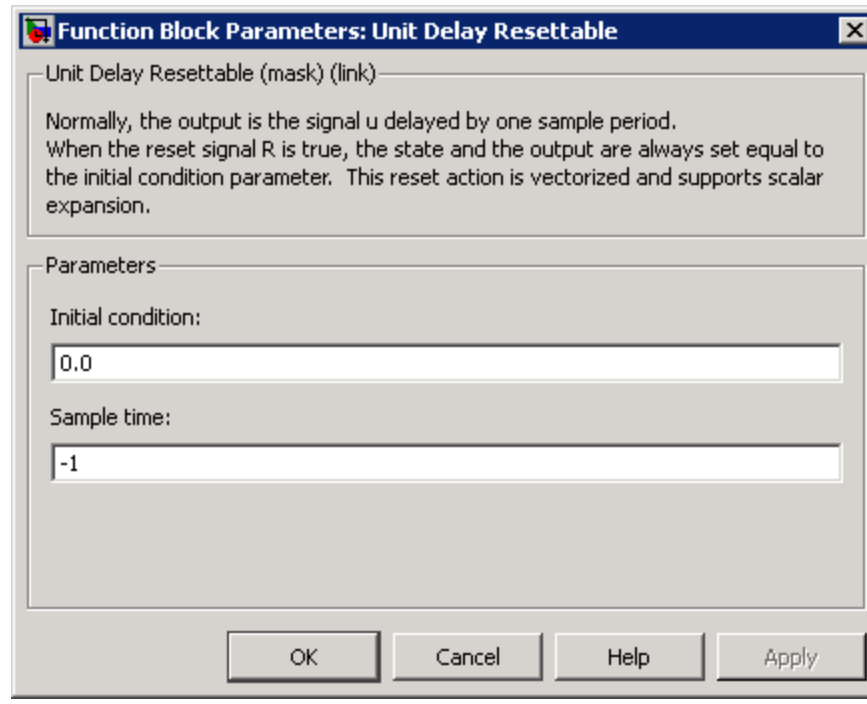
You specify the time between samples with the **Sample time** parameter. A setting of `-1` means that the block inherits the **Sample time**.

Data Type Support

The Unit Delay Resettable block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

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 "How to Specify the 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

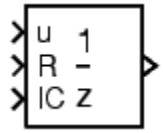
Purpose

Delay signal one sample period, with 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.

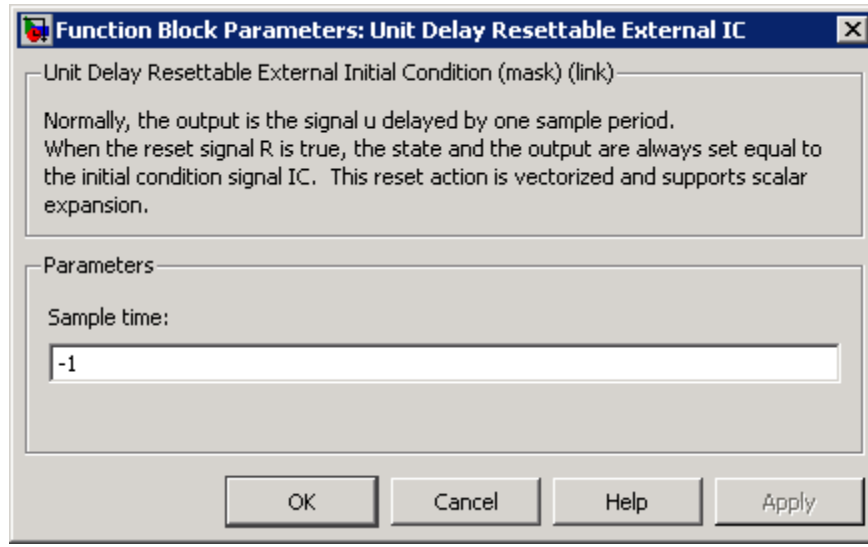
You specify the time between samples with the **Sample time** parameter. A setting of `-1` means that the block inherits the **Sample time**.

Data Type Support

The Unit Delay Resettable External IC block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

Parameters and Dialog Box



Sample time

Specify the time interval between samples. To inherit the sample time, set this parameter to `-1`. See "How to Specify the 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

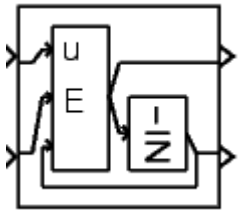
Purpose

Output signal and signal delayed by one sample period, if 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 input ports: one for the input signal `u` and one for the external enable signal `E`.

When the enable signal `E` is on, the first port outputs the signal and the second port outputs the signal delayed by one sample period. When the enable signal `E` is off, the block is disabled, and the state and output values do not change, except during resets.

The enable signal is on when `E` is not 0, and off when `E` is 0. This enable action is vectorized and supports scalar expansion.

Having two outputs is useful for implementing recursive calculations where the result includes the most recent inputs. The second output can feed back into calculations of the block's inputs without causing an algebraic loop. Meanwhile, the first output shows the most up-to-date calculations.

The **Initial condition** parameter specifies the block output for the first sampling period. The **Sample time** parameter specifies the time between samples. To inherit the sample time, set this parameter to `-1`.

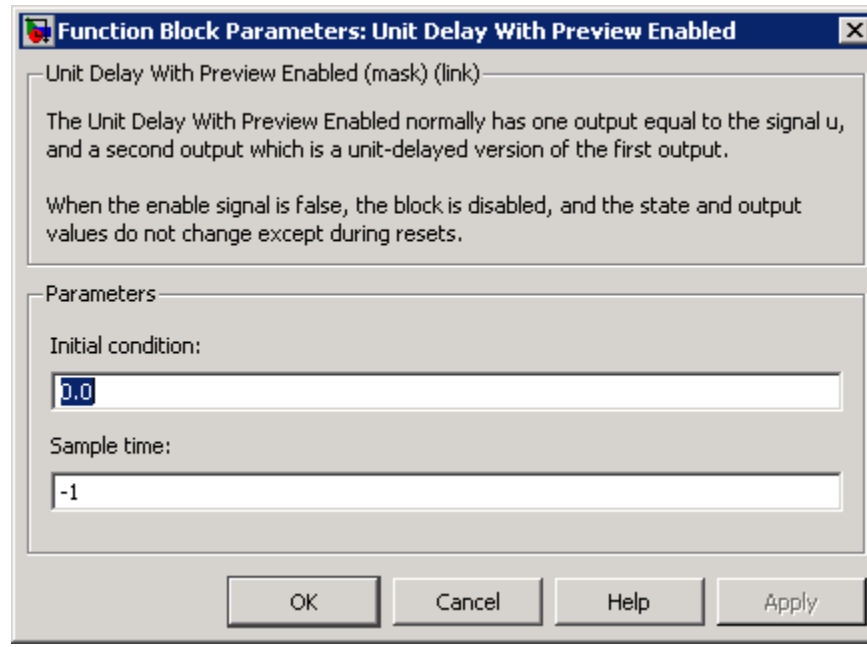
Data Type Support

The Unit Delay With Preview Enabled block accepts input signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

The outputs are the same data type as the input u.

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 "How to Specify the Sample Time" in the online documentation for more information.

Unit Delay With Preview Enabled

Characteristics

Direct Feedthrough	Yes, to first output port No, to second 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 With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

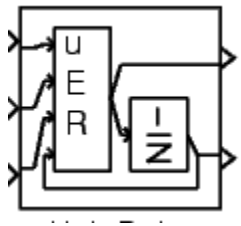
Purpose

Output signal and signal delayed by one sample period, if external enable signal is on, with external 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 a reset signal `R`.

The block has three input ports: one for the input signal `u`, one for the external enable signal `E`, and one for the external reset signal `R`.

When the enable signal `E` is on and the reset signal `R` is false, the first port outputs the signal and the second port outputs the 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 **Initial condition** parameter. The first output signal is forced to equal the initial condition. The second output signal is not affected until one time step later.

When the enable signal is off, the block is disabled, and the state and output values do not change, except during resets.

The enable signal is on when `E` is not 0, and off when `E` is 0. The enable and reset actions are vectorized and support scalar expansion.

Having two outputs is useful for implementing recursive calculations where the result includes the most recent inputs. The second output can feed back into calculations of the block's inputs without causing an algebraic loop. Meanwhile, the first output shows the most up-to-date calculations.

The **Sample time** parameter specifies the time between samples. To inherit the sample time, set this parameter to `-1`.

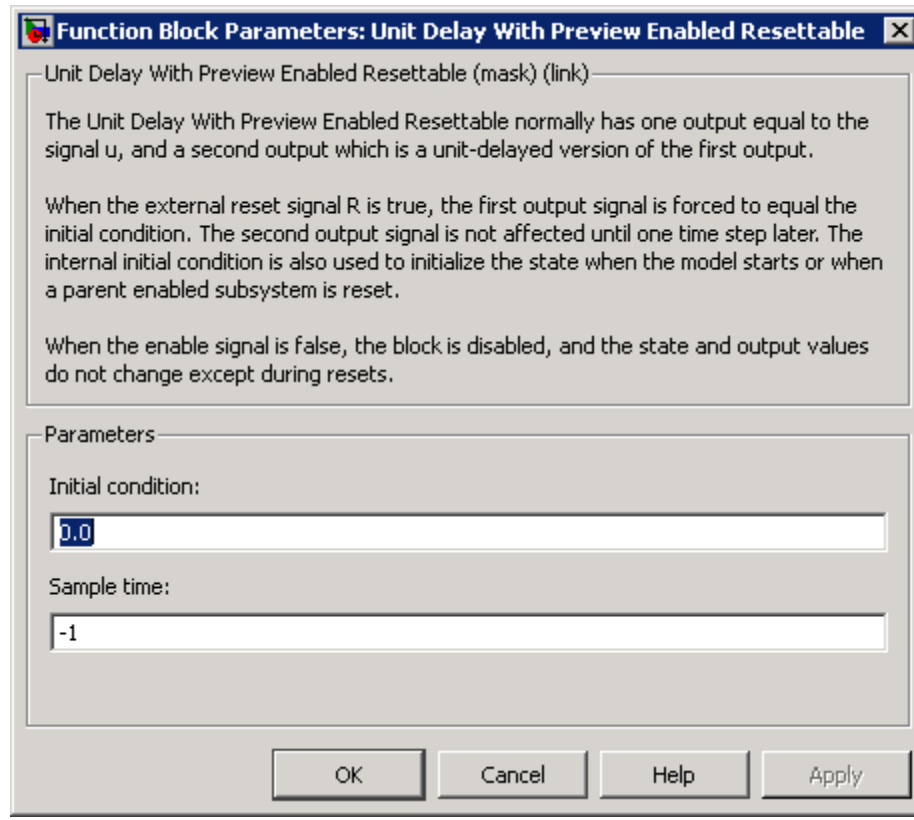
Data Type Support

The Unit Delay With Preview Enabled Resettable block accepts input signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

The outputs are the same data type as the input u.

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 "How to Specify the Sample Time" in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes, to first output port No, to second 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

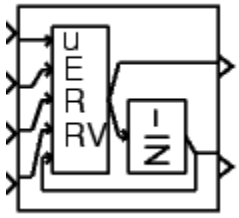
Purpose

Output signal and signal delayed by one sample period, if external enable signal is on, with 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 a reset signal `R`.

The block has four input ports: one for the input signal `u`, one for the external enable signal `E`, one for the external reset signal `R`, and one for the external reset value `RV`.

When the enable signal `E` is on and the reset signal `R` is false, the first port outputs the signal and the second port outputs the signal delayed by one sample period.

When the enable signal `E` is on and the reset signal `R` is true, the first output signal is forced to equal the reset value `RV`. The second output signal is not affected until one time step later, at which time it is equal to the reset value `RV` at the previous time step. The internal **Initial condition** has a direct effect on the second output only when the model starts or when a parent enabled subsystem is reset.

When the enable signal is off, the block is disabled, and the state and output values do not change, except during resets.

The enable signal is on when `E` is not 0, and off when `E` is 0. The enable and reset actions are vectorized and support scalar expansion.

Having two outputs is useful for implementing recursive calculations where the result includes the most recent inputs. The second output can feed back into calculations of the block's inputs without causing an algebraic loop. Meanwhile, the first output shows the most up-to-date calculations.

The **Sample time** parameter specifies the time between samples. To inherit the sample time, set this parameter to `-1`.

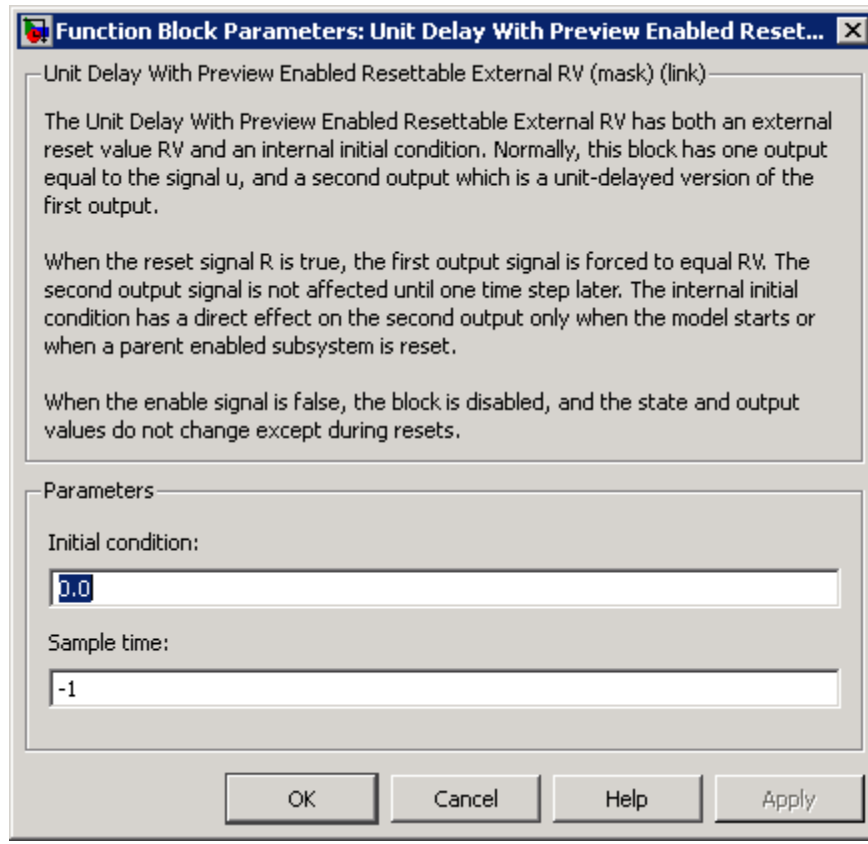
Data Type Support

The Unit Delay With Preview Enabled Resettable External RV block accepts input signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

The outputs are the same data type as the input u.

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 “How to Specify the Sample Time” in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes, to first output port No, to second 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 Resettable, Unit Delay With Preview Resettable External RV

Unit Delay With Preview Resettable

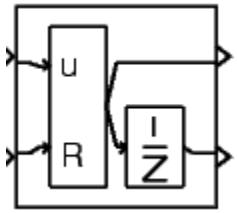
Purpose

Output signal and signal delayed by one sample period, with external 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 a reset signal `R`.

The block has two input ports: one for the input signal `u` and one for the external reset signal `R`.

When the reset signal `R` is false, the first port outputs the signal and the second port outputs the signal delayed by one sample period.

When the reset signal `R` is true, the block resets the current state to the initial condition given by the **Initial condition** parameter. The first output signal is forced to equal the initial condition. The second output signal is not affected until one time step later.

This reset action is vectorized and supports scalar expansion.

Having two outputs is useful for implementing recursive calculations where the result includes the most recent inputs. The second output can feed back into calculations of the block's inputs without causing an algebraic loop. Meanwhile, the first output shows the most up-to-date calculations.

The **Sample time** parameter specifies the time between samples. To inherit the sample time, set this parameter to `-1`.

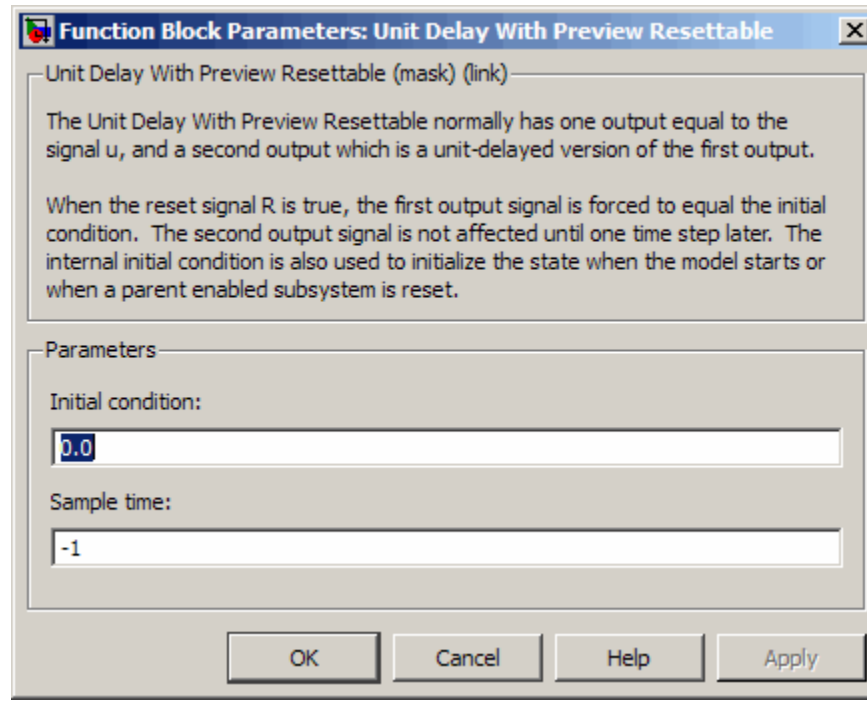
Data Type Support

The Unit Delay With Preview Resettable block accepts input signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

The outputs are the same data type as the input u.

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 "How to Specify the Sample Time" in the online documentation for more information.

Unit Delay With Preview Resettable

Characteristics

Direct Feedthrough	Yes, to first output port No, to second 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

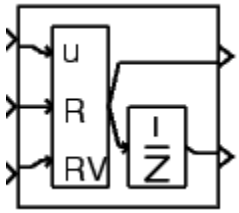
Purpose

Output signal and signal delayed by one sample period, with 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 a reset signal `R`.

The block has three input ports: one for the input signal `u`, one for the external reset signal `R`, and one for the external reset value `RV`.

When the reset signal `R` is false, the first port outputs the signal and the second port outputs the signal delayed by one sample period.

When the reset signal `R` is true, the first output signal is forced to equal the reset value `RV`. The second output signal is not affected until one time step later, at which time it is equal to the reset value `RV` at the previous time step. The internal **Initial condition** has a direct effect on the second output only when the model starts or when a parent enabled subsystem is reset.

This reset action is vectorized and supports scalar expansion.

Having two outputs is useful for implementing recursive calculations where the result includes the most recent inputs. The second output can feed back into calculations of the block's inputs without causing an algebraic loop. Meanwhile, the first output shows the most up-to-date calculations.

The **Sample time** parameter specifies the time between samples. To inherit the sample time, set this parameter to `-1`.

Data Type Support

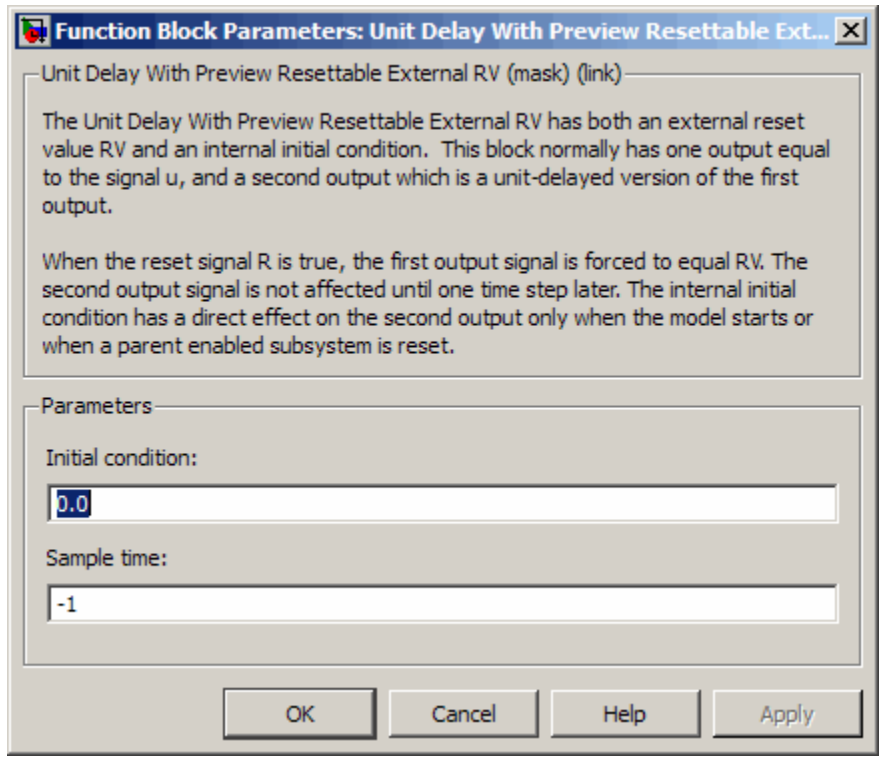
The Unit Delay With Preview Resettable External RV block accepts input signals of the following data types:

- Floating-point
- Built-in integer

- Fixed-point
- Boolean

The outputs are the same data type as the input u.

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 “How to Specify the Sample Time” in the online documentation for more information.

Characteristics

Direct Feedthrough	Yes, to first output port No, to second 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

Variable Time Delay, Variable Transport Delay

Purpose

Delay input by variable amount of time

Library

Continuous

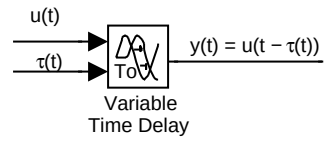
Description

The Variable Transport Delay and Variable Time Delay appear as two blocks in the Simulink block library. However, they are the same Simulink block with different settings of a **Select delay type** parameter. Use this parameter to specify the mode in which the block operates.

Variable Time Delay

In this mode, the block has a data input, a time delay input, and a data output. (See "How to Rotate a Block" in the Simulink documentation for a description of the port order for various block orientations.) The 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 time delay input.

$$y(t) = u(t - t_0) = u(t - \tau(t))$$



During the simulation, the block stores time and input value pairs in an internal buffer. At the start of 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 you want the output at a time that does not correspond to 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 the input at 4.5 from the input at 4 and uses the extrapolated value as its output at t = 5.

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

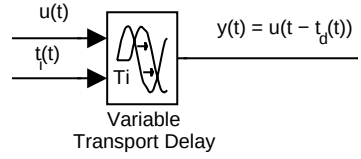
In this mode, the block output at the current time step is equal to the value of its data (top, or left) input at an earlier time equal to the current time minus a transportation delay.

$$y(t) = u(t - t_d(t))$$

Simulink software 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 time delay (bottom, or right) 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.

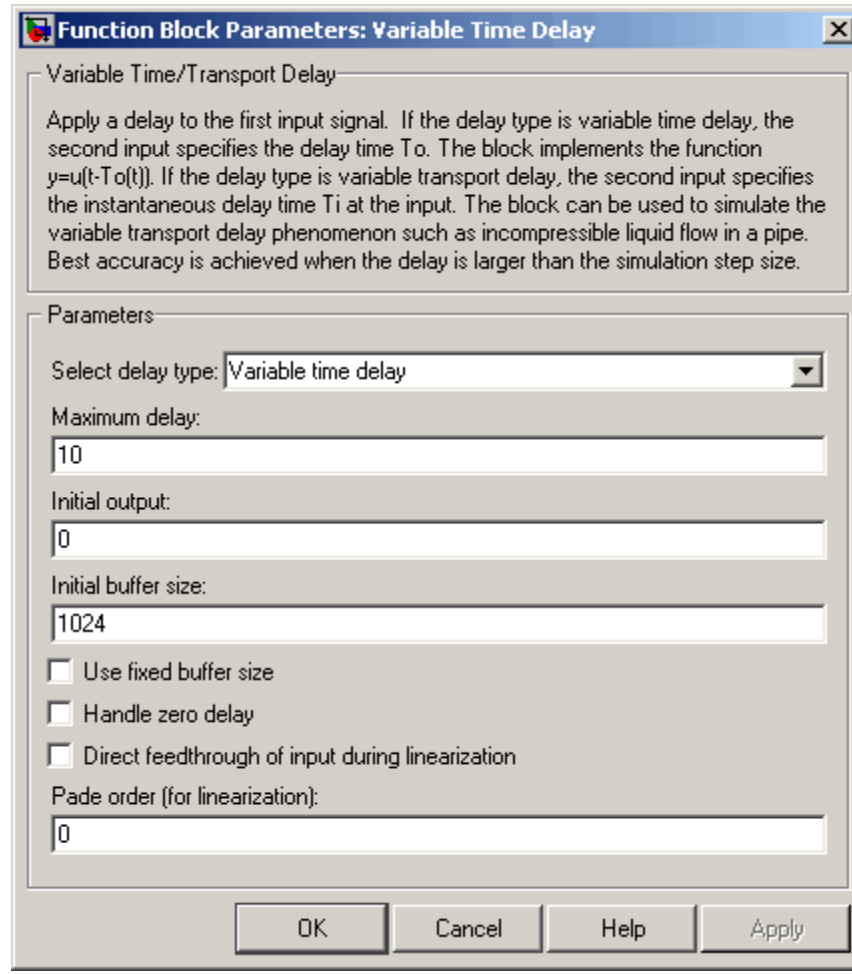
Data Type Support

The Variable Time Delay and Variable Transport Delay blocks accept and output real signals of type `double`.

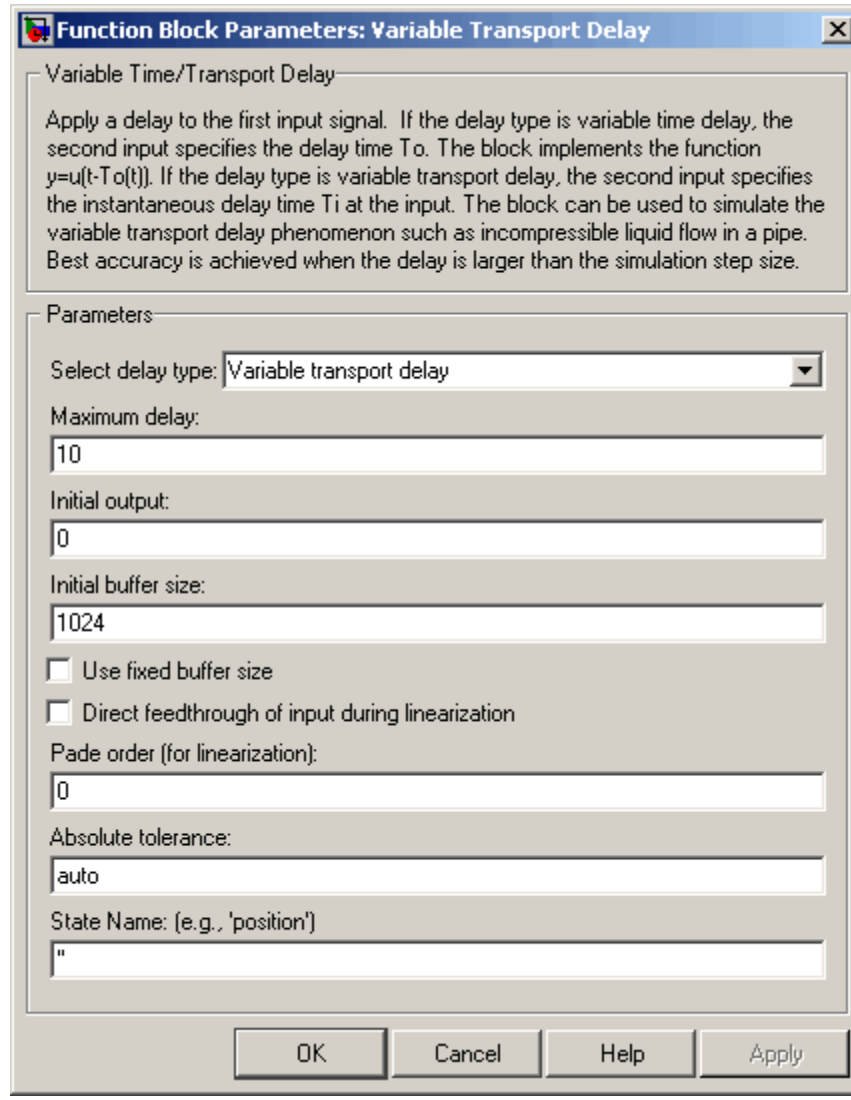
Parameters and Dialog Box

The parameters and dialog box differ, based on the mode in which the block is operating: variable time or variable transport. Most parameters exist in both modes.

The dialog box for the Variable Time Delay block appears as follows.



The dialog box for the Variable Transport Delay block appears as follows.


Function Block Parameters: Variable Transport Delay
Variable Time/Transport Delay
Apply a delay to the first input signal. If the delay type is variable time delay, the second input specifies the delay time To. The block implements the function y=u(t-To(t)). If the delay type is variable transport delay, the second input specifies the instantaneous delay time Ti at the input. The block can be used to simulate the variable transport delay phenomenon such as incompressible liquid flow in a pipe. Best accuracy is achieved when the delay is larger than the simulation step size.
Parameters
Select delay type: Variable transport delay
Maximum delay:
10
Initial output:
0
Initial buffer size:
1024
Use fixed buffer size
Direct feedthrough of input during linearization
Pade order (for linearization):
0
Absolute tolerance:
auto
State Name: (e.g., 'position')
''
OK
Cancel
Help
Apply

Select delay type

Specify the mode in which the block operates.

Settings

Default: The Variable Time Delay block has a default value of `Variable time delay`. The Variable Transport Delay block has a default value of `Variable transport delay`.

`Variable time delay`
: Specifies a Variable Time Delay block.

`Variable transport delay`
: Specifies a Variable Transport Delay block.

Dependencies

Setting this parameter to `Variable time delay` enables the **Handle zero delay** parameter.

Setting this parameter to `Variable transport delay` enables the **Absolute tolerance** and **State Name** parameters.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Maximum delay

Set the maximum value of the time delay input.

Settings

Default: 10

- This value defines the largest time delay input that this block allows. The block clips any delay that exceeds this value.
- This value cannot be negative. If the time delay becomes negative, the block clips it to zero and issues a warning message.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Initial output

Specify the output that the block generates until the simulation time first exceeds the time delay input.

Settings

Default: `0`

The initial output of this block cannot be `inf` or `NaN`.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Initial buffer size

Define the initial memory allocation for the number of input points to store.

Settings

Default: `1024`

- If the number of input points exceeds the initial buffer size, the block allocates additional memory.
- After simulation ends, a message shows the total buffer size needed.

Tips

- Because allocating memory slows down simulation, choose this 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.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Use fixed buffer size

Specify use of a fixed-size buffer to save input data from previous time steps.

Settings

Default: Off

 On

The block uses a fixed-size buffer.

 Off

The block does not use a fixed-size buffer.

The **Initial buffer size** parameter specifies the buffer's size. If the buffer is full, new data replaces data already in the buffer. Simulink software uses linear extrapolation to estimate output values that are not in the buffer.

Note ERT or GRT code generation uses a fixed-size buffer even if you do not select this check box.

Tips

- If the input data is linear, selecting this check box can save memory.
- If the input data is nonlinear, do not select this check box. Doing so might yield inaccurate results.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Handle zero delay

Convert this block to a direct feedthrough block.

Settings

Default: Off

 On

The block uses direct feedthrough.

 Off

The block does not use direct feedthrough.

Dependency

Setting **Select delay type** to `Variable time delay` enables this parameter.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Direct feedthrough of input during linearization

Cause the block to output its input during linearization and trim, which sets the block mode to direct feedthrough.

Settings

Default: Off

 On

Enables direct feedthrough of input.

 Off

Disables direct feedthrough of input.

Tips

- Selecting this check box can cause a change in the ordering of states in the model when you use the functions `linmod`, `dlinmod`, or `trim`. To extract this new state ordering:

  1. Compile the model using the following command, where `model` is the name of the Simulink model.

     ```
     [sizes, x0, x_str] = model([],[],[],'lincompile');
     ```

  2. 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 you pass a vector of states as input to the `linmod`, `dlinmod`, or `trim` functions, the state vector must use this new state ordering.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Pade order (for linearization)

Set the order of the Pade approximation for linearization routines.

Settings

Default: `0`

- 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.

Command-Line Information

See “Block-Specific Parameters” on page 8-100 for the command-line information.

Absolute tolerance

Specify the absolute tolerance for computing the block output.

Settings

Default: `auto`

- You can enter `auto` or a numeric value.
- If you enter `auto`, Simulink uses the absolute tolerance value in the Configuration Parameters dialog box (see "Solver Pane") to compute the block output.
- If you enter a numeric value, that value overrides the absolute tolerance in the Configuration Parameters dialog box.

Dependency

Setting **Select delay type** to `Variable transport delay` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

State Name (e.g., 'position')

Assign a unique name to each state.

Settings

Default: `' '`

If this field is blank, no name assignment occurs.

Tips

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'`.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, `{'a', 'b', 'c'}`. Each name must be unique.
- The state names apply only to the selected block.

- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

  For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.
- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell array, or structure.

Dependency

Setting **Select delay type** to `Variable transport delay` enables this parameter.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Direct Feedthrough	Yes, of the time delay (second) input
Sample Time	Continuous
Scalar Expansion	Yes, of input and all parameters except **Initial buffer size**
Dimensionalized	Yes
Zero-Crossing Detection	No

See Also

Transport Delay

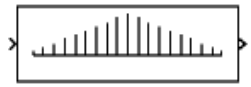
Purpose

Implement weighted moving average (obsolete)

Library

Discrete (until R2007b)

Description



Note The Weighted Moving Average block is obsolete. This block was removed from the Discrete library in R2008a and replaced with the Discrete FIR Filter block. However, existing models that contain the Weighted Moving Average block continue to work for backward compatibility.

Use the Discrete FIR Filter block in new models. Consider using the `slupdate` function to replace Weighted Moving Average with Discrete FIR Filter in existing models.

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. You can choose whether or not to specify the data type and scaling of the weights in the dialog with the **Gain data type** parameter.

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.

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.

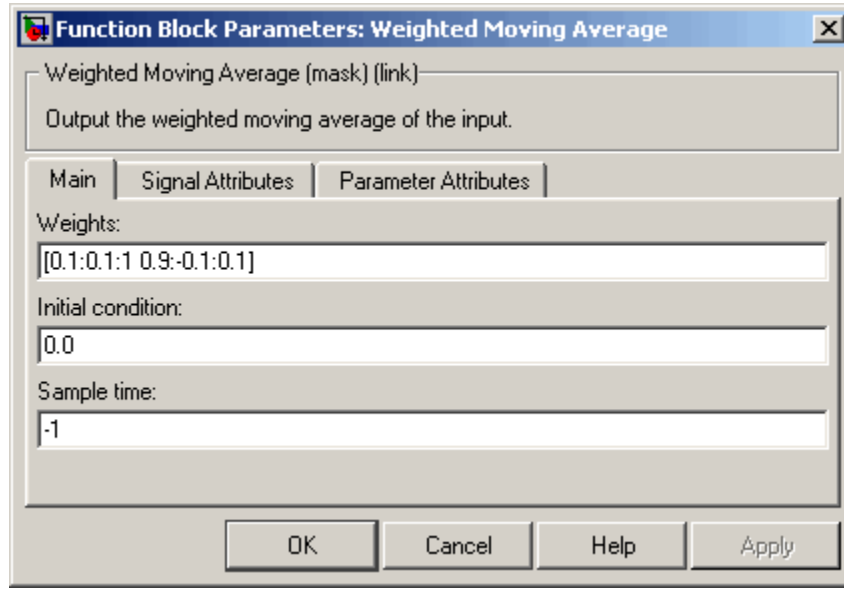
Weighted Moving Average (Obsolete)

Data Type Support

The Weighted Moving Average block supports all numeric data types supported by Simulink software, including fixed-point data types.

Parameters and Dialog Box

The **Main** pane of the Weighted Moving Average block dialog box 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.

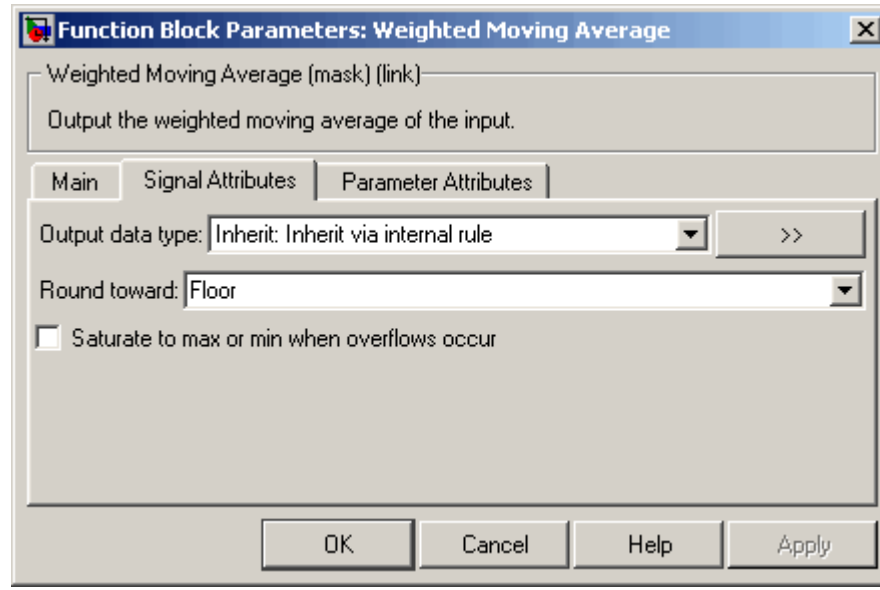
Initial condition

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 "How to Specify the Sample Time" in the online documentation for more information.

The **Signal Attributes** pane of the Weighted Moving Average block dialog box appears as follows:



Output data type

Specify the output data type. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via back propagation`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `float('single')`

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the **Output data type** parameter.

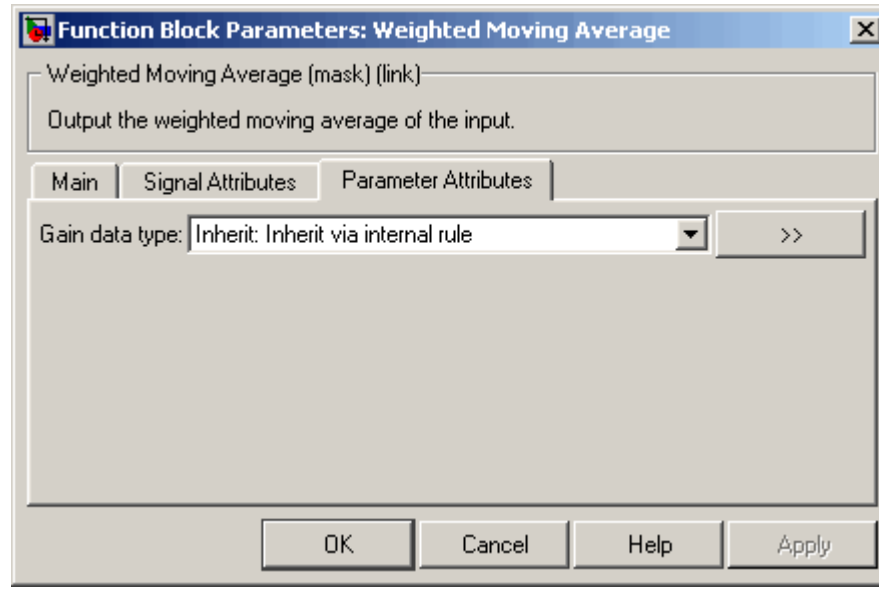
See "Specifying Block Output Data Types" in *Simulink User's Guide* for more information.

Lock output scaling against changes by the autoscaling tool
Select to lock scaling of outputs. This parameter is visible only if you enter an expression for the **Output data type** parameter.

Round toward
Rounding mode for the fixed-point output. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

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

The **Parameter Attributes** pane of the Weighted Moving Average block dialog appears as follows:



Gain data type

Specify the data type of the **Weights** parameter. You can set it to:

- A rule that inherits a data type, for example, `Inherit: Inherit via internal rule`
- The name of a data type object, for example, a `Simulink.NumericType` object
- An expression that evaluates to a data type, for example, `fixdt(1,16,0)`

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Gain data type** parameter. (See "Using the Data Type Assistant" in *Simulink User's Guide*.)

Examples

Suppose you want to configure this block for two outputs (SIMO mode) where the first output is given by

$$y_1(k) = a_1 \cdot u(k) + b_1 \cdot u(k-1) + c_1 \cdot u(k-2)$$

the second output is given by

$$y_2(k) = a_2 \cdot u(k) + b_2 \cdot u(k-1)$$

and the initial values of *u*(*k - 1*) and *u*(*k - 2*) are given by `ic1` and `ic2`, respectively. To configure the Weighted Moving Average block for this situation, you must specify the **Weights** parameter as `[a1 b1 c1; a2 b2 c2]` where `c2` = 0, and the **Initial condition** parameter as `[ic1 ic2]`.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

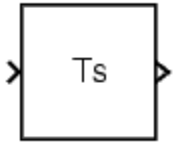
Purpose

Support calculations involving sample time

Library

Signal Attributes

Description



The Weighted Sample Time block is an implementation of the Weighted Sample Time Math block. See Weighted Sample Time Math for more information.

Weighted Sample Time Math

Purpose

Support calculations involving sample time

Library

Math Operations

Description

The Weighted Sample Time Math block adds, subtracts, multiplies, or divides its input signal, `u`, by a weighted sample time, `Ts`. If the input signal is continuous, `Ts` is the sample time of the Simulink model. Otherwise, `Ts` is the sample time of the discrete input signal.

You specify the math operation with the **Operation** parameter. Additionally, you can specify to use only the weight with either the sample time or its inverse.

Enter the weighting factor in the **Weight value** parameter. If the weight, `w`, is `1`, that value does not appear in the equation on the block icon.

The block computes its output using the precedence rules for MATLAB operators (see "Operator Precedence" in the MATLAB documentation). For example, if the **Operation** parameter specifies `+`, the block calculates output using this equation:

```
u + (Ts * w)
```

However, if the **Operation** parameter specifies `/`, the block calculates output using this equation:

```
(u / Ts) / w
```

Data Type Support

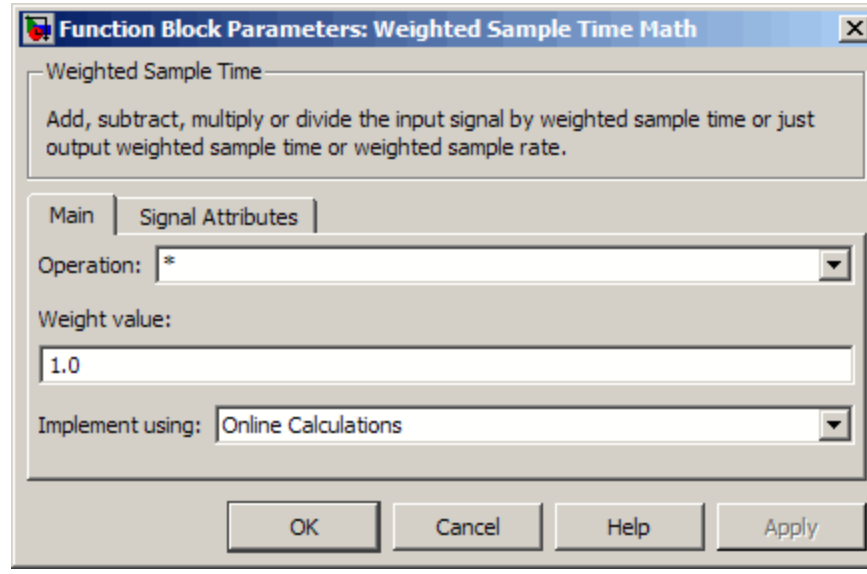
The Weighted Sample Time Math block accepts signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

For more information, see "Data Types Supported by Simulink" in the Simulink documentation.

Parameters and Dialog Box

The **Main** pane of the Weighted Sample Time Math block dialog box appears as follows:



Operation

Specify operation to use: `+`, `-`, `*`, `/`, `Ts Only`, or `1/Ts Only`.

Weight value

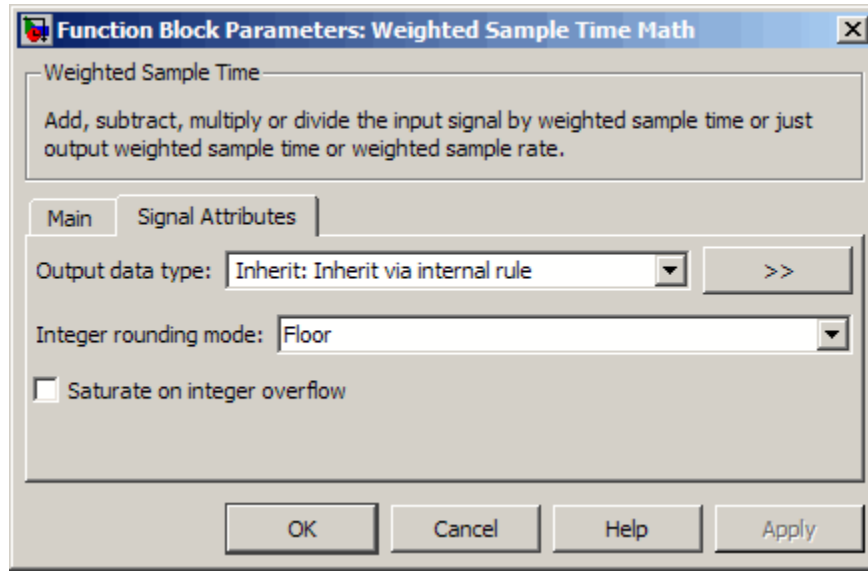
Enter weight of sample time.

Implement using

Specify online calculations or offline scaling adjustment. This parameter is visible only if you specify `*` or `/` as the **Operation** parameter.

Note When the **Implement using** parameter is not visible, operations default to online calculations.

The **Signal Attributes** pane of the Weighted Sample Time Math block dialog box appears as follows:



Output data type

Specify whether the block inherits the output data type by an internal rule or back propagation.

Tip If you enter an expression in the edit field, the expression must evaluate to one of the two inherit rules.

Integer rounding mode

Select the rounding mode for fixed-point operations. For more information, see "Rounding" in the *Simulink Fixed Point User's Guide*.

Saturate on integer overflow

Select to have fixed-point overflows saturate. Otherwise, they wrap.

When you select this check box, saturation applies to every internal operation on the block, not just the output or result. In general, the code generation process can detect when overflow is not possible, in which case, no saturation code is necessary.

This parameter is visible only if:

- The **Operation** parameter specifies `+` or `-`.
- The **Operation** parameter specifies `*` or `/` and the **Implement using** parameter specifies `Online Calculations`.

Characteristics

Direct Feedthrough	For all math operations except `Ts` and `1/Ts`
Scalar Expansion	No, the weight is always a scalar
Zero-Crossing Detection	No

While Iterator

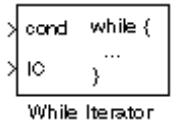
Purpose

Repeatedly execute contents of subsystem at current time step while condition is satisfied

Library

Ports & Subsystems / While Iterator Subsystem

Description



While Iterator

The While Iterator block, when placed in a subsystem, repeatedly executes the contents of the subsystem at the current time step while a specified condition is true.

Note Placing a While Iterator block in a subsystem makes it an atomic subsystem if it is not already an atomic subsystem.

The output of a While Iterator subsystem can not be a function-call signal. Simulink software will display an error message if the simulation is run or the diagram updated.

You can use this block to implement the block-diagram equivalent of a C program `while` or `do-while` loop. In particular, the block's **While loop style** parameter allows you to choose either of the following while loop modes:

- `do-while`

  In this mode, the While Iterator block has one input, the while condition input, whose source must reside in the subsystem. At each time step, the block runs all the blocks in the subsystem once and then checks whether the while condition input is true. If the input is true, the iterator block runs the blocks in the subsystem again. This process continues as long as the while condition input is true and the number of iterations is less than or equal to the iterator block's **Maximum number of iterations** parameter.

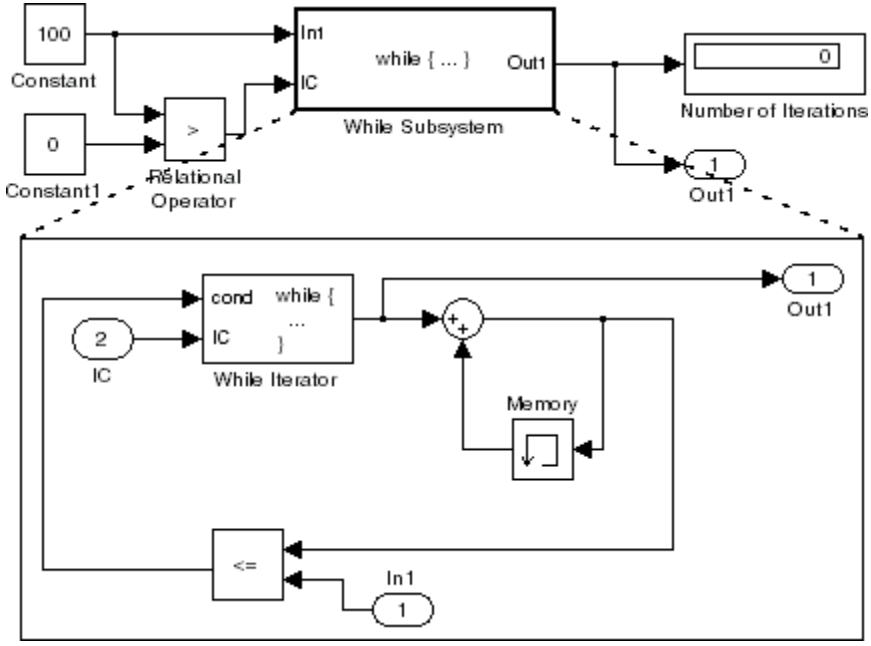
- `while`

  In this mode, the iterator block has two inputs: a while condition input and an initial condition (IC) input. The source of the initial condition signal must be external to the while subsystem. At the

beginning of the time step, if the IC input is true, the iterator block executes the contents of the subsystem and then checks the while condition input. If the while condition input is true, the iterator executes the subsystem again. This process continues as long as the while condition input is true and the number of iterations is less than or equal to the iterator block's **Maximum number of iterations** parameter. If the IC input is false at the beginning of a time step, the iterator does not execute the contents of the subsystem during the time step.

Note Unless you are certain that the while condition will become false at some point in the simulation, you should specify a maximum number of iterations to avoid endless loops, which can be broken only by terminating MATLAB.

The While Iterator block can optionally output the current iteration number, starting at 1. The following example uses this capability to compute N, where N is the first N integers whose sum is less than 100.



This example is the diagrammatic equivalent to the following pseudocode.

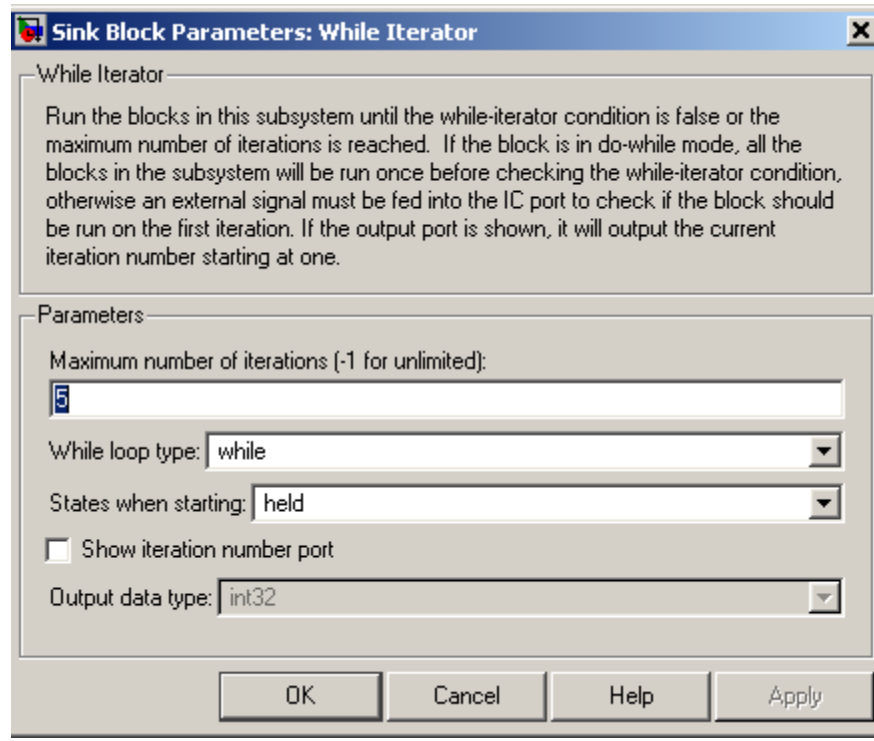
```
max_sum = 100;
sum = 0;
iteration_number = 0;
cond = (max_sum > 0);
while (cond != 0) {
 iteration_number = iteration_number + 1;
 sum = sum + iteration_number;
 if (sum > max_sum OR iteration_number > max_iterations)
  cond = 0;
}
```

Data Type Support

Acceptable data inputs for the condition ports are any numeric data type supported by Simulink software, as well as any fixed-point type, that include a `0` value. For a discussion on the data types supported by Simulink software, see "Data Types Supported by Simulink" in the Simulink documentation.

The While Iterator block's optional output port can output any of the following data types: `double`, `int32`, `int16`, or `int8`.

Parameters and Dialog Box



Maximum number of iterations

The maximum number of iterations allowed. A value of `-1` allows any number of iterations as long as the while condition input is true. Note that if you specify -1 and the while condition never

becomes false, the simulation will run forever. In this case, the only way to stop the simulation is to terminate the MATLAB process. Therefore, you should not specify -1 as the value of this parameter unless you are certain that the while condition will become false at some point in the simulation.

While loop style

Specifies the type of while loop implemented by this block. See the preceding block description for more information.

States when starting

Set this field to `reset` if you want the iterator block to reset the states of the blocks in the while subsystem to their initial values at the beginning of each time step (i.e., before executing the first loop iteration in the current time step). To cause the states of blocks in the subsystem to persist across time steps, set this field to `held` (the default).

Show iteration number port

If you select this check box, the While Iterator block outputs its iteration value. This value starts at `1` and is incremented by 1 for each succeeding iteration. By default, this check box is not selected.

Output data type

If you select the **Show iteration number port** check box (the default), this field is enabled. Use it to set the data type of the iteration number output to `int32`, `int16`, `int8`, or `double`.

Characteristics

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

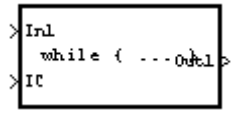
Purpose

Represent subsystem that executes repeatedly while condition is satisfied during simulation time step

Library

Ports & Subsystems

Description



The While Iterator Subsystem block is a Subsystem block that is preconfigured to serve as a starting point for creating a subsystem that executes repeatedly while a condition is satisfied during a simulation time step.

See the While Iterator block and "Modeling Control Flow Logic" for more information.

When using simplified initialization mode, you cannot place any block needing elapsed time within an Iterator Subsystem. In simplified initialization mode, Iterator subsystems do not maintain elapsed time, so Simulink will report an error if any such block (such as the Discrete-Time Integrator block) is placed within the subsystem. For more information on simplified initialization modes, see "Underspecified initialization detection".

Width

Purpose

Output width of input vector

Library

Signal Attributes

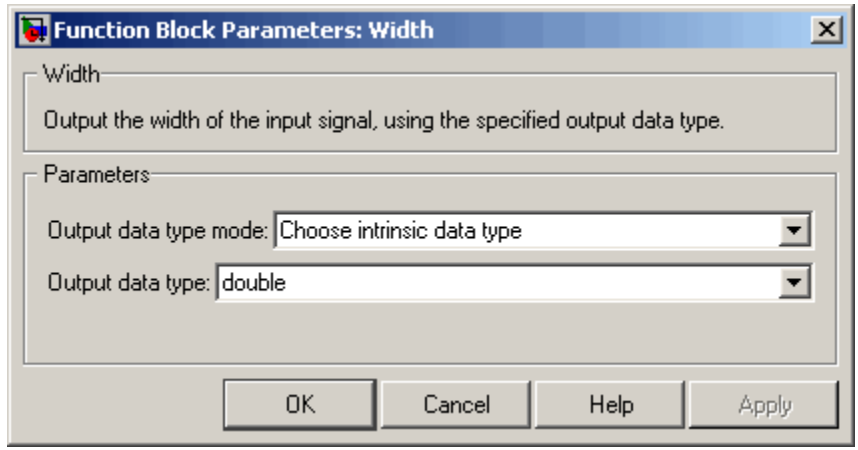
Description

The Width block generates as output the width of its input vector.

Data Type Support

The Width block accepts real or complex signals of any data type supported by Simulink software, including fixed-point and enumerated data types. The Width block supports mixed-type signal vectors.

Parameters and Dialog Box



Note The Width block ignores the **Data type override** setting of the Fixed-Point Tool.

Output data type mode

Specify the output data type to be the same as the input, or inherit the data type by back propagation. You can also choose to specify a built-in data type from the drop-down list in the **Output data type** parameter.

Output data type

This parameter is visible when `Choose intrinsic data type` is selected from the **Output data type mode** parameter. Choose a built in data type from the drop down list.

Characteristics

Sample Time	Constant
Dimensionalized	Yes
Multidimensionalized	Yes

Wrap To Zero

Purpose

Set output to zero if input is above threshold

Library

Discontinuities

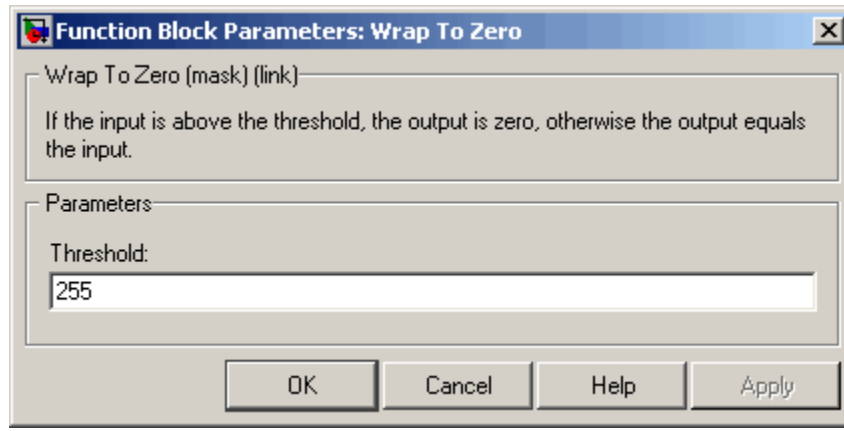
Description

The Wrap To Zero block sets the output to zero if the input is above the value set by the **Threshold** parameter, and outputs the input if the input is less than or equal to the **Threshold**.

Data Type Support

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

Parameters and Dialog Box



Threshold

When the input exceeds the threshold, the output is set to zero.

Characteristics

Direct Feedthrough	Yes
Scalar Expansion	Yes
Multidimensionalized	Yes

Purpose

Display X-Y plot of signals using 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* direction) against data in the second input (the *y* direction). (See "How to Rotate a Block" in *Simulink User's Guide* for a description of the port order for various block orientations.) This block is useful for examining limit cycles and other two-state data. Data outside the specified range is not displayed.

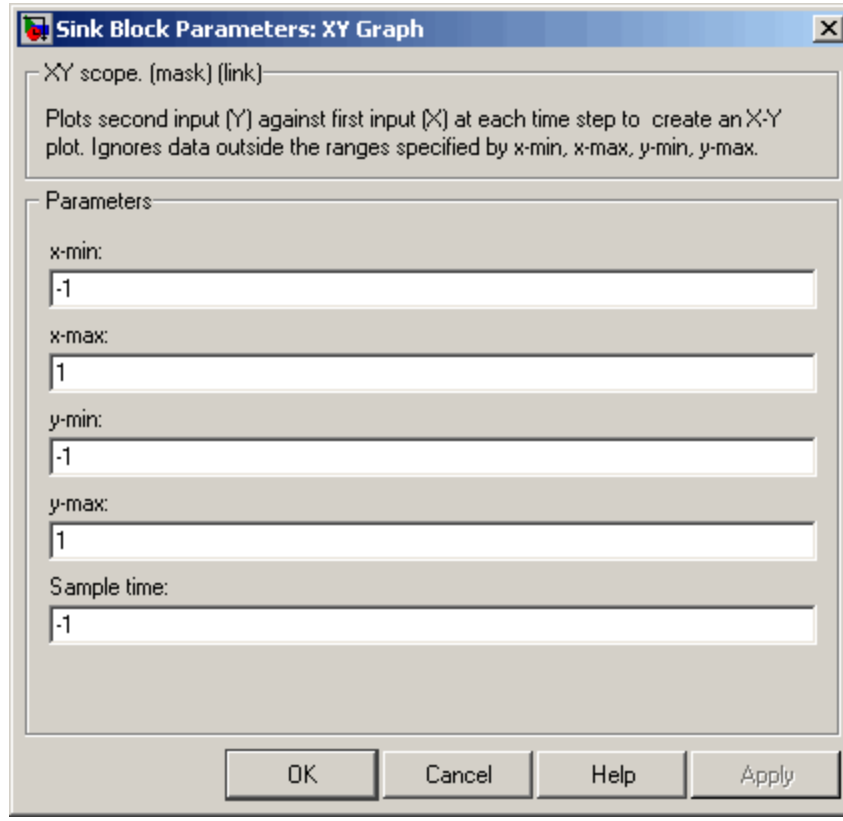
Simulink software opens a figure window for each XY Graph block in the model at the start of the simulation.

Data Type Support

The XY Graph block accepts real signals of the following data types:

- Floating-point
- Built-in integer
- Fixed-point
- Boolean

Parameters and Dialog Box



x-min
The minimum *x*-axis value. The default is `-1`.

x-max
The maximum *x*-axis value. The default is `1`.

y-min
The minimum *y*-axis value. The default is `-1`.

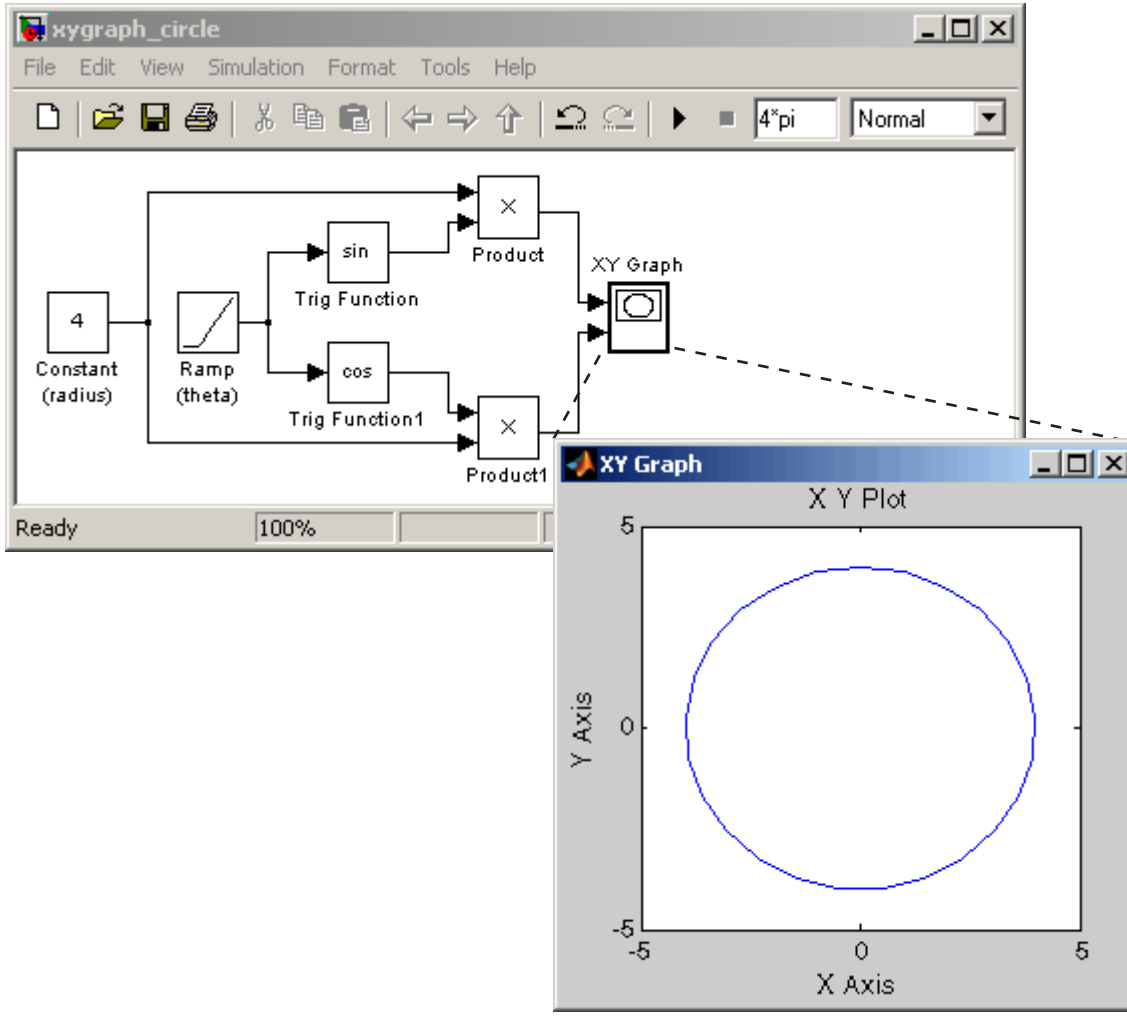
y-max
The maximum *y*-axis value. The default is `1`.

Sample time

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

Examples

The following model computes the points that define a circle of radius 4, centered at the origin of the *x*-*y* plane. The XY Graph block displays the circle.


xygraph_circle
File Edit View Simulation Format Tools Help
4*pi
Normal
4
Constant
(radius)
Ramp
(theta)
sin
Trig Function
cos
Trig Function1
Product
Product1
XY Graph
Ready
100%
XY Graph
X Y Plot
Y Axis
X Axis
5
0
-5
-5
0
5

Characteristics

Sample Time	Specified in the **Sample time** parameter
States	0

Zero-Order Hold

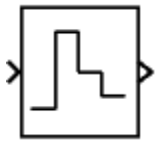
Purpose

Implement zero-order hold of one sample period

Library

Discrete

Description



The Zero-Order Hold block samples and holds the input for the sample period you specify. The block accepts one input and generates one output. Each signal can be scalar or vector. If the input is a vector, the block holds all elements of the vector for the same sample period.

You specify the time between samples with the **Sample time** parameter. A setting of `-1` means the block inherits the **Sample time**.

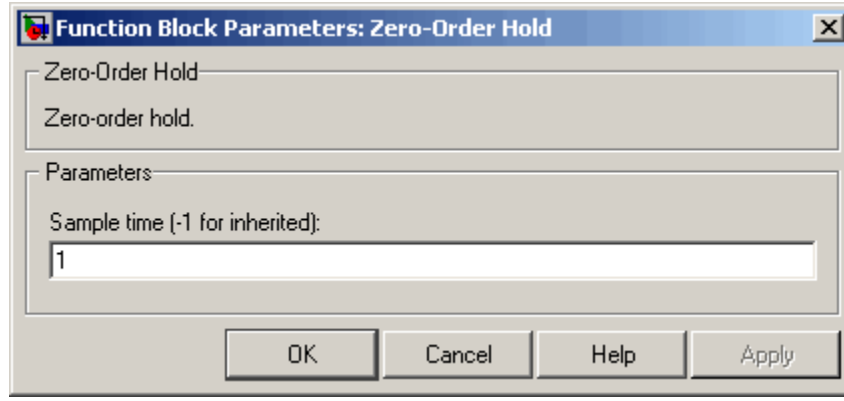
Tip Do not use the Zero-Order Hold block to create a fast-to-slow transition between blocks operating at different sample rates. Instead, use the Rate Transition block.

Data Type Support

The Zero-Order Hold block accepts real or complex signals of any data type supported by Simulink software, including fixed-point and enumerated data types.

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

Parameters and Dialog Box



Sample time (-1 for inherited)

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

Bus Support

The Zero-Order Hold block is a bus-capable block. The input can be a virtual or nonvirtual bus signal. No block-specific restrictions exist. All signals in a nonvirtual bus input to a Zero-Order Hold block must have the same sample time, even if the elements of the associated bus object specify inherited sample times. You can use a Rate Transition block to change the sample time of an individual signal, or of all signals in a bus. See "Using Composite Signals" and "Bus-Capable Blocks" in the *Simulink User's Guide* for more information.

Characteristics

Bus-capable	Yes
Direct Feedthrough	Yes
Sample Time	Specified in the **Sample time** parameter
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

Zero-Pole

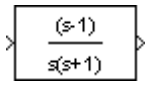
Purpose

Model system by zero-pole-gain transfer function

Library

Continuous

Description



The Zero-Pole block models a system that you define with the zeros, poles, and gain of a Laplace-domain transfer function. This block can model single-input single output (SISO) and single-input multiple-output (SIMO) systems.

Conditions for Using This Block

The Zero-Pole block assumes the following conditions:

- The transfer function has the form

$$H(s) = K\frac{Z(s)}{P(x)} = K\frac{(s-Z(1))(s-Z(2))\ldots(s-Z(m))}{(s-P(1))(s-P(2))\ldots(s-P(n))}$$

  where Z represents the zeros, P the poles, and K the gain of the transfer function.

- The number of poles must be greater than or equal to the number of zeros.
- If the poles and zeros are complex, they must be complex-conjugate pairs.
- For a multiple-output system, all transfer functions must have the same poles. The zeros can differ in value, but the number of zeros for each transfer function must be the same.

Note You cannot use a Zero-Pole block to model a multiple-output system when the transfer functions have a differing number of zeros or a single zero each. Use multiple Zero-Pole blocks to model such systems.

Modeling a Single-Output System

For a single-output system, the input and the output of the block are scalar time-domain signals. To model this system:

1. Enter a vector for the zeros of the transfer function in the **Zeros** field.
2. Enter a vector for the poles of the transfer function in the **Poles** field.
3. Enter a 1-by-1 vector for the gain of the transfer function in the **Gain** field.

Modeling a Multiple-Output System

For a multiple-output system, the block input is a scalar and the output is a vector, where each element is an output of the system. To model this system:

1. Enter a matrix of zeros in the **Zeros** field.

   Each *column* of this matrix contains the zeros of a transfer function that relates the system input to one of the outputs.

2. Enter a vector for the poles common to all transfer functions of the system in the **Poles** field.
3. Enter a vector of gains in the **Gain** field.

   Each element is the gain of the corresponding transfer function in **Zeros**.

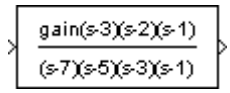
Each element of the output vector corresponds to a column in **Zeros**.

Transfer Function Display on the Block

The Zero-Pole block displays the transfer function depending on how you specify the zero, pole, and gain parameters.

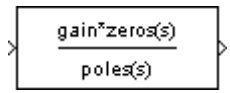
- If you specify each parameter as an expression or a vector, the block shows the transfer function with the specified zeros, poles, and gain. If you specify a variable in parentheses, the block evaluates the variable.

For example, if you specify **Zeros** as `[3,2,1]`, **Poles** as `(poles)`, where `poles` is `[7,5,3,1]`, and **Gain** as `gain`, the block looks like this:



- If you specify each parameter as a variable, the block shows the variable name followed by `(s)` if appropriate.

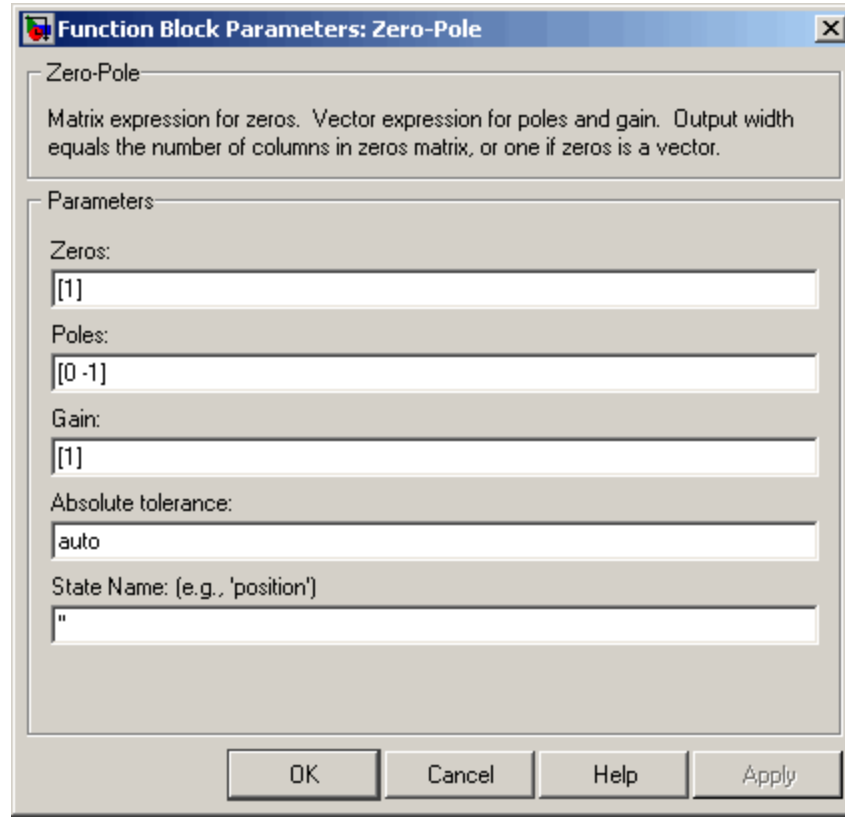
  For example, if you specify **Zeros** as `zeros`, **Poles** as `poles`, and **Gain** as `gain`, the block looks like this:



Data Type Support

The Zero-Pole block accepts real signals of type `double`.

Parameters and Dialog Box



Zeros

Define the matrix of zeros.

Settings

Default: `[1]`

Tips

- For a single-output system, enter a vector for the zeros of the transfer function.
- For a multiple-output system, enter a matrix. Each *column* of this matrix contains the zeros of a transfer function that relates the system input to one of the outputs.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Poles

Define the vector of poles.

Settings

Default: `[0 -1]`

Tips

- For a single-output system, enter a vector for the poles of the transfer function.
- For a multiple-output system, enter a vector for the poles common to all transfer functions of the system.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Gain

Define the vector of gains.

Settings

Default: `[1]`

Tips

- For a single-output system, enter a 1-by-1 vector for the gain of the transfer function.
- For a multiple-output system, enter a vector of gains. Each element is the gain of the corresponding transfer function in **Zeros**.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Absolute tolerance

Specify the absolute tolerance for computing the block output.

Settings

Default: `auto`

- You can enter `auto` or a numeric value.
- If you enter `auto`, Simulink uses the absolute tolerance value in the Configuration Parameters dialog box (see "Solver Pane") to compute the block output.
- If you enter a numeric value, that value overrides the absolute tolerance in the Configuration Parameters dialog box.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

State Name (e.g., 'position')

Assign a unique name to each state.

Settings

Default: `' '`

If this field is blank, no name assignment occurs.

Tips

- To assign a name to a single state, enter the name between quotes, for example, `'velocity'`.
- To assign names to multiple states, enter a comma-delimited list surrounded by braces, for example, `{'a', 'b', 'c'}`. Each name must be unique.
- The state names apply only to the selected block.
- The number of states must divide evenly among the number of state names.
- You can specify fewer names than states, but you cannot specify more names than states.

  For example, you can specify two names in a system with four states. The first name applies to the first two states and the second name to the last two states.
- To assign state names with a variable in the MATLAB workspace, enter the variable without quotes. A variable can be a string, cell array, or structure.

Command-Line Information

See "Block-Specific Parameters" on page 8-100 for the command-line information.

Characteristics

Direct Feedthrough	Only if the lengths of the **Poles** and **Zeros** parameters are equal
Sample Time	Continuous

Scalar Expansion	No
States	Length of the **Poles** vector
Dimensionalized	No
Zero-Crossing Detection	No

See Also

Discrete Zero-Pole

3

Function Reference

Model Construction

`addterms`	Add terminators to unconnected ports in model
`add_block`	Add block to model
`add_line`	Add line to Simulink system
`add_param`	Add parameter to Simulink system
`attachConfigSet`	Associate configuration set or configuration reference with model
`attachConfigSetCopy`	Copy configuration set or configuration reference and associate it with model
`bdclose`	Close any or all Simulink system windows unconditionally
`bdIsLoaded`	Whether block diagram is in memory
`bdroot`	Return name of top-level Simulink system
`closeDialog`	Close configuration parameters dialog
`close_system`	Close Simulink system window or block dialog box
`delete_block`	Delete block from Simulink system
`delete_line`	Delete line from Simulink system
`delete_param`	Delete system parameter added via `add_param` command
`detachConfigSet`	Dissociate configuration set or configuration reference from model
`disableimplicitsignalresolution`	Convert model to use only explicit signal resolution
`docblock`	Get or set editor invoked by Simulink DocBlock

`find_mdlrefs`	Find Model blocks and referenced models at all levels or at top level only
`find_system`	Find systems, blocks, lines, ports, and annotations
`gcb`	Get pathname of current block
`gcbh`	Get handle of current block
`gcs`	Get pathname of current system
`getActiveConfigSet`	Get model's active configuration set or configuration reference
`getCallbackAnnotation`	Get information about annotation
`getConfigSet`	Get one of model's configuration sets or configuration references
`getConfigSets`	Get names of all of model's configuration sets or configuration references
`getfullname`	Get pathname of block or line
`get_param`	Get system and block parameter values
`legacy_code`	Use Legacy Code Tool
`libinfo`	Get information about library blocks referenced by model
`load_system`	Invisibly load Simulink model
`modeladvisor`	Open Model Advisor
`new_system`	Create empty Simulink system
`openDialog`	Open configuration parameters dialog
`open_system`	Open Simulink system window or block dialog box
`replace_block`	Replace blocks in Simulink model
`save_system`	Save Simulink system

`setActiveConfigSet`	Specify model's active configuration set or configuration reference
`set_param`	Set Simulink system and block parameter values
`signalbuilder`	Create and access Signal Builder blocks
`simulink`	Open Simulink block library
`Simulink.BlockDiagram.addBusToVector`	Add Bus to Vector blocks to convert virtual bus signals into vector signals
`Simulink.BlockDiagram.copyContentsToSubSystem`	Copy contents of block diagram to empty subsystem
`Simulink.BlockDiagram.createSubSystem`	Create a subsystem containing a specified set of blocks
`Simulink.BlockDiagram.deleteContents`	Delete contents of block diagram
`Simulink.Bus.cellToObject`	Convert cell array containing bus information to bus objects
`Simulink.Bus.createObject`	Create bus objects for blocks
`Simulink.Bus.objectToCell`	Convert bus objects to cell array containing bus information
`Simulink.Bus.save`	Save bus objects in M-file
`Simulink.ModelReference.protect`	Obscure referenced model contents to hide intellectual property embodied by the model
`Simulink.SubSystem.convertToModelReference`	Convert atomic subsystem or function call subsystem to model reference
`Simulink.SubSystem.copyContentsToBlockDiagram`	Copy contents of subsystem to empty block diagram
`Simulink.SubSystem.deleteContents`	Delete contents of subsystem
`slCharacterEncoding`	Change MATLAB character set encoding

`sldiscmdl`	Discretize model that contains continuous blocks
`slIsFileChangedOnDisk`	Determine whether model has changed since it was loaded
`slmdldiscui`	Open Model Discretizer GUI
`slreplace_mux`	Replace Mux blocks used to create buses with Bus Creator blocks
`slupdate`	Replace blocks from previous releases with latest versions
`view_mdlrefs`	Display graph of model reference dependencies

Simulation

`add_exec_event_listener`	Register listener for block method execution event
`model`	Execute particular phase of simulation of model
`sim`	Simulate dynamic system
`simplot`	Plot simulation data in figure window
`Simulink.Block.getSampleTimes`	Return sample time information for a block
`Simulink.BlockDiagram.get-InitialState`	Return initial state structure of block diagram
`Simulink.BlockDiagram.get-SampleTimes`	Return all sample times associated with a model
`Simulink.BlockDiagram.getChecksum`	Return checksum of model
`Simulink.SubSystem.getChecksum`	Return checksum of subsystem
`slbuild`	Build standalone and model reference targets
`sldebug`	Start simulation in debug mode
`sldiagnostics`	Display diagnostic information about Simulink system
`unpack`	Extract signal logging objects from signal logs and write them into MATLAB workspace
`who`	List names of top-level data logging objects in Simulink data log
`whos`	List names and types of top-level data logging objects in Simulink data log

Linearization and Trimming

`linmod, dlinmod, linmod2, linmodv5`	Extract continuous- or discrete-time linear state-space model of system around operating point
`trim`	Find trim point of dynamic system

Data Type

`fixdt`	Create `Simulink.NumericType` object describing fixed-point or floating-point data type
`fixptbestexp`	Exponent that gives best precision for fixed-point representation of value
`fixptbestprec`	Determine maximum precision available for fixed-point representation of value
`fixpt_evenspace_cleanup`	Modify breakpoints of lookup table to have even spacing
`fixpt_interp1`	Implement 1-D lookup table
`fixpt_look1_func_approx`	Optimize fixed-point approximation of nonlinear function by interpolating lookup table data points
`fixpt_look1_func_plot`	Plot fixed-point approximation function for lookup table
`fixpt_set_all`	Set property for each fixed-point block in subsystem
`float`	Create MATLAB structure describing floating-point data type
`fxptdlg`	Invoke Fixed-Point Tool
`num2fixpt`	Convert number to nearest value representable by specified fixed-point data type
`sfix`	Create MATLAB structure describing signed generalized fixed-point data type

`sfrac`	Create MATLAB structure describing signed fractional data type
`sint`	Create MATLAB structure describing signed integer data type
`tunablevars2parameterobjects`	Create Simulink parameter objects from tunable parameters
`ufix`	Create MATLAB structure describing unsigned generalized fixed-point data type
`ufrac`	Create MATLAB structure describing unsigned fractional data type
`uint`	Create MATLAB structure describing unsigned integer data type

4

Functions — Alphabetical List

add_block

Purpose

Add block to model

Syntax

```
add_block('src', 'dest')
block = add_block('src', 'dest')
add_block('src', 'dest', 'param1', value1, ...)
add_block('src', 'dest', 'MakeNameUnique', 'on')
add_block('src_inport', 'dest_inport', 'CopyOption',
   'duplicate')
```

Description

`add_block('src', 'dest')` copies the *'src'* block to a new *'dest'* block. Specify the full path name for *'src'* and *'dest'*, unless the source block is a built-in, nonmasked Simulink block. Instead, you can specify *'src'* to be `'built-in/blocktype'`, where *blocktype* is the value of the BlockType parameter of the source block. If the *'src'* block is a Subsystem block, then `add_block` copies all the blocks in the subsystem. The parameter values of the new block are identical to the parameter values of the source block.

`block = add_block('src', 'dest')` returns the handle of the newly created block.

`add_block('src', 'dest', 'param1', value1, ...)` creates a copy of the *'src'* block, with the named parameters having the specified values. Additional arguments must occur in parameter/value pairs.

`add_block('src', 'dest', 'MakeNameUnique', 'on')` creates a unique name for the new block, based on the name of the *'dest'* block. The `add_block` function creates a unique name only if the *'dest'* block name exists in the model into which you add the new block. By default, `MakeNameUnique` is `off`.

`add_block('src_inport', 'dest_inport', 'CopyOption', 'duplicate')` applies only to Inport blocks. It creates a copy with the same port number as the *'src_inport'* block.

Before you add a block from a source model, open the source model. For example, use `load_system` (model opens invisibly) or `open_system` (model opens visibly). Also open the destination model if it is different from the source model.

Calling `add_block` triggers the `CopyFnc` and `PreCopyFcn` block callback functions.

Definitions

A *built-in block* is a block that is available in Simulink block libraries.

Examples

Copy the Scope block from the Sinks subsystem of the `simulink` system to a block named `Scope` in the `Controller` subsystem of the `f14` system. Open the destination model (`f14`); you do not need to open the `simulink/Sinks` library.

```
open_system('f14');
add_block('simulink/Sinks/Scope', 'f14/Contoller/Scope')
```

Create a subsystem named `Controller2` in the `f14` system.

```
open_system('f14');
add_block('built-in/SubSystem', 'f14/Controller2')
```

Copy the built-in Gain block to a block named `Speed` in the `f14` system and assign the `Gain` parameter a value of 4.

```
add_block('built-in/Gain', 'F14/Speed', 'Gain', '4')
```

Copy the block named `Mu` in `vdp` and create a copy. Because the model already contains a `Mu` block, the command names the new block `Mu1`. Open the `vdp` model, which is both the source and destination model, and get the handle of the added block.

```
open_system('vdp');
block = add_block('vdp/Mu', 'vdp/Mu', 'MakeNameUnique', 'on')
```

See Also

`delete_block` | `replace_block` | `set_param`

Tutorials

- "Creating Block Annotations Programmatically"

How To

- "Adding Blocks to Your Model"
- "Creating Model Callback Functions"

Purpose

Register listener for 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 *Simulink User's Guide* for more information). Simulink software invokes the registered listener whenever the specified event occurs during simulation of the model.

Note Simulink software 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 *Simulink User's Guide*.)

`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.

Purpose

Add line to 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`

Purpose

Add parameter to 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. Simulink software saves these new parameters with the model file.

Note If you attempt to add a parameter that has the same name as an existing parameter of the system, Simulink software displays an error.

Examples

This command

```
add_param('vdp','DemoName','VanDerPolEquation','EquationOrder','2')
```

adds the parameters `DemoName` and `EquationOrder` with string values `'VanDerPolEquation'` and `'2'` to the `vdp` system. Afterward, you can use the following command to retrieve the value of the `DemoName` parameter.

```
get_param('vdp','DemoName')
```

See Also

`delete_param`, `get_param`, `set_param`

addterms

Purpose

Add terminators to unconnected ports in 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`

Purpose

Associate configuration set or configuration reference with model

Syntax

```
attachConfigSet('model', configObj)
attachConfigSet('model', configObj, allowRename)
```

Arguments

model
: The name of an open model, or `gcs` to specify the current model

configObj
: A configuration set (`Simulink.ConfigSet`) or configuration reference (`Simulink.ConfigSetRef`)

allowRename
: Boolean that determines how Simulink software handles a name conflict

Description

`attachConfigSet` associates the configuration set or configuration reference (configuration object) specified by *`configObj`* with *`model`*.

You cannot attach a configuration object to a model if the configuration object is already attached to another model, or has the same name as another configuration object attached to the same model. The optional Boolean argument *`allowRename`* determines how Simulink software handles a name conflict between configuration objects. If *`allowRename`* is `false` and the configuration object specified by *`configObj`* has the same name as a configuration object already attached to *`model`*, Simulink software generates an error. If *`allowRename`* is `true` and a name conflict occurs, Simulink software provides a unique name for *`configObj`* before associating *`configObj`* with *`model`*.

Example

The following example creates a copy of the current model's active configuration object and attaches it to the model, changing its name if necessary to be unique. The code is the same whether the object is a configuration set or configuration reference.

```
myConfigObj = getActiveConfigSet(gcs);
copiedConfig = myConfigObj.copy;
copiedConfig.Name = 'DevConfig';
```

```
attachConfigSet(gcs, copiedConfig, true);
```

See Also

"Setting Up Configuration Sets", "Referencing Configuration Sets"

`attachConfigSetCopy`, `closeDialog`, `detachConfigSet`, `getActiveConfigSet`, `getConfigSet`, `getConfigSets`, `openDialog`, `setActiveConfigSet`

Purpose

Copy configuration set or configuration reference and associate it with model

Syntax

```
myConfigObj = attachConfigSetCopy('model', configObj)
myConfigObj = attachConfigSetCopy('model',
configObj, allowRename)
```

Arguments

model
: The name of an open model, or `gcs` to specify the current model

configObj
: A configuration set (`Simulink.ConfigSet`) or configuration reference (`Simulink.ConfigSetRef`)

allowRename
: Boolean that specifies how Simulink software handles a name conflict

Description

`attachConfigSetCopy` copies the configuration set or configuration reference (configuration object) specified by *`configObj`* and associates the copy with *`model`*. Simulink software returns the copied configuration object as *`newConfigObj`*.

You cannot attach a configuration object to a model if the configuration object has the same name as another configuration object attached to the same model. The optional Boolean argument *`allowRename`* determines how Simulink software handles a name conflict between configuration objects. If *`allowRename`* is `false` and the configuration object specified by *`configObj`* has the same name as a configuration object already attached to *`model`*, Simulink software generates an error. If *`allowRename`* is `true` and a name conflict occurs, Simulink software provides a unique name for the copy of *`configObj`* before associating it with *`model`*.

Example

The following example creates a copy of `ModelA`'s active configuration object and attaches it to `ModelB`, changing the name if necessary to be unique. The code is the same whether the object is a configuration set or configuration reference.

```
myConfigObj = getActiveConfigSet('ModelA');
newConfigObj = attachConfigSetCopy('ModelB', myConfigObj, true);
```

See Also

"Setting Up Configuration Sets", "Referencing Configuration Sets"

`attachConfigSet`, `closeDialog`, `detachConfigSet`, `getActiveConfigSet`, `getConfigSet`, `getConfigSets`, `openDialog`, `setActiveConfigSet`

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`

bdIsLoaded

Purpose

Whether block diagram is in memory

Syntax

```
isLoaded = bdIsLoaded(bdnames)
```

Description

`isLoaded = bdIsLoaded(bdnames)` returns whether or not a block diagram is in memory. `bdnames` can be a string or a cell array of strings. All strings must be valid block diagram names (which are the same as valid MATLAB variable names). It is an error to supply a path to a block or subsystem.

`isLoaded` is a logical array with one entry for each block diagram name. Examples:

Examples

```
bdIsLoaded('sf_car')
```

returns a logical scalar.

```
bdIsLoaded({'sf_car','vdp'})
```

returns a 1*2 logical array.

See Also

`find_system`

Purpose

Return name of 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`

close_system

Purpose

Close Simulink system window or block dialog box

Syntax

```
close_system
close_system('sys')
close_system('sys', saveflag)
close_system('sys', 'newname')
close_system('sys', 'newname','ErrorIfShadowed', true)
```

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` returns an error. The current system is defined in the description of the `gcs` command.

`close_system('sys')` closes the specified system, subsystem, or block window.

`'sys'` can be a string (which can be a system, a subsystem, or a full block pathname), a cell array of strings, a numeric handle, or an array of numeric handles. This command displays an error if *`'sys'`* is a MATLAB keyword, `'simulink'`, or more than 63 characters long.

`close_system('sys', saveflag)`, if *`saveflag`* is 1, saves the specified top-level system to a file with its current name, then closes the specified top-level system window and removes it from memory. If *`saveflag`* is 0, the system is closed without saving. A single *`saveflag`* can be supplied, in which case it is applied to all block diagrams. Alternatively, separate *`saveflag`*s can be supplied for each block diagram, as a numeric array.

`close_system('sys', 'newname')` saves the specified top-level system to a file with the specified new name, then closes the system.

Additional arguments can be supplied when saving a block diagram. These are exactly the same as for `save_system`:

- `ErrorIfShadowed`: true or false (default: false)
- `BreakAllLinks`: true or false (default: false)
- `SaveAsVersion`: MATLAB version name (default: current)

- `OverwriteIfChangedOnDisk`: true or false (default: false)
- `SaveModelWorkspace`: true or false (default: false)

If you try to specify additional options when you are doing something other than saving a block diagram, they are ignored. You see a warning if you try to save when closing something other than a block diagram (e.g., a subsystem or a Block Properties dialog).

Examples

This command closes the current system.

```
close_system
```

This command closes the `vdp` system, unless it has been modified, in which case it returns an error.

```
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', true)
```

All three of the following commands save and close `mymodel` (saved with the same name), and replace links to library blocks with copies of the library blocks in the saved file:

```
close_system('mymodel',1,'BreakAllLinks',true)
```

```
close_system('mymodel','mymodel','BreakAllLinks',true)
close_system('mymodel',[],'BreakAllLinks',true)
```

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')
```

Note The `close_system` command cannot be used in a block or menu callback to close the root-level model. Attempting to close the root-level model in a block or menu callback results in an error and discontinues the callback's execution.

See Also

`bdclose`, `gcs`, `new_system`, `open_system`, `save_system`

Purpose

Close configuration parameters dialog

Syntax

```
closeDialog(configObj)
```

Arguments

configObj

A configuration set (`Simulink.ConfigSet`) or configuration reference (`Simulink.ConfigSetRef`)

Description

`closeDialog` closes an open configuration parameters dialog box. If *`configObj`* is a configuration set, the function closes the dialog box that displays the configuration set. If *`configObj`* is a configuration reference, the function closes the dialog box that displays the referenced configuration set, or generates an error if the reference does not specify a valid configuration set. If the dialog box is already closed, the function does nothing.

Example

The following example closes a configuration parameters dialog box that shows the current parameters for the current model. The parameter values derive from the active configuration set or configuration reference (configuration object). The code is the same in either case; the only difference is which type of configuration object is currently active.

```
myConfigObj = getActiveConfigSet(gcs);
closeDialog(myConfigObj);
```

See Also

"Setting Up Configuration Sets", "Referencing Configuration Sets"

`attachConfigSet`, `attachConfigSetCopy`, `detachConfigSet`, `getActiveConfigSet`, `getConfigSet`, `getConfigSets`, `openDialog`, `setActiveConfigSet`

delete_block

Purpose

Delete block from 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`, `replace_block`

Purpose

Delete line from Simulink system

Syntax

```
delete_line('sys', 'oport', 'iport')
delete_line('system', [x y])
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('system', [x y])` deletes all of the lines in the system that contain the specified point, including any branches.

`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`

delete_param

Purpose

Delete system parameter added via `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

```
add_param('vdp','DemoName','VanDerPolEquation','EquationOrder','2')
delete_param('vdp','DemoName')
```

adds the parameters `DemoName` and `EquationOrder` to the `vdp` system, then deletes `DemoName` from the system.

See Also

`add_param`

Purpose

Dissociate configuration set or configuration reference from model

Syntax

```
detachConfigSet('model', 'configObjName')
```

Arguments

model
: The name of an open model, or `gcs` to specify the current model

configObjName
: The name of a configuration set (`Simulink.ConfigSet`) or configuration reference (`Simulink.ConfigSetRef`)

Description

`detachConfigSet` detaches the configuration set or configuration reference (configuration object) specified by '*configObjName*' from *model*. If no such configuration object is attached to the model, an error occurs.

Examples

The following example detaches the configuration object named `DevConfig` from the current model. The code is the same whether `DevConfig` is a configuration set or configuration reference.

```
detachConfigSet(gcs, 'DevConfig');
```

See Also

"Setting Up Configuration Sets", "Referencing Configuration Sets"

`attachConfigSet`, `attachConfigSetCopy`, `closeDialog`, `getActiveConfigSet`, `getConfigSet`, `getConfigSets`, `openDialog`, `setActiveConfigSet`

disableimplicitsignalresolution

Purpose

Convert model to use only explicit signal resolution

Syntax

```
retVal = disableimplicitsignalresolution(model)
retVal = disableimplicitsignalresolution(model,
displayOnly)
```

Description

`retVal = disableimplicitsignalresolution(model)` inputs a model, reports all signals and states that implicitly resolve to signal objects, and converts the model to resolve only signals and states that explicitly require it. The report and any changes are limited to the model itself; they do not include blocks that are library links.

Before executing this function, ensure that all relevant Simulink data objects are defined in the base workspace. The function ignores any data objects that are defined elsewhere.

The function scans *`model`*, returns a structure of handles to signals and states that resolve implicitly to signal objects, and performs the following operations on *`model`*:

- Search the model for all output ports and block states that resolve to Simulink signal objects.
- Modify these ports and blocks to enforce signal object resolution in the future.
- Set the model's `SignalResolutionControl` parameter to `'UseLocalSettings'` (GUI: **Explicit Only**.
- If any Stateflow output data resolves to a Simulink signal object:
  - Turn off hierarchical scoping of signal objects from within the Stateflow chart.
  - Explicitly label the output signal of the Stateflow chart.
  - Enforce signal object resolution for this signal in the future.

Any changes made by `disableimplicitsignalresolution` permanently change the model. Be sure to back up the model before calling the function with *`displayOnly`* defaulted to or specified as `false`.

retVal = disableimplicitsignalresolution(*model*, *displayOnly*) is equivalent to disableimplicitsignalresolution(*model*) if *displayOnly* is false.

If *displayOnly* is true, the function returns a structure of handles to signals and states that resolve implicitly to signal objects, but leaves the model unchanged.

Inputs

displayOnly

Boolean specifying whether to change the model (false) or just generate a report (true)

Default: false

model

Model name or handle

Output

retVal

A MATLAB structure containing:

Signals	Handles to ports with signal names that resolve to signal objects
States	Handles to blocks with states that resolve to signal objects

See Also

Simulink.Signal

How To

- "Data Validity Diagnostics Overview"
- "Resolving Symbols"
- "Signal Properties Dialog Box"

docblock

Purpose

Get or set editor invoked by Simulink DocBlock

Syntax

```
docblock('setEditorHTML', editCmd)
docblock('setEditorDOC', editCmd)
docblock('setEditorTXT', editCmd)
editCmd = docblock('getEditorHTML')
editCmd = docblock('getEditorDOC')
editCmd = docblock('getEditorTXT')
```

Description

`docblock('setEditorHTML', editCmd)` sets the HTML editor invoked by a DocBlock. The *`editCmd`* string specifies a command, executed at the MATLAB prompt, which launches a custom HTML editor. By default, a DocBlock invokes Microsoft Word (if available) as the HTML editor; otherwise, it opens HTML documents using the editor you specified on the **Editor/Debugger Preferences** pane of the Preferences dialog box.

Use the `"%<FileName>"` token in the *`editCmd`* string to represent the full pathname to the document. Use the empty string `''` as the *`editCmd`* to reset the DocBlock to its default editor for a particular document type.

`docblock('setEditorDOC', editCmd)` sets the Rich Text Format (RTF) editor invoked by a DocBlock. The *`editCmd`* string specifies a command, executed at the MATLAB prompt, which launches a custom RTF editor. By default, a DocBlock invokes Microsoft Word (if available) as the RTF editor. Otherwise, it opens RTF documents using the editor you specified on the **Editor/Debugger Preferences** pane of the Preferences dialog box.

`docblock('setEditorTXT', editCmd)` sets the text editor invoked by a DocBlock. The *`editCmd`* string specifies a command, executed at the MATLAB prompt, which launches a custom text editor. By default, a DocBlock invokes the editor you specified on the **Editor/Debugger Preferences** pane of the Preferences dialog box.

`editCmd` `= docblock('getEditorHTML')` returns the value of the current command used to invoke an HTML editor when double-clicking a DocBlock.

editCmd = docblock('getEditorDOC') returns the value of the current command used to invoke a RTF editor when double-clicking a DocBlock.

editCmd = docblock('getEditorTXT') returns the value of the current command used to invoke a text editor when double-clicking a DocBlock.

Examples

Specify Microsoft Notepad as the DocBlock editor for RTF documents:

```
docblock('setEditorRTF','system(''notepad "%<FileName>"'');')
```

Reset the DocBlock to use its default editor for RTF documents:

```
docblock('setEditorRTF','')
```

Specify Mozilla Composer as the HTML editor for the DocBlock:

```
docblock('setEditorHTML','system(''/usr/local/bin/mozilla ...
  -edit "%<FileName>" &'');')
```

See Also

DocBlock

How To

·

find_mdlrefs

Purpose

Find Model blocks and referenced models at all levels or at top level only

Syntax

```
[refMdls, mdlBlks] = find_mdlrefs(modelName)
[refMdls, mdlBlks] = find_mdlrefs(modelName, allLevels)
[refMdls, mdlBlks] = find_mdlrefs(modelName,
'Param1', Val1,
   ... ,'ParamN', ValN)
```

Description

`[refMdls, mdlBlks] = find_mdlrefs(modelName)` finds all Model blocks and referenced models contained by the model hierarchy of which *`modelName`* is the top model.

`[refMdls, mdlBlks] = find_mdlrefs(modelName, allLevels)` is equivalent to the preceding syntax if *`allLevels`* is `true`. If *`allLevels`* is `false`, the function searches only the top level of *`modelName`*, ignoring any subordinate hierarchy.

`[refMdls, mdlBlks] = find_mdlrefs(modelName, 'Param1', Val1, ... ,'ParamN', ValN)` searches the model as specified by the optional name/value pairs *`'Param1', Val1, ... , 'ParamN', ValN`*.

Inputs

`modelName`

The model to be searched for Model blocks and referenced models.

`allLevels`

Boolean specifying whether to search the complete hierarchy (`true`) or only the top model *`modelName`* (`false`). Default: `false`.

`'Param1', Val1, ... , 'ParamN', ValN`

One or more name/value pairs that control the function's actions. The possible names and values, and their defaults, are:

`'AllLevels'`	Boolean specifying whether to search the complete hierarchy (`true`) or only the top level (`false`). The default if you omit `'AllLevels'` is `true`: the function searches the complete hierarchy of which *`modelName`* is the top model.
`'IncludeProtectedModels'`	Boolean that specifies whether the output *`refMdls`* includes the names of protected models (`true`) or excludes them(`false`). The default if you omit `'IncludeProtectedModels'` is `false`: the function excludes protected models from *`refMdls`*.
`'Variants'`	String that specifies whether the output *`refMdls`* includes the names of variant models. The possible values are: • `'ActiveVariants'` — Include the active variant of each variant Model block included in the search. • `'ActivePlusCodeVariants'` — Include all variants for all variant Model blocks included in the search whose **Generate preprocessor conditionals** option is selected.

- `'Allvariants'` — Include all variants for all variant Model blocks included in the search.

The default if you omit `'Variants'` is `'ActivePlusCodeVariants'`.

Note that `find_mdlrefs` provides two different ways to search all levels of the model *`modelName`*. Both techniques give the same results, but only the name/value technique allows you to control inclusion of protected and variant models in *`refMdls`*.

Outputs

`mdlBlks`

A list containing the names of all Model blocks in the hierarchy of which *`modelName`* is the top model, or in *`modelName`* alone, depending on the value of `'AllLevels'`.

`refMdls`

A list containing the names of all models referenced by Model block in the hierarchy of which *`modelName`* is the top model, or in *`modelName`* alone, depending on the value of `'AllLevels'`. Optional inputs to `find_mdlrefs` control inclusion of protected and variant models. The last element of the `refMdls` is *`modelName`* itself.

Example

Return in `MyMdlBlks` the names all Model blocks at the top level of `MyModel`. Return in `MyRefMdls` the names of all models referenced by Model blocks at the top level, including the active variant of every variant Model block:

```
[MyRefMdls, MyMdlBlks] = find_mdlrefs(MyModel, 'AllLevels', false, 'Va
```

See Also

Model | `view_mdlrefs`

How To

- “Referencing a Model”
- “Using Model Reference Variants”
- “Protecting Referenced Models”

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 8, "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 software.

```
find_system('clutch', 'regexp', 'on', 'blocktype', 'port')
```

See Also

`get_param`, `set_param`

fixdt

Purpose

Create `Simulink.NumericType` object describing fixed-point or floating-point data type

Syntax

```
a = fixdt(Signed, WordLength)
a = fixdt(Signed, WordLength, FractionLength)
a = fixdt(Signed, WordLength, TotalSlope, Bias)
a = fixdt(Signed, WordLength, SlopeAdjustmentFactor,
    FixedExponent, Bias)
a = fixdt(DataTypeNameString)
[DataType,IsScaledDouble] = fixdt(DataTypeNameString)
```

Description

`fixdt(Signed, WordLength)` returns a `Simulink.NumericType` object describing a fixed-point data type with unspecified scaling. The scaling would typically be determined by another block parameter. `Signed` can be `0` (false) for unsigned or `1` (true) for signed.

`fixdt(Signed, WordLength, FractionLength)` returns a `Simulink.NumericType` object describing a fixed-point data type with binary point scaling.

`fixdt(Signed, WordLength, TotalSlope, Bias)` or `fixdt(Signed, WordLength, SlopeAdjustmentFactor, FixedExponent, Bias)` returns a `Simulink.NumericType` object describing a fixed-point data type with slope and bias scaling.

`fixdt(DataTypeNameString)` returns a `Simulink.NumericType` object describing an integer, fixed-point, or floating-point data type specified by a data type name. The data type name can be either the name of a built-in Simulink data type or the name of a fixed-point data type that conforms to the naming convention for fixed-point names established by the Simulink Fixed Point product. For more information, see "Fixed-Point Data Type and Scaling Notation" in the Simulink Fixed Point documentation.

`[DataType,IsScaledDouble] = fixdt(DataTypeNameString)` returns a `Simulink.NumericType` object describing an integer, fixed-point, or floating-point data type specified by a data type name and a flag

that indicates whether the specified data type name was the name of a scaled double data type.

See Also

`float`, `sfix`, `sfrac`, `sint`, `ufix`, `ufrac`, `uint`

fixpt_evenspace_cleanup

Purpose

Modify breakpoints of lookup table to have even spacing

Syntax

```
xdata_modified = fixpt_evenspace_cleanup(xdata,xdt,xscale)
```

Description

`xdata_modified = fixpt_evenspace_cleanup(xdata,xdt,xscale)` modifies breakpoints of a lookup table to have even spacing after quantization. By adjusting breakpoints to have even spacing after quantization, Real-Time Workshop generated code can exclude breakpoints from memory.

`xdata` is the breakpoint vector of a lookup table to make evenly spaced, such as `0:0.005:1`. *`xdt`* is the data type of the breakpoints, such as `sfix(16)`. *`xscale`* is the scaling of the breakpoints, such as `2^-12`. Using these three inputs, `fixpt_evenspace_cleanup` returns the modified breakpoints in *`xdata_modified`*.

This function works only with nontunable data and considers data to have even spacing relative to the scaling slope. For example, the breakpoint vector [0 2 5], which has spacing value 2 and 3, appears to have uneven spacing. However, the difference between the maximum spacing 3 and the minimum spacing 2 equals 1. If the scaling slope is 1 or greater, a spacing variation of 1 represents a 1-bit change or less. In this case, the `fixpt_evenspace_cleanup` function considers a spacing variation of 1 bit or less to be even.

Modifications to breakpoints can change the numerical behavior of a lookup table. To check for changes, test the model using simulation, rapid prototyping, or other appropriate methods.

Examples

Modify breakpoints of a lookup table to have even spacing after quantization:

```
xdata = 0:0.005:1;
xdt = sfix(16);
xscale = 2^-12;
xdata_modified = fixpt_evenspace_cleanup(xdata,xdt,xscale)
```

See Also

`fixdt` | `fixpt_interp1` | `fixpt_look1_func_approx` | `fixpt_look1_func_plot` | `sfix` | `ufix`

Tutorials

- "Effects of Spacing on Speed, Error, and Memory Usage"
- "Creating Lookup Tables for a Sine Function"

fixpt_interp1

Purpose

Implement 1-D lookup table

Syntax

```
y = fixpt_interp1(xdata,ydata,x,xdt,xscale,ydt,yscale,
   rndmeth)
```

Description

`y = fixpt_interp1(xdata,ydata,x,xdt,xscale,ydt,yscale,rndmeth)` implements a one-dimensional lookup table to find output `y` for input `x`. If `x` falls between two `xdata` values (breakpoints), `y` is the result of interpolating between the corresponding `ydata` values. If `x` is greater than the maximum value in `xdata`, `y` is the maximum `ydata` value. If `x` is less than the minimum value in `xdata`, `y` is the minimum `ydata` value.

If the input data type `xdt` or the output data type `ydt` is floating point, `fixpt_interp1` performs the interpolation using floating-point calculations. Otherwise, `fixpt_interp1` uses integer-only calculations. These calculations handle the input scaling `xscale` and the output scaling `yscale` and obey the rounding method `rndmeth`.

Inputs

`xdata`

Vector of breakpoints for the lookup table, such as `linspace(0,8,33)`.

`ydata`

Vector of table data that correspond to the breakpoints for the lookup table, such as `sin(xdata)`.

`x`

Vector of input values for the lookup table to process, such as `linspace(-1,9,201)`.

`xdt`

Data type of input `x`, such as `sfix(8)`.

`xscale`

Scaling for input `x`, such as `2^-3`.

`ydt`

Data type of output `y`, such as `sfix(16)`.

`yscale`

Scaling for output `y`, such as `2^-14`.

`rndmeth`

Rounding mode supported by fixed-point Simulink blocks:

`'Ceiling'`	Round to the nearest representable number in the direction of positive infinity.
`'Floor'` (default)	Round to the nearest representable number in the direction of negative infinity.
`'Nearest'`	Round to the nearest representable number.
`'Toward Zero'`	Round to the nearest representable number in the direction of zero.

Examples

Interpolate outputs for `x` using a 1-D lookup table that approximates the sine function:

```
xdata = linspace(0,8,33).';
ydata = sin(xdata);
% Define input x as a vector of 201 evenly
% spaced points between -1 and 9 (includes
% values both lower and higher than the range
% of breakpoints in xdata)
x = linspace(-1,9,201).';
```

```
% Interpolate output values for x
y = fixpt_interp1(xdata,ydata,x,sfix(8),2^-3,sfix(16),...
 2^-14,'Floor')
```

See Also

`fixpt_evenspace_cleanup` | `fixpt_look1_func_approx` | `fixpt_look1_func_plot`

Tutorials

- "Tutorial: Producing Lookup Table Data"

Purpose

Optimize fixed-point approximation of nonlinear function by interpolating lookup table data points

Syntax

```
[xdata,ydata,errworst] = fixpt_look1_func_approx('func',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,nptsmax)
[xdata,ydata,errworst] = fixpt_look1_func_approx('func',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,[])
[xdata,ydata,errworst] = fixpt_look1_func_approx('func',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,[],nptsmax)
[xdata,ydata,errworst] = fixpt_look1_func_approx('func',...
xmin,xmax,xdt,xscale,ydtydt,yscale,rndmeth,errmax,nptsmax,spacing)
```

Description

`[xdata,ydata,errworst] = fixpt_look1_func_approx('func',... xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,nptsmax)` returns the optimal breakpoints of a lookup table, an ideal function applied to the breakpoints, and the worst-case approximation error. The lookup table satisfies the maximum acceptable error and maximum number of points that you specify.

`[xdata,ydata,errworst] = fixpt_look1_func_approx('func',... xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,[])` returns the optimal breakpoints of a lookup table, an ideal function applied to the breakpoints, and the worst-case approximation error. The lookup table satisfies the maximum acceptable error that you specify.

`[xdata,ydata,errworst] = fixpt_look1_func_approx('func',... xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,[],nptsmax)` returns the optimal breakpoints of a lookup table, an ideal function applied to the breakpoints, and the worst-case approximation error. The lookup table satisfies the maximum number of points that you specify.

`[xdata,ydata,errworst] = fixpt_look1_func_approx('func',... xmin,xmax,xdt,xscale,ydtydt,yscale,rndmeth,errmax,nptsmax,spacing)` returns the optimal breakpoints of a lookup table, an ideal function applied to the breakpoints, and the worst-case approximation error. The lookup table satisfies the maximum acceptable error, maximum number of points, and breakpoint spacing that you specify.

In each case, `fixpt_look1_func_approx` interpolates between lookup table data points to optimize the fixed-point approximation. The inputs *`xmin`* and *`xmax`* specify the range over which to approximate the breakpoints. The inputs *`xdt`*, *`xscale`*, *`ydt`*, *`yscale`*, and *`rndmeth`* follow conventions used by fixed-point Simulink blocks.

The inputs *`errmax`*, *`nptsmax`*, and *`spacing`* are optional. Of these inputs, you must specify at least *`errmax`* or *`nptsmax`*. If you omit one of those two inputs, you must use brackets, `[]`, in place of the omitted input. `fixpt_look1_func_approx` ignores that requirement for the lookup table.

If you do not specify spacing, and more than one spacing satisfies *`errmax`* and *`nptsmax`*, `fixpt_look1_func_approx` chooses in this order: power-of-2 spacing, even spacing, uneven spacing. This behavior applies when you specify both *`errmax`* and *`nptsmax`*, but not when you specify just one of the two.

Inputs

`func`

Function of `x` for which to approximate breakpoints. Enclose this expression in single quotes, for example, `'sin(2*pi*x)'`.

`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:

`'Ceiling'`	Round to the nearest representable number in the direction of positive infinity.
`'Floor'` (default)	Round to the nearest representable number in the direction of negative infinity.
`'Nearest'`	Round to the nearest representable number.
`'Toward Zero'`	Round to the nearest representable number in the direction of zero.

`errmax`

Maximum acceptable error between the ideal function and the approximation given by the lookup table.

`nptsmax`

Maximum number of points for the lookup table.

`spacing`

Spacing of breakpoints for the lookup table:

`'even'`	Even spacing
`'pow2'`	Even, power-of-2 spacing
`'unrestricted'` (default)	Uneven spacing

If you specify...	The breakpoints of the lookup table...
errmax and *nptsmax*	Meet both criteria, if possible. The *errmax* requirement has higher priority than *nptsmax*. If the breakpoints cannot meet both criteria with the specified spacing, *nptsmax* does not apply.
errmax only	Meet the error criteria, and `fixpt_look1_func_approx` returns the fewest number of points.
nptsmax only	Meet the points criteria, and `fixpt_look1_func_approx` returns the smallest worst-case error.

Outputs

xdata

Vector of breakpoints for the lookup table.

ydata

Vector of values from applying the ideal function to the breakpoints.

errworst

Worst-case error, which is the maximum absolute error between the ideal function and the approximation given by the lookup table.

Examples

Approximate a fixed-point sine function using a lookup table:

```
func = 'sin(2*pi*x)';
% Define the range over which to optimize breakpoints
xmin = 0;
xmax = 0.25;
% Define the data type and scaling for the inputs
xdt = ufix(16);
xscale = 2^-16;
% Define the data type and scaling for the outputs
ydt = sfix(16);
yscale = 2^-14;
% Specify the rounding method
rndmeth = 'Floor';
% Define the maximum acceptable error
errmax = 2^-10;
% Choose even, power-of-2 spacing for breakpoints
spacing = 'pow2';
% Create the lookup table
[xdata,ydata,errworst] = fixpt_look1_func_approx(func,...
 xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,[],spacing);
```

See Also

`fixpt_evenspace_cleanup` | `fixpt_interp1` | `fixpt_look1_func_plot`

Tutorials

- "Tutorial: Producing Lookup Table Data"

How To

- "Summary for Using Lookup Table Approximation Functions"

fixpt_look1_func_plot

Purpose

Plot fixed-point approximation function for lookup table

Syntax

```
fixpt_look1_func_plot(xdata,ydata,'func',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth)
errworst = fixpt_look1_func_plot(xdata,ydata,'func',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth)
```

Description

`fixpt_look1_func_plot(xdata,ydata,'func',... xmin,xmax,xdt,xscale,ydt,yscale,rndmeth)` plots a lookup table approximation function and the error from the ideal function.

`errworst = fixpt_look1_func_plot(xdata,ydata,'func',... xmin,xmax,xdt,xscale,ydt,yscale,rndmeth)` plots a lookup table approximation function and the error from the ideal function. The output `errworst` is the maximum absolute error.

You can use `fixpt_look1_func_approx` to generate `xdata` and `ydata`, the breakpoints and table data for the lookup table, respectively. `fixpt_look1_func_approx` applies the ideal function to the breakpoints in `xdata` to produce `ydata`. While this method is the easiest way to generate `ydata`, you can choose other values for `ydata` as input for `fixpt_look1_func_plot`. Choosing different values for `ydata` can, in some cases, produce a lookup table with a smaller maximum absolute error.

Inputs

`xdata`

Vector of breakpoints for the lookup table.

`ydata`

Vector of values from applying the ideal function to the breakpoints.

`func`

Function of `x` for which to approximate breakpoints. Enclose this expression in single quotes, for example, `'sin(2*pi*x)'`.

`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:

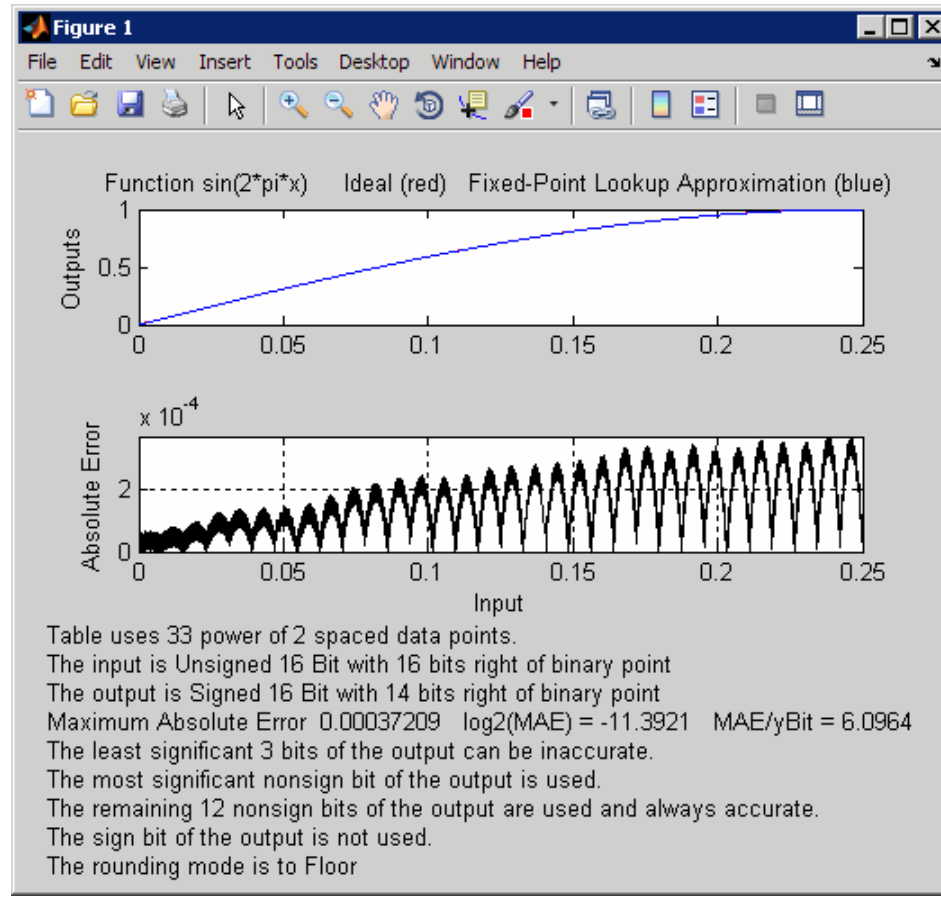
`'Ceiling'`	Round to the nearest representable number in the direction of positive infinity.
`'Floor'` (default)	Round to the nearest representable number in the direction of negative infinity.
`'Nearest'`	Round to the nearest representable number.
`'Toward Zero'`	Round to the nearest representable number in the direction of zero.

Examples

Plot a fixed-point approximation of the sine function using data points generated by `fixpt_look1_func_approx`:

```
func = 'sin(2*pi*x)';
% Define the range over which to optimize breakpoints
xmin = 0;
xmax = 0.25;
% Define the data type and scaling for the inputs
xdt = ufix(16);
xscale = 2^-16;
% Define the data type and scaling for the outputs
ydt = sfix(16);
yscale = 2^-14;
% Specify the rounding method
rndmeth = 'Floor';
% Define the maximum acceptable error
errmax = 2^-10;
% Choose even, power-of-2 spacing for breakpoints
spacing = 'pow2';
% Generate data points for the lookup table
[xdata,ydata,errworst]=fixpt_look1_func_approx(func,...
 xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,[],spacing);
% Plot the sine function (ideal and fixed-point) & errors
fixpt_look1_func_plot(xdata,ydata,func,xmin,xmax,...
 xdt,xscale,ydt,yscale,rndmeth);
```

`fixpt_look1_func_plot` plots the fixed-point sine function, using generated data points, and plots the error between the ideal function and the fixed-point function. The maximum absolute error and the number of points required appear on the plot. The error drops to zero at a breakpoint, but increases between breakpoints due to curvature differences between the ideal function and the line drawn between breakpoints.



The lookup table requires 33 points to achieve a maximum absolute error of `2^-11.3922`.

See Also

`fixpt_evenspace_cleanup | fixpt_interp1 | fixpt_look1_func_approx`

Tutorials

- "Tutorial: Producing Lookup Table Data"

How To

- "Summary for Using Lookup Table Approximation Functions"

Purpose

Set property for each fixed-point block in subsystem

Syntax

```
fixpt_set_all(SystemName,fixptPropertyName,
   fixptPropertyValue)
```

Description

`fixpt_set_all(`*`SystemName`*`,`*`fixptPropertyName`*`,`*`fixptPropertyValue`*`)` sets the property `fixptPropertyName` of every applicable block in the model or subsystem *`SystemName`* to the value *`fixptPropertyValue`*

Examples

Set each fixed-point block in a model `Filter_1` to round towards the floor and saturate upon overflow:

```
% Round towards the floor
fixpt_set_all('Filter_1','RndMeth','Floor')

% Saturate upon overflow
fixpt_set_all('Filter_1','DoSatur','on')
```

fixptbestexp

Purpose

Exponent that gives best precision for fixed-point representation of value

Syntax

```
out = fixptbestexp(RealWorldValue, TotalBits, IsSigned)
out = fixptbestexp(RealWorldValue, FixPtDataType)
```

Description

`out = fixptbestexp(RealWorldValue, TotalBits, IsSigned)` returns the exponent that gives the best precision for the fixed-point representation of `RealWorldValue`. `TotalBits` specifies the number of bits for the fixed-point number. `IsSigned` specifies whether the fixed-point number is signed: `1` indicates the number is signed and `0` indicates the number is not signed.

`out = fixptbestexp(RealWorldValue, FixPtDataType)` returns the exponent that gives the best precision based on the data type `FixPtDataType`.

Examples

Get the exponent that gives the best precision for the real-world value 4/3 using a signed, 16-bit number:

```
out = fixptbestexp(4/3,16,1)
out =
   -14
```

Alternatively, specify the fixed-point data type:

```
out = fixptbestexp(4/3,sfix(16))
out =
   -14
```

This shows that the maximum precision representation of 4/3 is obtained by placing 14 bits to the right of the binary point:

01.01010101010101

You can specify the precision of this representation in fixed-point blocks by setting the scaling to `2^-14` or `2^fixptbestexp(4/3,16,1)`.

See Also

`fixptbestprec`

fixptbestprec

Purpose

Determine maximum precision available for fixed-point representation of value

Syntax

```
out = fixptbestprec(RealWorldValue,TotalBits,IsSigned)
out = fixptbestprec(RealWorldValue,FixPtDataType)
```

Description

`out = fixptbestprec(RealWorldValue,TotalBits,IsSigned)` determines the maximum precision for the fixed-point representation of the real-world value specified by `RealWorldValue`. You specify the number of bits for the fixed- point number with `TotalBits`, and you specify whether the fixed-point number is signed with `IsSigned`. If `IsSigned` is `1`, the number is signed. If `IsSigned` is `0`, the number is not signed. The maximum precision is returned to `out`.

`out = fixptbestprec(RealWorldValue,FixPtDataType)` determines the maximum precision based on the data type specified by `FixPtDataType`.

Examples

Example 1

The following command returns the maximum precision available for the real-world value 4/3 using a signed, 8-bit number:

```
out = fixptbestprec(4/3,8,1)
out =
   0.015625
```

Alternatively, you can specify the fixed-point data type:

```
out = fixptbestprec(4/3,sfix(8))
out =
   0.015625
```

This value means that the maximum precision available for 4/3 is obtained by placing six bits to the right of the binary point since 2^{-6} equals 0.015625:

01.010101

Example 2

You can use the maximum precision as the scaling in fixed-point blocks. This enables you to use `fixptbestprec` to perform a type of autoscaling if you would like to designate a known range of your simulation. For example, if your known range is -13 to 22, and you are using a safety margin of 30%:

```
knownMax = 22;
knownMin = -13;
localSafetyMargin = 30;
slope = max( fixptbestprec( (1+localSafetyMargin/100)* ...
 [knownMax,knownMin], sfix(16) ) );
```

The variable `slope` can then be used in the expression that you specify for the **Output data type** parameter in a block mask. Be sure to select the **Lock output data type setting against changes by the fixed-point tools** check box in the same block to prevent the Fixed-Point Tool from overriding the scaling. If you know the range, you can use this technique in place of relying on a model simulation to provide the range to the autoscaling tool, as described in `autofixexp` in the Simulink Fixed Point documentation.

See Also

`fixptbestexp`

float

Purpose

Create MATLAB structure describing floating-point data type

Syntax

```
a = float('single')
a = float('double')
a = float(TotalBits, ExpBits)
```

Description

`float('single')` returns a MATLAB structure that describes the data type of an IEEE single (32 total bits, 8 exponent bits).

`float('double')` returns a MATLAB structure that describes the data type of an IEEE double (64 total bits, 11 exponent bits).

`float(TotalBits, ExpBits)` returns a MATLAB structure that describes a nonstandard floating-point data type that mimics the IEEE style. That is, the numbers are normalized with a hidden leading one for all exponents except the smallest possible exponent. However, the largest possible exponent might not be treated as a flag for Infs and NaNs.

Note `float(TotalBits, ExpBits)` will be removed in a future release.

`float` is automatically called when a floating-point number is specified in a block dialog box.

Examples

Define a nonstandard, IEEE style, floating-point data type with 31 total bits (excluding the hidden leading one) and 9 exponent bits:

```
a = float(31,9)
a =
       Class: 'FLOAT'
    MantBits: 21
     ExpBits: 9
```

See Also

`fixdt`, `sfix`, `sfrac`, `sint`, `ufix`, `ufrac`, `uint`

Purpose

Edit print frames for Simulink and Stateflow block diagrams

Syntax

```
frameedit
frameedit filename
```

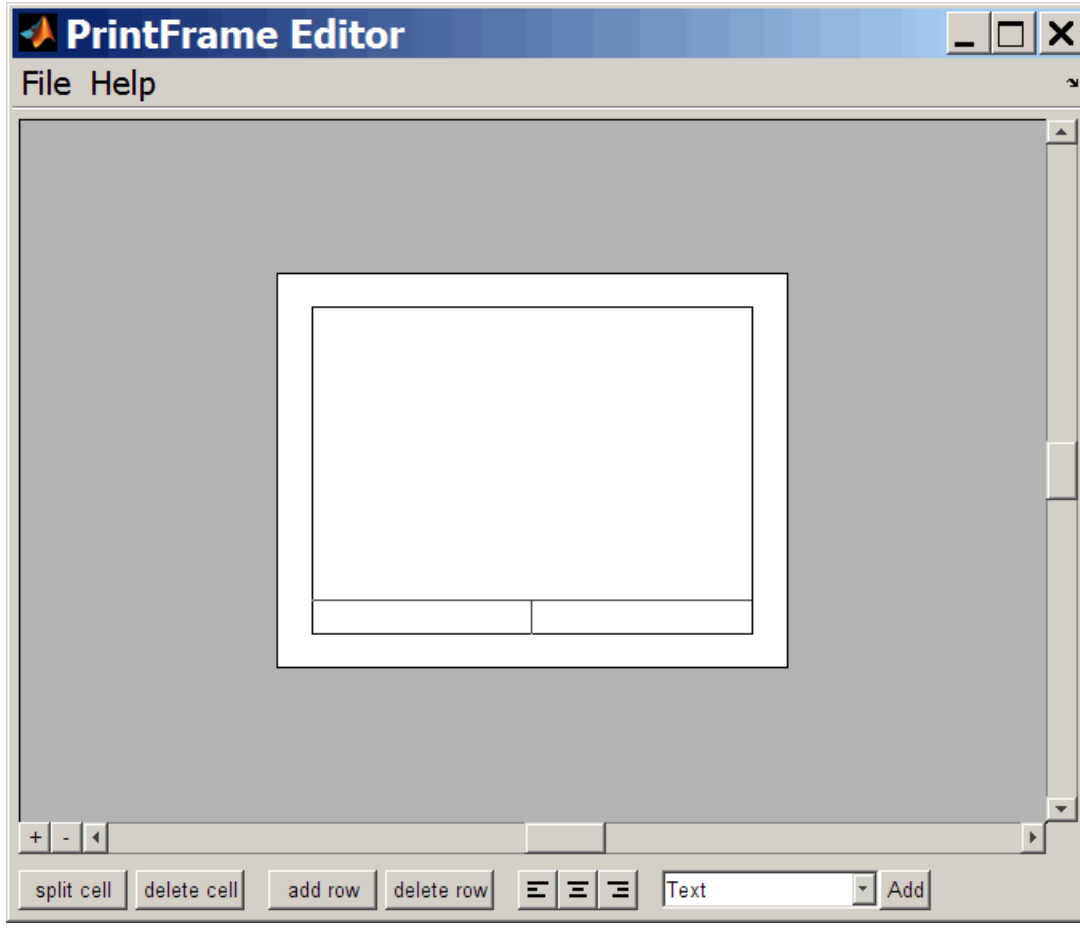
Description

`frameedit` starts the PrintFrame Editor, a graphical user interface you use to create borders for Simulink and Stateflow block diagrams. With no argument, `frameedit` opens the **PrintFrame Editor** window with a new file.

`frameedit filename` opens the **PrintFrame Editor** window with the specified filename, where `filename` is a figure file (`.fig`) previously created and saved using `frameedit`.

Remarks

This illustrates the main features of the PrintFrame Editor.



Closing the PrintFrame Editor

To close the **PrintFrame Editor** window, click the close box in the upper right corner, or select **Close** from the **File** menu.

Printing Simulink Block Diagrams with Print Frames

Select **Print** from the Simulink **File** menu. Check the **Frame** box and supply the filename for the print frame you want to use. Click **OK** in the **Print** dialog box.

Getting Help for the PrintFrame Editor

For further instructions on using the PrintFrame Editor, select **PrintFrame Editor Help** from the **Help** menu in the PrintFrame Editor.

fxptdlg

Purpose

Invoke Fixed-Point Tool

Syntax

```
fxptdlg('modelname')
```

Description

`fxptdlg('modelname')` launches the Fixed-Point Tool for the Simulink model specified by `modelname`. You can also access this tool by the following methods:

- From the Simulink **Tools** menu, select **Fixed-Point > Fixed-Point Tool**.
- From a subsystem context (right-click) menu, select **Fixed-Point > Fixed-Point Tool**.

In conjunction with Simulink Fixed Point software, the Fixed-Point Tool provides convenient access to:

- Model and subsystem parameters that control the signal logging, fixed-point instrumentation mode, and data type override, namely, `MinMaxOverflowArchiveMode`, `MinMaxOverflowLogging`, and `DataTypeOverride` (see "Model Parameters" on page 8-2)
- Plotting capabilities that enable you to plot data that resides in the MATLAB workspace, namely, simulation results associated with Scope, To Workspace, and root-level Outport blocks, in addition to logged signal data (see "Logging Signals" in the *Simulink User's Guide*)
- An interactive autoscaling feature that proposes fixed-point scaling for appropriately configured objects in your model, and then allows you to selectively accept and apply the scaling proposals

You can launch the Fixed-Point Tool for any system or subsystem, and the tool controls the object selected in its **Model Hierarchy** pane. If Simulink Fixed Point software is installed, the Fixed-Point Tool **Contents** pane displays the name, data type, design minimum and maximum values, minimum and maximum simulation values, and scaling of each model object that logs fixed-point data. Additionally, if

a signal saturates or overflows, the tool displays the number of times saturation or overflow occurred. You can display an object's dialog box by right-clicking the appropriate entry in the **Contents** pane and selecting **Properties**.

Note The Fixed-Point Tool works only for models that simulate in Normal mode. The tool does not work when you simulate your model in Accelerator or Rapid Accelerator mode (see "Accelerating Models" in *Simulink User's Guide*).

Most of the functionality in the Fixed-Point Tool is for use with the Simulink Fixed Point software. However, even if you do not have Simulink Fixed Point software, you can use data type override to simulate a model that specifies fixed-point data types. In this mode, the Simulink software replaces fixed-point data types with floating-point data types when simulating the model. Data type override mode allows you to share fixed-point models with people in your company who do not have Simulink Fixed Point software.

To simulate a model without using Simulink Fixed Point:

1. From the Simulink **Tools** menu, select **Fixed-Point > Fixed-Point Tool**.

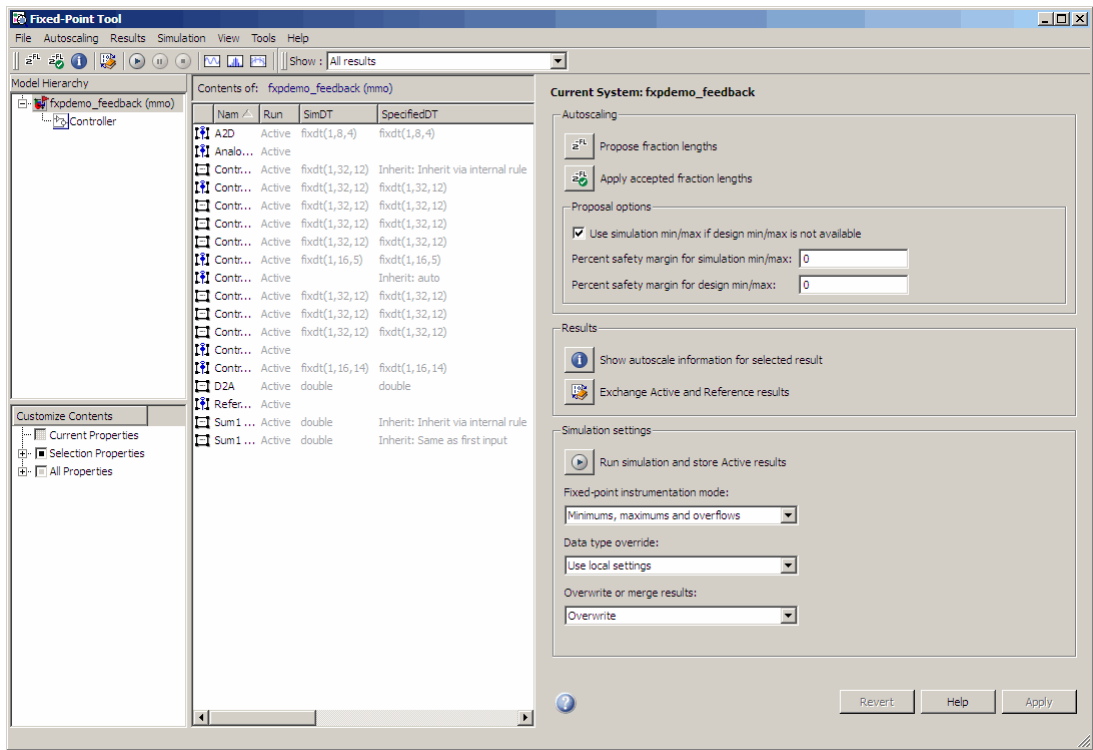
   The Fixed-Point Tool appears.

2. Set the **Fixed-point instrumentation mode** parameter to `Force off`.

3. Set the **Data type override** parameter to `True doubles` or `True singles`.

Note If a parameter in your model specifies a `fi` object, you can prevent the checkout of a Fixed-Point Toolbox license by setting the `fipref` `DataTypeOverride` property to `TrueDoubles`. See the Fixed-Point Toolbox documentation for more information.

Parameters and Dialog Box

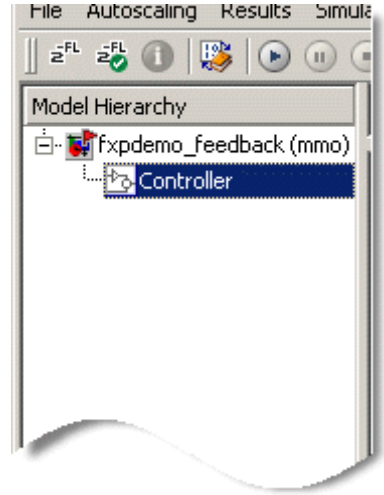


The Fixed-Point Tool includes the following components:

- **Model Hierarchy** pane (see "Model Hierarchy Pane" on page 4-69)
- **Contents** pane (see "Contents Pane" on page 4-70)
- **Dialog** pane (see "Dialog Pane" on page 4-77)
- **Main** toolbar (see "Main Toolbar" on page 4-93)

Model Hierarchy Pane

The **Model Hierarchy** pane displays a tree-structured view of the Simulink model hierarchy. The first node in the pane represents a Simulink model. Expanding the root node displays subnodes that represent the model's subsystems, Embedded MATLAB Function blocks, Stateflow charts, and referenced models.



The Fixed-Point Tool's **Contents** pane displays elements that comprise the object selected in the **Model Hierarchy** pane. The **Dialog** pane provides parameters for specifying the selected object's data type override and fixed-point instrumentation mode. Objects that control the **Fixed-point instrumentation mode** parameter display a red flag on

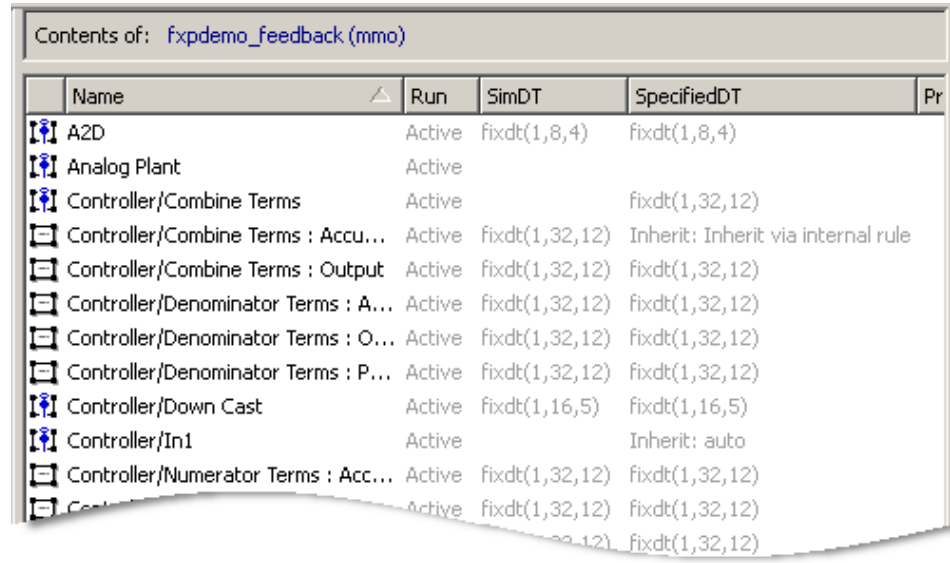
their icons, while those that control the **Data type override** parameter display a green flag. The **Model Hierarchy** pane indicates the value of these parameters by displaying the following abbreviations next to the object name:

Abbreviation	Parameter Value
Fixed-point instrumentation mode	
mmo	Minimums, maximums and overflows
o	Overflows only
fo	Force off
Data type override	
scl	Scaled doubles
dbl	True doubles
sgl	True singles
fo	Force off

See “Dialog Pane” on page 4-77 for more information about these parameters.

Contents Pane

The **Contents** pane displays a tabular view of objects that log fixed-point data in the system or subsystem selected in the **Model Hierarchy** pane. The table rows correspond to model objects, such as blocks, block parameters, and Stateflow data. The table columns correspond to attributes of those objects, such as the data type, design minimum and maximum values, and simulation minimum and maximum values.



Note The **Contents** pane displays information only after you simulate a system or propose fraction lengths.

The **Contents** pane displays columns that correspond to the following properties and controls:

Column Label	Description
Name	Identifies path and name of block.
Run	Indicates whether the Fixed-Point Tool stores results as an active or a reference run.
SimDT	Data type the block uses during simulation.
SpecifiedDT	Data type the block specifies in its parameter dialog box, for example, the value of its **Output data type** parameter.

Column Label	Description
ProposedDT	Data type that the Fixed-Point Tool proposes.
Accept	Check box that enables you to selectively accept the Fixed-Point Tool's scaling proposal.
DesignMin	Minimum value the block specifies in its parameter dialog box, for example, the value of its **Output minimum** parameter.
SimMin	Minimum value that occurs during simulation.
ProposedMin	Minimum value that results from the data type the Fixed-Point Tool proposes.
DesignMax	Maximum value the block specifies in its parameter dialog box, for example, the value of its **Output maximum** parameter.
SimMax	Maximum value that occurs during simulation.
ProposedMax	Maximum value that results from the data type the Fixed-Point Tool proposes.
OverflowWraps	Number of overflows that wrap during simulation.
Saturations	Number of overflows that saturate during simulation.
DTGroup	Identification tag associated with objects that share data types.
DivByZero	Number of divide-by-zero instances that occur during simulation.
LogSignal	Check box that allows you to enable or disable signal logging for an object.

Column Label	Description
InitValueMin	Minimum initial value for a signal or parameter. Some model objects provide parameters that allow you to specify the initial values of their signals. For example, the Constant block includes a **Constant value** that initializes the block output signal. **Note** The Fixed-Point Tool uses this parameter when it proposes scaling.
InitValueMax	Maximum initial value for a signal or parameter. Some model objects provide parameters that allow you to specify the initial values of their signals. For example, the Constant block includes a **Constant value** that initializes the block output signal. **Note** The Fixed-Point Tool uses this parameter when it proposes scaling.
ModelRequiredMin	Minimum value of a parameter used during simulation. For example, the Lookup Table (n-D) uses the **Breakpoints** and **Table data** parameters to perform its lookup operation and generate output. In this example, the block uses more than one parameter so the Fixed-Point Tool sets **ModelRequiredMin** to the minimum of the minimum values of all these parameters.

Column Label	Description
	Note The Fixed-Point Tool uses this parameter when it proposes scaling.
ModelRequiredMax	Maximum value of a parameter used during simulation. For example, the Lookup Table (n-D) uses the **Breakpoints** and **Table data** parameters to perform its lookup operation and generate output. In this example, the block uses more than one parameter so the Fixed-Point Tool sets **ModelRequiredMax** to the maximum of the maximum values of all these parameters. **Note** The Fixed-Point Tool uses this parameter when it proposes scaling.

The following topics describe ways in which you can customize the **Contents** pane:

- "Changing Column Order and Width"
- "Sorting Rows by Column"
- "Hiding Columns"

Changing Column Order and Width

You can alter the order and width of columns that appear in the **Contents** pane as follows:

- To move a column, click and drag the head of a column to a new location among the column headers.

- To make a column wider or narrower, click and drag the right edge of a column header. If you double-click the right edge of a column header, the column width changes to fit its contents.

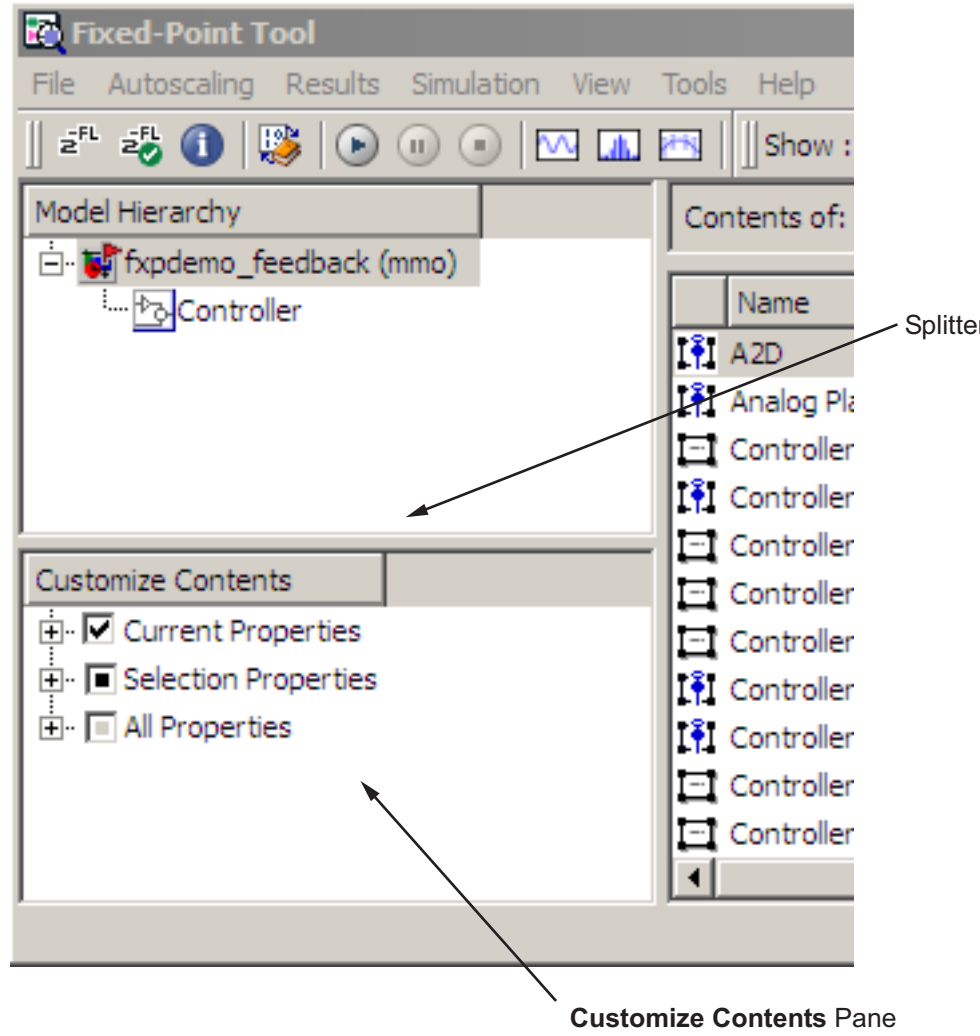
Sorting Rows by Column

By default, the **Contents** pane displays its contents in ascending order of the **Name** column. You can alter the order in which the **Contents** pane displays its rows as follows:

- To sort all the rows in ascending order of another column, click the head of that column.
- To change the order from ascending to descending, simply click again on the head of that column.

Hiding Columns

You can select the properties that the **Contents** pane displays or hides by using the **Customize Contents** pane. When visible, the pane appears in the lower-left corner of the Fixed-Point Tool window.



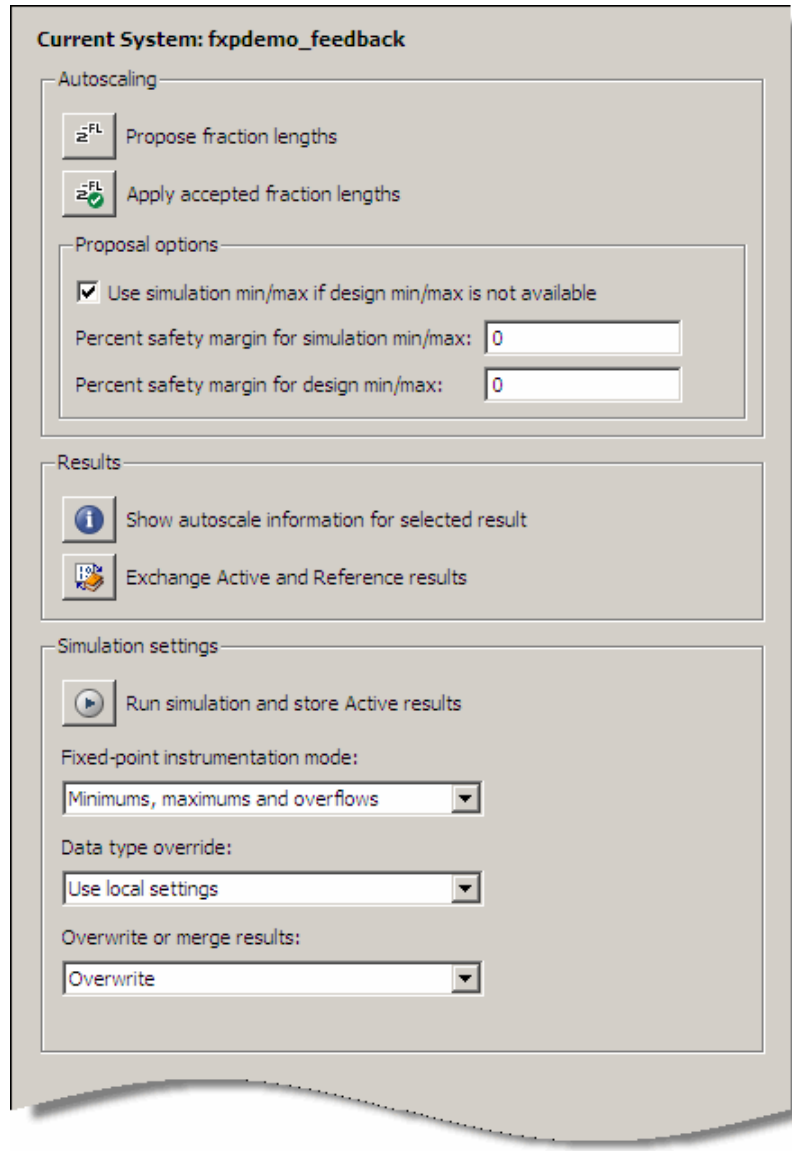
- To access the **Customize Contents** pane, from the Fixed-Point Tool **View** menu, select **Customize Contents**. A splitter divides the **Customize Contents** pane from the **Model Hierarchy** pane

above it. Drag the splitter up or down to adjust the relative size of the two panes.

- To hide properties from the **Contents** pane, in the **Customize Contents** pane, expand the `Current Properties` node and uncheck the properties that you do not want to appear.
- To display additional properties in the **Contents** pane, in the **Customize Contents** pane, expand the `All Properties` node and select the desired properties.

Dialog Pane

Use the **Dialog** pane to view and change properties associated with the object selected in the **Model Hierarchy** pane.


Current System: fxpdemo_feedback
Autoscaling
Propose fraction lengths
Apply accepted fraction lengths
Proposal options
Use simulation min/max if design min/max is not available
Percent safety margin for simulation min/max:
0
Percent safety margin for design min/max:
0
Results
Show autoscale information for selected result
Exchange Active and Reference results
Simulation settings
Run simulation and store Active results
Fixed-point instrumentation mode:
Minimums, maximums and overflows
Data type override:
Use local settings
Overwrite or merge results:
Overwrite

Tips

From the Fixed-Point Tool **View** menu, you can select **Dialog View** to hide the **Dialog** pane, making more room for displaying results.

Propose fraction lengths

Collect range data and proposes fraction lengths for model objects.

Action

Use this button to perform the first phase of the autoscaling procedure, in which the Fixed-Point Tool collects range data for model objects—either from design minimum and maximum values the objects specify explicitly, or from logged minimum and maximum values that occur during simulation. Based on these values, the tool proposes fraction lengths for blocks whose:

- **Lock output data type setting against changes by the fixed-point tools** or **Lock data type settings against changes by the fixed-point tools** check box is cleared.
- Data type specifies a generalized fixed-point number.

The Fixed-Point Tool lists its scaling proposals in the **Contents** pane. The tool alerts you to potential scaling issues for each object in the list by displaying a green, yellow, or red icon, as shown here:


The proposed scaling poses no issues for this object.


The proposed scaling poses potential issues for this object. Open the Autoscale Information dialog box to review these issues.


The proposed scaling will introduce data type errors if applied to this object. Open the Autoscale Information dialog box for details about how to resolve these issues.

Command-Line Alternative

No command line alternative available.

Apply accepted fraction lengths

Apply scaling proposals.

Action

Use this button to perform the second phase of the autoscaling procedure, in which the Fixed-Point Tool applies the scaling proposals to the objects whose **Accept** check box in the **Contents** pane is selected.

Command-Line Alternative

No command line alternative available.

Use simulation min/max if design min/max are not available

Propose fraction lengths based on simulation minimum and maximum values if no design minimum and maximum values are available.

Settings

Default: On

 On

Proposes fraction lengths based on simulation minimum and maximum values, but only for blocks that do not specify design minimum or maximum values using, for example, **Output minimum** and **Output maximum** parameters.

 Off

Ignores simulation minimum and maximum values when proposing fraction lengths.

Dependency

This parameter enables **Percent safety margin for simulation min/max**.

Command-Line Alternative

No command line alternative available.

Percent safety margin for simulation min/max

Specify multiplication safety factor for simulation minimum and maximum values.

Settings

Default: 0

The simulation minimum and maximum values are multiplied by the factor designated by this parameter, allowing you to specify a range different from that obtained from the simulation run. For example, a value of 55 specifies that a range *at least* 55 percent larger is desired. A value of -15 specifies that a range *up to* 15 percent smaller is acceptable.

Dependencies

This parameter is enabled by **Use simulation min/max if design min/max are not available**.

Before performing autoscaling, you must either specify design minimum and maximum values or run a simulation to collect simulation minimum and maximum data.

Command-Line Alternative

No command line alternative available.

Percent safety margin for design min/max

Specify multiplication safety factor for design minimum and maximum values.

Settings

Default: 0

The design minimum and maximum values are multiplied by the factor designated by this parameter. For example, a value of 55 specifies that a range *at least* 55 percent larger is desired. A value of -15 specifies that a range *up to* 15 percent smaller is acceptable.

Dependency

Before performing autoscaling, you must either specify design minimum and maximum values or run a simulation to collect simulation minimum and maximum data.

Command-Line Alternative

No command line alternative available.

Show autoscale information for selected result

Display the Autoscale Information dialog box for object selected in the **Contents** pane.

Action

Use this option to:

- Determine why a fraction length cannot be proposed, for example, fraction lengths cannot be proposed for Library Links or Mask Subsystems.
- Provide access to showing blocks with shared data types.
- Obtain more information on the proposal for the selected result.

Command-Line Alternative

No command line alternative available.

Exchange Active and Reference results

Swap results between active and reference run.

Action

Use this button to swap the results that the Fixed-Point Tool stores as an active run with those that it stores as a reference run.

Command-Line Alternative

No command-line alternative available.

Run simulation and store Active results

Simulate model and store results.

Action

Simulates the model and stores the results as active run, denoted by the `Active` label in the **Run** column of the **Contents** pane.

Tips

Using the **Run simulation and store Active results** button is the same as simulating the model using the **Start simulation** button in the Model Editor.

Command-Line Alternative

Command: `sim`

Fixed-point instrumentation mode

Control which objects log minimum, maximum and overflow data during simulation.

Settings

Default: `Use local settings`

`Use local settings`
: Logs data according to the value of this parameter set for each subsystem. Otherwise, settings for parent systems always override those of child systems.

`Minimums, maximums and overflows`
: Logs minimum value, maximum value, and overflow data for all blocks in the current system or subsystem during simulation.

`Overflows only`
: Logs only overflow data for all blocks in the current system or subsystem.

`Force off`
: Does not log data for any block in the current system or subsystem. Use this selection to work with models containing fixed-point enabled blocks if you do not have a Simulink Fixed Point license.

Tips

- The Fixed-Point Tool marks the system controlling the **Fixed-point instrumentation mode** with a red flag  .
- You cannot change the instrumentation mode for linked subsystems or referenced models.

Dependencies

The value of this parameter for parent systems controls min/max logging for all child subsystems, unless `Use local settings` is selected.

Command-Line Alternative

Parameter: `'MinMaxOverflowLogging'`
Type: string

Value: `'UseLocalSettings' | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
Default: `'UseLocalSettings'`

Data type override

Control data type override of objects that allow you to specify data types in their dialog boxes.

Settings

Default: `Use local settings`

The value of this parameter for parent systems controls data type override for all child subsystems, unless `Use local settings` is selected.

`Use local settings`
: Overrides data types according to the setting of this parameter for each subsystem. Otherwise, settings for parent systems override those of child systems.

`Scaled doubles`
: Overrides the output data type of all blocks in the current system or subsystem with doubles; however, the scaling and bias specified in the mask of each block is maintained.

`True doubles`
: Overrides the output data type of all blocks in the current system or subsystem with double-precision. The overridden values have no scaling or bias.

`True singles`
: Overrides the output data type of all blocks in the current system or subsystem with single-precision. The overridden values have no scaling or bias.

`Force off`
: No data type override is performed on any block in the current system or subsystem.

Tips

- Set this parameter to `True doubles` or `True singles` to work with models containing fixed-point enabled blocks if you do not have a Simulink Fixed Point license.

- You cannot change the **Data type override** setting on linked subsystems or referenced models.
- The Fixed-Point Tool marks the system controlling the **Data type override** mode with a green flag  .

Dependencies

- The following Simulink blocks allow you to set data types in their block masks, but ignore the **Data type override** setting:
  - Probe
  - Trigger
  - Width
- The Embedded MATLAB Function block ignores the **Data type override** parameter if it specifies `Scaled doubles`.

Command-Line Alternative

Parameter: `'DataTypeOverride'`
Type: string
Value: `'UseLocalSettings' | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
Default: `'UseLocalSettings'`

Overwrite or merge results

Control how simulation results are stored.

Settings

Default: `Overwrite`

`Overwrite`
: Clears all existing simulation results from the **Contents** pane before displaying new simulation results.

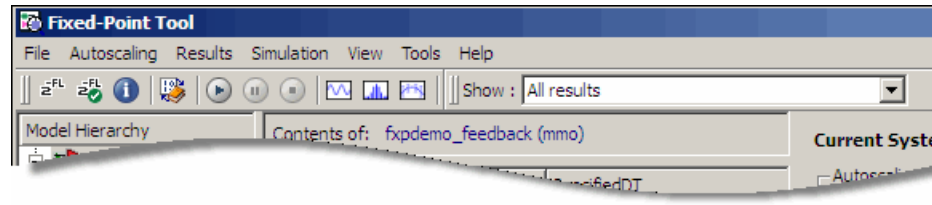
`Merge`
: Merges new simulation results with existing simulation results in the **Contents** pane.

Command-Line Alternative

Parameter: `'MinMaxOverflowArchiveMode'`
Type: string
Value: `'Overwrite' | 'Merge'`
Default: `'Overwrite'`

Main Toolbar

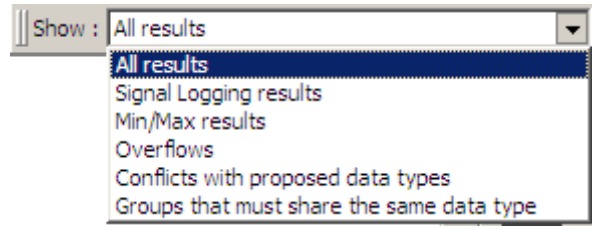
The Fixed-Point Tool's main toolbar appears near the top of the Fixed-Point Tool window under the Fixed-Point Tool's menu.



The toolbar contains the following buttons that execute commonly used Fixed-Point Tool commands:

Button	Usage
	Propose fraction lengths.
	Apply accepted fraction lengths.
	Display autoscale information.
	Exchange active results with reference results.
	Simulate a model and store results as active run.
	Pause a simulation.
	Stop a simulation.
	Create a time series plot.
	Create a histogram plot.
	Create a time series difference (A-R) plot.

The toolbar also contains the **Show** option:

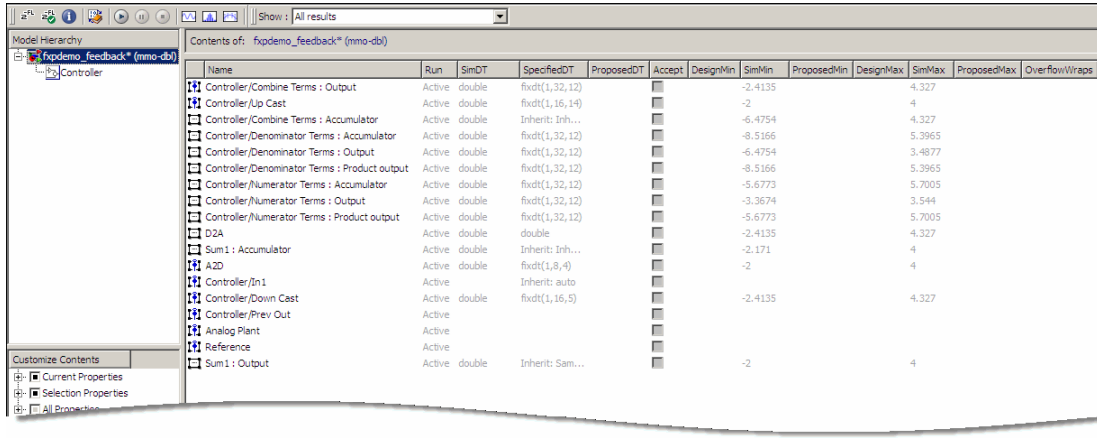


The **Show** option specifies the type of results to display in the **Contents** pane. The **Contents** pane displays information only after you simulate a system or propose fraction lengths. If there are no results that satisfy a particular filter option, the list will be blank.

Options include:

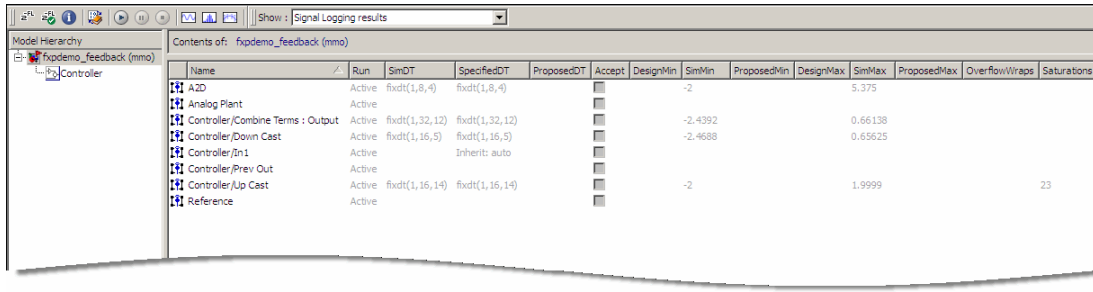
- All results

  Displays all results for the selected tree node.



- Signal Logging results

  For the selected tree node, displays blocks whose output ports have logged signal data. The Fixed-Point tool marks these blocks with the logged signal icon  .

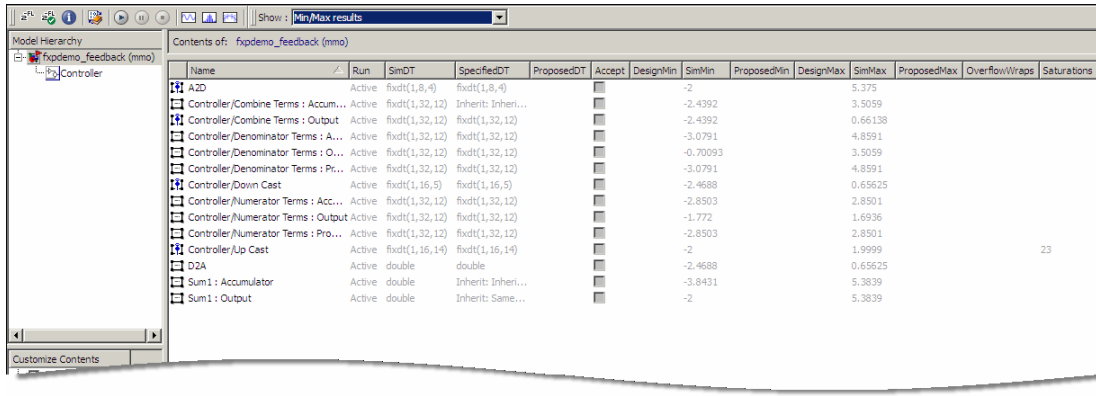


Note You can plot simulation results associated with logged signal data. For more information, see "Plot Interface" on page 4-98.

- Min/Max results

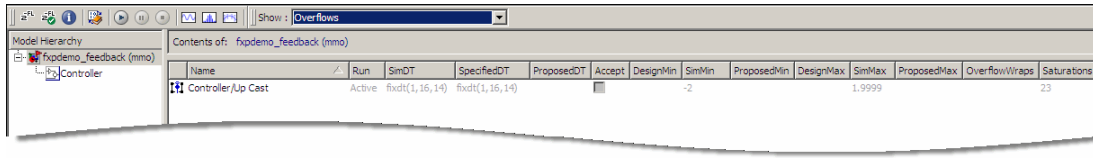
  For the selected tree node, displays blocks that record design Min/Max, simulation Min/Max, and overflow data.

  Prerequisites: **Fixed-point instrumentation mode** should not be set to `Force Off`.



- Overflows

  For the selected tree node, displays blocks that have non-zero overflows recorded. If a block has its **Saturate on integer overflow** option selected, overflow information appears in the **Saturations** column, otherwise it appears in the **OverflowWraps** column.



- Conflicts with proposed data types

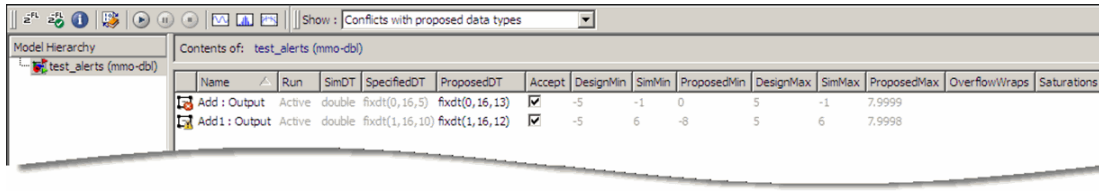
  For the selected tree node, displays results that have potential scaling issues.

  Prerequisites: This information is available only after you propose fraction lengths.

  The Fixed-Point Tool marks these results with a yellow or red icon, as shown here:

The proposed scaling poses potential issues for this object. Open the Autoscale Information dialog box to review these issues.

The proposed scaling will introduce data type errors if applied to this object. Open the Autoscale Information dialog box for details about how to resolve these issues.



- Groups that must share the same data type

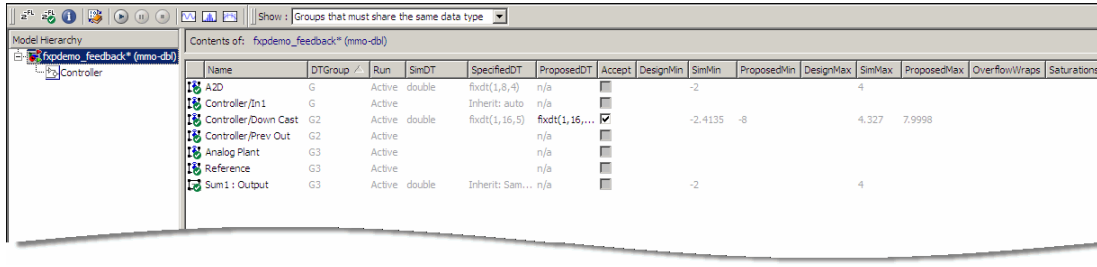
  For the selected tree node, displays blocks that must share the same data type because of data type propagation rules.

  Prerequisites: This information is available only after you propose fraction lengths.

  The Fixed-Point Tool allocates an identification tag to blocks that must share the same data type. This identification tag is displayed in the **DTGroup** column as follows:

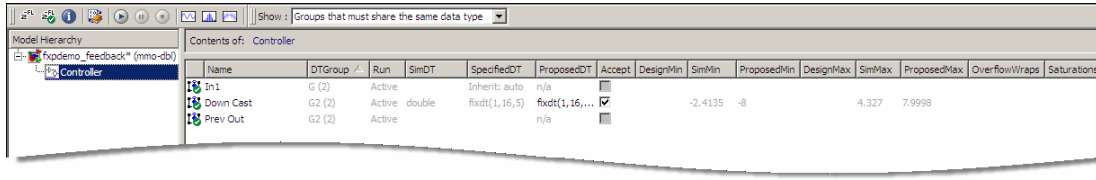
  - If the selected tree node is the model root

    All results for the model are listed. The **DTGroup** column is sorted by default so that you can easily view all blocks in a group.



- If the selected tree node is a subsystem

  The identification tags have a suffix that indicates the total number of results in each group. For example, `G2(2)` means group `G2` has 2 members. This information enables you to see how many members of a group belong to the selected subsystem and which groups share data types across subsystem boundaries.



Plot Interface

The Fixed-Point Tool provides plotting capabilities that enable you to plot signals for graphical analysis. The tool can access signal data that resides in the MATLAB workspace, allowing you to plot simulation results associated with:

- Scope blocks whose **Save data to workspace** parameter is selected
- To Workspace blocks
- Root-level Outport blocks, when the **Output** check box on the **Data Import/Export** pane of the Configuration Parameters dialog box is selected
- Logged signal data (see "Logging Signals" in the *Simulink User's Guide*)

The toolbar contains tools for interactive zooming, panning, rotating, querying, and editing plots. For more information, see "Figure Toolbars" in the MATLAB documentation.

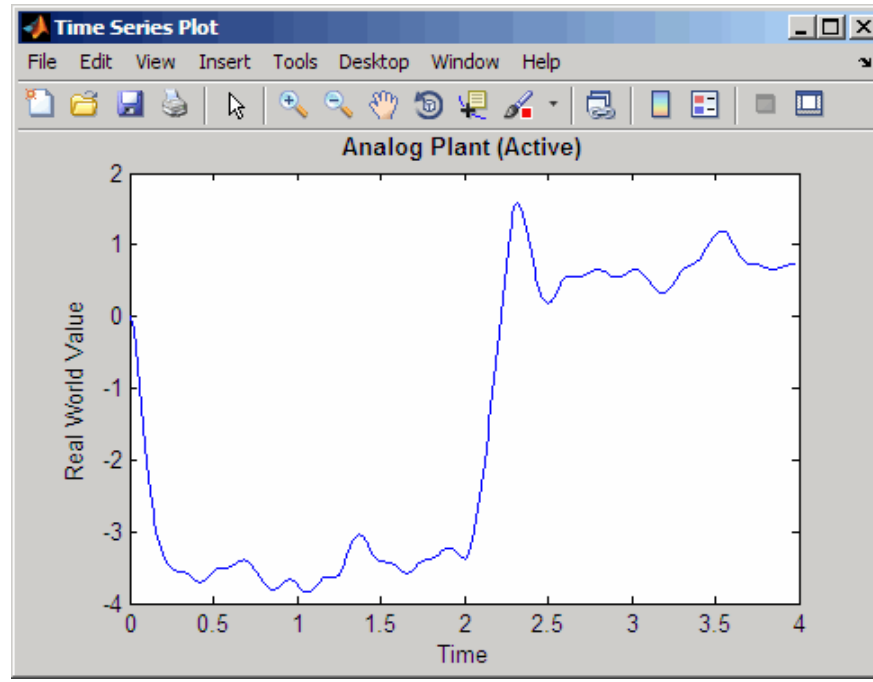
Tip The **Contents** pane of the Fixed-Point Tool displays an antenna icon next to items that you can plot.

You can create the following types of plots using the Fixed-Point Tool's interface:

- Time Series Plot
- Histogram Plot
- Time Series Difference (A-R) Plot

Time Series Plot

Plots data as a function of time.



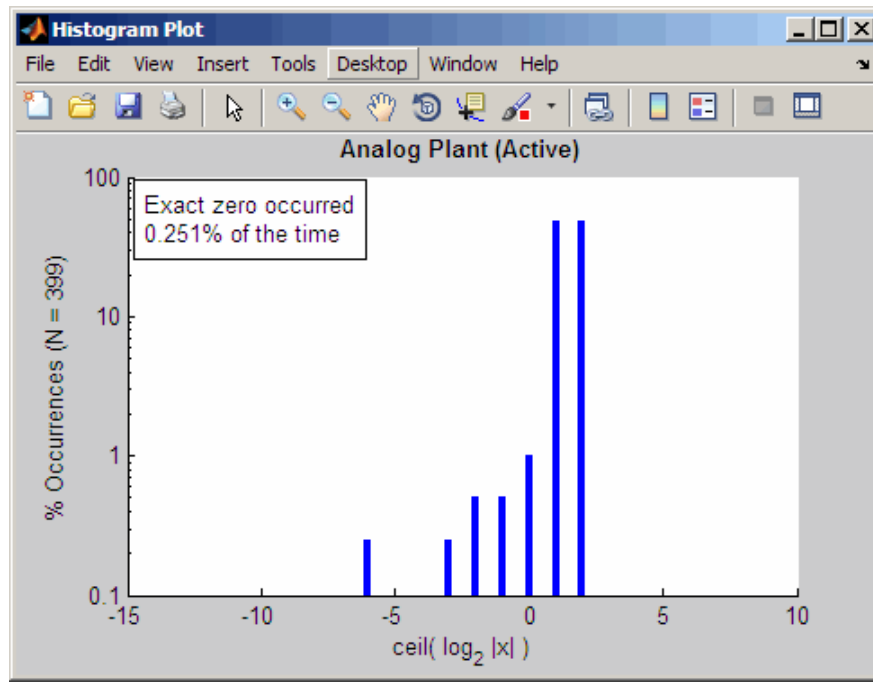
Histogram Plot

Helps you visualize the dynamic range of a signal.

This plot provides information about the:

- Total number of samples (N)
- The maximum number of bits needed to prevent overflow
- Number of times each bit has been used to represent the data (as a percentage of the total number of samples)
- Number of times that exact zero occurred (without the effect of quantization). This does not include the number of zeroes that occurred due to rounding.

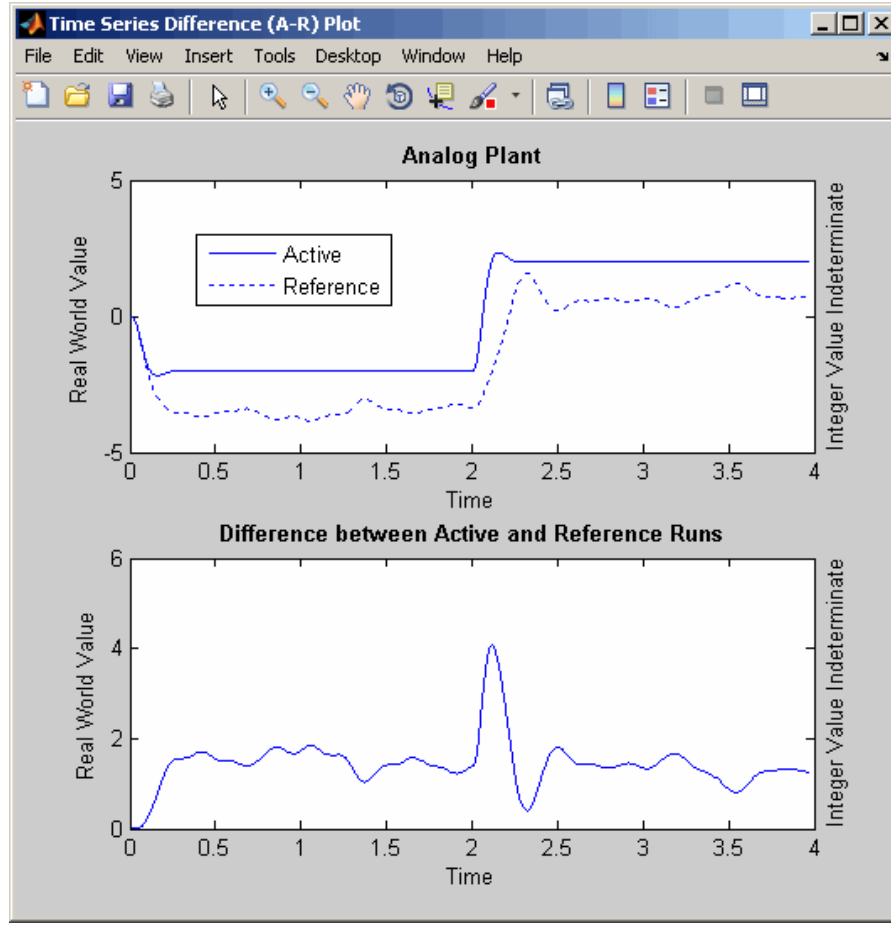
You can use this information to estimate the word size needed to represent the signal.



Time Series Difference (A-R) Plot

Plots both the active and reference versions of a signal on the upper axes, and plots the difference between the active and reference versions of that signal on the lower axes. For more details, see "Tutorial: Feedback Controller" in the Simulink Fixed Point documentation.

This plot also provides details of the real world value of the signal on the left y-axis, and the corresponding integer value on the right y-axis. If the Fixed-Point Tool cannot determine the scaling for a fixed-point data type, it cannot display the integer value information. In this case, it labels the right y-axis **Integer Value Indeterminate**. For more information on scaling, see "Scaling" in the Simulink Fixed Point documentation.



Synchronized Zooming on the Time Series Difference (A-R) Plot

By default, synchronized zooming is enabled for the Time Series Difference (A-R) plot. If you zoom in or out on a selected area on one plot, only the x-axis zooms on the other plot. To return to the original view at any time:

1 Right-click on either plot to open the context menu

2 Select **Reset to Original View**

Signal Logging Options

The Fixed-Point Tool provides options that allow you to control signal logging in a model (see "Logging Signals" in the *Simulink User's Guide*). Using these options, you can enable or disable logging for multiple signals simultaneously, based on signal attributes such as:

- The location of signals in a model hierarchy
- Whether or not signals have names

The Fixed-Point Tool does not control signal logging for referenced models or subsystems with library links. It ignores these objects when enabling or disabling signal logging in a model hierarchy.

You can control the signal logging of a Simulink subsystem, that is placed inside of a Stateflow Chart, from the subsystem parent node.

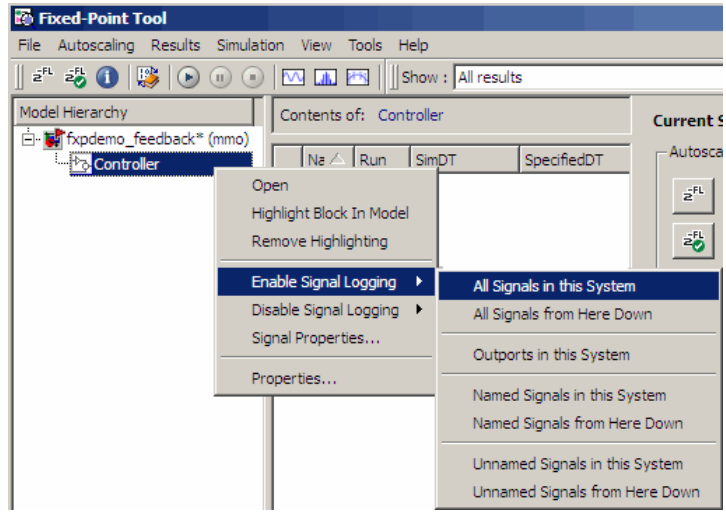
How to Access the Signal Logging Options in the Fixed-Point Tool

1 In the **Model Hierarchy** pane, right-click a node that represents either a model or a subsystem.

The Fixed-Point Tool displays a context menu for the selected node.

2 In the context menu, select either **Enable Signal Logging** or **Disable Signal Logging**.

The Fixed-Point Tool displays a submenu that lists a variety of signal logging options.



Signal Logging Options

Choose from the following signal logging options:

Select...	To Enable or Disable Signal Logging for...
All Signals in this System	All signals in the selected system
All Signals from Here Down	All signals in the selected system and its subsystems
Outports in this System	All Outport blocks in the selected system
Named Signals in this System	All signals in the selected system, whose **Signal name** parameter specifies a value
Named Signals from Here Down	All signals in the selected system and its subsystems, whose **Signal name** parameter specifies a value

Select...	To Enable or Disable Signal Logging for...
Unnamed Signals in this System	All signals in the selected system, whose **Signal name** parameter is empty
Unnamed Signals from Here Down	All signals in the selected system and its subsystems, whose **Signal name** parameter is empty

Examples

To learn how to use the tool, see "Tutorial: Feedback Controller" in the Simulink Fixed Point documentation.

See Also

autofixexp in the Simulink Fixed Point documentation.

gcb

Purpose

Get pathname of 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 handle of 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 pathname of 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 system and block parameter values

Syntax

```
paramValue = get_param(object, paramName)
paramValues = get_param(objectCellArray, paramName)
paramValue = get_param(objectHandle, paramName)
paramValue = get_param(0, paramName)
paramStruct = get_param(object, 'ObjectParameters')
paramCellArray = get_param(object, 'DialogParameters')
```

Description

`paramValue = get_param(object, paramName)` returns the value of the specified parameter. Some parameters are case-sensitive, and some are not. To prevent problems, treat all parameters as case-sensitive.

`paramValues = get_param(objectCellArray, paramName)` 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.

`paramValue = get_param(objectHandle, paramName)` returns the value of the specified parameter of the object whose handle is `objectHandle`.

`paramValue = get_param(0, paramName)` returns the current value of a Simulink session parameter or the default value of a model or block parameter.

`paramStruct = get_param(object, 'ObjectParameters')` returns a structure that describes `object`'s parameters.

`paramCellArray = get_param(object, 'DialogParameters')` returns a cell array containing the names of `object`'s dialog parameters.

Inputs

object

A model object for which `get_param` is to return a specified parameter value.

objectCellArray

A cell array of full path specifiers of objects for which `get_param` is to return the values of a specified parameter.

objectHandle

A handle to a model object for which `get_param` is to return a specified parameter value.

`paramName`

The name of a parameter for which `get_param` is to return a value or values.

Outputs

`paramCellArray`

A cell array containing the names of the dialog parameters of *`object`*.

`paramStruct`

A structure containing information about the parameters of *`object`*. Each field of the structure corresponds to the parameter with the same name as the field. Each field contains three subordinate fields: `Type`, `Enum` (if applicable), and `Attributes`. These fields respectively specify the parameter's data type, enumerated values (if applicable) , and attributes.

`paramValue`

The value of the parameter specified by *`paramName`*.

`paramValues`

A list containing the value of the parameter specified by *`paramName`* in each object specified in *`objectCellArray`*.

Examples

Return 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)
```

Display the block types of all blocks in the current system:

```
blks = find_system(gcs, 'Type', 'block');
listblks = get_param(blks, 'BlockType')

listblks =

    'SubSystem'
    'Inport'
    'Constant'
    'Gain'
    'Sum'
    'Outport'
```

Return the name of the currently selected block:

```
get_param(gcb, 'Name')
```

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'
```

Get the dialog parameters of a Sine Wave block:

```
p = get_param('untitled/Sine Wave', 'DialogParameters')
p =
          SineType: [1x1 struct]
        TimeSource: [1x1 struct]
         Amplitude: [1x1 struct]
              Bias: [1x1 struct]
         Frequency: [1x1 struct]
```

```
                    Phase: [1x1 struct]
                  Samples: [1x1 struct]
                   Offset: [1x1 struct]
               SampleTime: [1x1 struct]
          VectorParams1D: [1x1 struct]
```

See Also

`find_system | gcb | set_param`

How To

- "Associating User Data with Blocks"
- Chapter 8, "Model and Block Parameters"
- "Using MATLAB Commands to Change Workspace Data"

Purpose

Get model's active configuration set or configuration reference

Syntax

```
myConfigObj = getActiveConfigSet('model')
```

Arguments

model

The name of an open model, or `gcs` to specify the current model

Description

`getActiveConfigSet` returns the configuration set or configuration reference (configuration object) that is the active configuration object of '*model*'.

Example

The following example returns the active configuration object of the current model. The code is the same whether the object is a configuration set or configuration reference.

```
myConfigObj = getActiveConfigSet(gcs);
```

See Also

"Setting Up Configuration Sets", "Referencing Configuration Sets"

`attachConfigSet`, `attachConfigSetCopy`, `closeDialog`, `detachConfigSet`, `getConfigSet`, `getConfigSets`, `openDialog`, `setActiveConfigSet`

getCallbackAnnotation

Purpose

Get information about 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 model's configuration sets or configuration references

Syntax

```
myConfigObj = getConfigSet('model', 'configObjName')
```

Arguments

model
: The name of an open model, or `gcs` to specify the current model

configObjName
: The name of a configuration set (`Simulink.ConfigSet`) or configuration reference (`Simulink.ConfigSetRef`)

Description

`getConfigSet` returns the configuration set or configuration reference (configuration object) that is attached to *`model`* and is named *`configObjName`*. If no such object exists, an error occurs.

Example

The following example returns the configuration object that is named `DevConfig` and attached to the current model. The code is the same whether `DevConfig` is a configuration set or configuration reference.

```
myConfigObj = getConfigSet(gcs, 'DevConfig');
```

See Also

"Setting Up Configuration Sets", "Referencing Configuration Sets"

`attachConfigSet`, `attachConfigSetCopy`, `closeDialog`, `detachConfigSet`, `getActiveConfigSet`, `getConfigSets`, `openDialog`, `setActiveConfigSet`

getConfigSets

Purpose

Get names of all of model's configuration sets or configuration references

Syntax

```
myConfigObjNames = getConfigSets('model')
```

Arguments

model

The name of an open model, or `gcs` to specify the current model

Description

`getConfigSets` returns a cell array of strings specifying the names of all configuration sets and configuration references (configuration objects) attached to '*model*'.

Example

The following example obtains the names of the configuration objects attached to the current model.

```
myConfigObjNames = getConfigSets(gcs)
```

See Also

"Setting Up Configuration Sets", "Referencing Configuration Sets"

`attachConfigSet`, `attachConfigSetCopy`, `closeDialog`, `detachConfigSet`, `getActiveConfigSet`, `getConfigSet`, `openDialog`, `setActiveConfigSet`

Purpose

Get pathname of block or line

Syntax

```
path=getfullname(handle)
```

Description

`path=getfullname(handle)` returns the full pathname of the block or line specified by `handle`.

Examples

`getfullname(gcb)` returns the pathname of the block currently selected in the model editor's window.

The following code returns the pathname of the line currently selected in the model editor's window.

```
line = find_system(gcs, 'SearchDepth', 1, 'FindAll', 'on', ...
      'Type', 'line', 'Selected', 'on');
path = getfullname(line);
```

See Also

`gcb`, `find_system`

legacy_code

Purpose

Use Legacy Code Tool

Syntax

```
legacy_code('help')
specs = legacy_code('initialize')
legacy_code('sfcn_cmex_generate', specs)
legacy_code('compile', specs, compilerOptions)
legacy_code('generate_for_sim', specs, modelname)
legacy_code('slblock_generate', specs, modelname)
legacy_code('sfcn_tlc_generate', specs)
legacy_code('rtwmakecfg_generate', specs)
legacy_code('backward_compatibility')
```

Arguments

specs

A structure with the following fields:

Name the S-function

`SFunctionName` (Required) — A string specifying a name for the S-function to be generated by the Legacy Code Tool.

Define Legacy Code Tool Function Specifications

- `InitializeConditionsFcnSpec` — A nonempty string specifying a reentrant function that the S-function calls to initialize and reset states. You must declare this function by using tokens that Simulink software can interpret as explained in “Declaring Legacy Code Tool Function Specifications”.
- `OutputFcnSpec` — A nonempty string specifying the function that the S-function calls at each time step. You must declare this function by using tokens that Simulink software can interpret as explained in “Declaring Legacy Code Tool Function Specifications”.
- `StartFcnSpec` — A string specifying the function that the S-function calls when it begins execution. This function can access S-function parameter arguments only. You must declare this function by using tokens that Simulink software can

interpret as explained in “Declaring Legacy Code Tool Function Specifications”.

- `TerminateFcnSpec` — A string specifying the function that the S-function calls when it terminates execution. This function can access S-function parameter arguments only. You must declare this function by using tokens that Simulink software can interpret as explained in “Declaring Legacy Code Tool Function Specifications”.

Define Compilation Resources

- `HeaderFiles` — A cell array of strings specifying the file names of header files required for compilation.
- `SourceFiles` — A cell array of strings specifying source files required for compilation. You can specify the source files using absolute or relative path names.
- `HostLibFiles` — A cell array of strings specifying library files required for host compilation. You can specify the library files using absolute or relative path names.
- `TargetLibFiles` — A cell array of strings specifying library files required for target (that is, standalone) compilation. You can specify the library files using absolute or relative path names.
- `IncPaths` — A cell array of strings specifying directories containing header files. You can specify the directories using absolute or relative path names.
- `SrcPaths` — A cell array of strings specifying directories containing source files. You can specify the directories using absolute or relative path names.
- `LibPaths` — A cell array of strings specifying directories containing host and target library files. You can specify the directories using absolute or relative path names.

Specify a Sample Time

`SampleTime` — One of the following:

- `'inherited'` (default) — Sample time is inherited from the source block.
- `'parameterized'` — Sample time is represented as a tunable parameter. Generated code can access the parameter by calling MEX API functions, such as `mxGetPr` or `mxGetData`.
- Fixed — Sample time that you explicitly specify. For information on how to specify sample time, see "How to Specify the Sample Time".

If you specify this field, you must specify it last.

Define S-Function Options

`Options` — A structure that controls S-function options. The structure's fields include:

- `isMacro` — A logical value specifying whether the legacy code is a C macro. By default, the value is false (`0`).
- `isVolatile` — A logical value specifying the setting of the S-function `SS_OPTION_NONVOLATILE` option. By default, the value is true (`1`).
- `canBeCalledConditionally` — A logical value specifying the setting of the S-function `SS_OPTION_CAN_BE_CALLED_CONDITIONALLY` option. By default, the value is true (`1`).
- `useTlcWithAccel` — A logical value specifying the setting of the S-function `SS_OPTION_USE_TLC_WITH_ACCELERATOR` option. By default, the value is true (`1`).
- `language` — A string specifying either `'C'` or `'C++'` as the target language of the S-function that Legacy Code Tool will produce. By default, the value is `'C'`.

Note The Legacy Code Tool can interface with C++ functions, but not C++ objects. For a work around, see "Legacy Code Tool Limitations" in the Simulink documentation.

- `singleCPPMexFile` — A logical value that, if true (`1`), specifies that generated code:
  - Requires you to generate and manage an inlined S-function as only one file (`.cpp`) instead of two (`.c` and `.tlc`).
  - Maintains model code style (level of parentheses usage and preservation of operand order in expressions and condition expressions in if statements) as specified by model configuration parameters.

  By default, the value is false (`0`).

Limitations You cannot set the `singleCPPMexFile` field to true (`1`) if

- `Options.language='C++'`
- You use one of the following Simulink objects with the `IsAlias` property set to true (`1`):

  ```
  Simulink.Bus
  Simulink.AliasType
  Simulink.NumericType
  ```

- The Legacy Code Tool function specification includes a `void*` or `void**` to represent scalar work data for a state argument
- `- HeaderFiles` field of the Legacy Code Tool structure specifies multiple header files

modelname
: The name of a Simulink model into which Legacy Code Tool is to insert the masked S-function block generated when you specify `legacy_code` with the action string `'slblock_generate'`. If you omit this argument, the block appears in an empty model editor window.

Description

The `legacy_code` function creates a MATLAB structure for registering the specification for existing C or C++ code and the S-function being generated. In addition, the function can generate, compile and link, and create a masked block for the specified S-function. Other options include generating

- A TLC file for simulation in Accelerator mode or code generation
- An `rtwmakecfg.m` file that you can customize to specify dependent source and header files that reside in a different directory than that of the generated S-function

`legacy_code('help')` displays instructions for using Legacy Code Tool.

`specs` `= legacy_code('initialize')` initializes the Legacy Code Tool data structure, *`specs`*, which registers characteristics of existing C or C++ code and properties of the S-function that the Legacy Code Tool generates.

`legacy_code('sfcn_cmex_generate',` *`specs`*`)` generates an S-function source file as specified by the Legacy Code Tool data structure, *`specs`*.

`legacy_code('compile',` *`specs`*`,` *`compilerOptions`*`)` compiles and links the S-function generated by the Legacy Code Tool based on the data structure, *`specs`*, and any compiler options that you might specify. The following examples show how to specify no options, one option, and multiple options:

```
legacy_code('compile', s);
legacy_code('compile', s, '-DCOMPILE_VALUE1=1');
legacy_code('compile', s,...
```

```
        {'-DCOMPILE_VALUE1=1', '-DCOMPILE_VALUE2=2',...
'-DCOMPILE_VALUE3=3'});
```

`legacy_code('generate_for_sim', specs, modelname)` generates, compiles, and links the S-function in a single step. The function also generates a TLC file for accelerated simulations, if the `Options.useTlcWithAccel` field of the Legacy Code Tool data structure is set to 1.

`legacy_code('slblock_generate', specs, modelname)` generates a masked S-Function block for the S-function generated by the Legacy Code Tool based on the data structure, *`specs`*. The block appears in the Simulink model specified by `modelname`. If you omit `modelname`, the block appears in an empty model editor window.

`legacy_code('sfcn_tlc_generate', specs)` generates a TLC file for the S-function generated by the Legacy Code Tool based on the data structure, *`specs`*. This option is relevant if you want to

- Force Accelerator mode in Simulink software to use the TLC inlining code of the generated S-function. See the description of the `ssSetOptions` SimStruct function and `SS_OPTION_USE_TLC_WITH_ACCELERATOR` S-function option for more information.
- Use Real-Time Workshop software to generate code from your Simulink model. See "Automating the Generation of Files for Fully Inlined S-Functions Using Legacy Code Tool" for more information.

`legacy_code('rtwmakecfg_generate', specs)` generates an `rtwmakecfg.m` file for the S-function generated by the Legacy Code Tool based on the data structure, *`specs`*. This option is relevant only if you use Real-Time Workshop software to generate code from your Simulink model. See "Using the rtwmakecfg.m API to Customize Generated Makefiles" and "Automating the Generation of Files for Fully Inlined S-Functions Using Legacy Code Tool" in the Real-Time Workshop documentation for more information.

`legacy_code('backward_compatibility')` automatically updates syntax for using Legacy Code Tool, as made available from MATLAB Central in releases before R2006b, to the supported syntax described in this reference page and in "Integrating Existing C Functions into Simulink Models with the Legacy Code Tool" in *Writing S-Functions*.

See Also

- "Integrating Existing C Functions into Simulink Models with the Legacy Code Tool" in the Writing S-Functions documentation
- "Automating the Generation of Files for Fully Inlined S-Functions Using Legacy Code Tool" in the Real-Time Workshop documentation

Purpose

Get information about library blocks referenced by model

Syntax

```
libdata = libinfo(system)
libdata = libinfo(system, constraint1, value1, ...,
   constraintN, valueN))
```

Description

`libdata = libinfo(system)` returns information about library blocks referenced by *`system`* and all the systems underneath it.

`libdata = libinfo(system, constraint1, value1, ..., constraintN, valueN))` restricts the search as indicated by the search constraint(s) *`c1`*, *`v1`*, ..., *`cn`*, *`vn`*.

Inputs

`system`

The system to search recursively for library blocks.

`constraint1`, *`value1`* `...`

One or more pairs, each consisting of a search constraint followed by a constraint value. You can specify any of the search constraints that you can use with `find_system`.

Output

`libinfo`

An array of structures that describes each library block referenced by *`system`*. 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

See Also `find_system`

How To

- "Working with Block Libraries"

Purpose

Extract continuous- or discrete-time linear state-space model of system around operating point

Syntax

```
argout = linmod('sys');
argout = linmod('sys',x,u);
argout = linmod('sys', x, u, para);
argout = linmod('sys', x, u, 'v5');
argout = linmod('sys', x, u, para, 'v5');
argout = linmod('sys', x, u, para, xpert, upert, 'v5');

argout = dlinmod('sys', Ts, x, u);
argout = dlinmod('sys',Ts, x, u, para, 'v5');
argout = dlinmod('sys',Ts, x, u, para, xpert, upert, 'v5');

argout = linmod2('sys', x, u);
argout = linmod2('sys', x, u, para);

argout = linmodv5('sys');
argout = linmodv5('sys',x,u);
argout = linmodv5('sys', x, u, para);
argout = linmod('sys', x, u, para, xpert, upert);
```

Arguments

`sys`	The name of the Simulink system from which the linear model is to be extracted.
`x` and `u`	The state and the input vectors. If specified, they set the operating point at which the linear model is to be extracted. When a model has model references using the Model block, you must use the Simulink structure format to specify `x`. To extract the `x` structure from the model, use the following command:

```
x = Simulink.BlockDiagram.getInitialState('sys');
```

You can then change the operating point values within this structure by editing `x.signals.values`.

If the state contains different data types (for example, `'double'` and `'uint8'`), then you cannot use a vector to specify this state. You must use a structure instead. In addition, you can only specify the state as vector if its data type is `'double'`.

`Ts` — Sample time of the discrete-time linearized model

`'v5'` — An optional argument that invokes the perturbation algorithm created prior to MATLAB 5.3. Invoking this optional argument is equivalent to calling `linmodv5`.

`para` — A three-element vector of optional arguments:

- `para(1)` — Perturbation value of delta, the value used to perform the perturbation of the states and the inputs of the model. This is valid for linearizations using the `'v5'` flag. The default value is 1e-05.
- `para(2)` — Linearization time. For blocks that are functions of time, this parameter can be set with a nonnegative value of `t` giving the time at which Simulink software evaluates the blocks when linearizing a model. The default value is 0.
- `para(3)` — Set `para(3)=1` to remove extra states associated with blocks that have no path from input to output. The default value is 0.

`xpert` and `upert`

The perturbation values used to perform the perturbation of all the states and inputs of the model. The default values are

```
xpert = para(1) + 1e-3*para(1)*abs(x)
upert = para(1) + 1e-3*para(1)*abs(u)
```

When a model has model references using the Model block, you must use the Simulink structure format to specify `xpert`. To extract the `xpert` structure, use the following command:

```
xpert = Simulink.BlockDiagram.getInitialState('sys');
```

You can then change the perturbation values within this structure by editing `xpert.signals.values`.

The perturbation input arguments are only available when invoking the perturbation algorithm created prior to MATLAB 5.3, either by calling `linmodv5` or specifying the `'v5'` input argument to `linmod`.

`argout`

`linmod`, `dlinmod`, and `linmod2` return state-space representations if you specify the output (left-hand) side of the equation as follows:

- `[A,B,C,D] = linmod('sys', x, u)` obtains the linearized model of sys around an operating point with the specified state variables x and the input u. If you omit x and u, the default values are zero.

`linmod` and `dlinmod` both also return transfer function and MATLAB data structure representations of the linearized system, depending on how you specify the output (left-hand) side of the equation. Using `linmod` as an example:

- `[num, den] = linmod('sys', x, u)` returns the linearized model in transfer function form.
- `sys_struc = linmod('sys', x, u)` returns a structure that contains the linearized model, including state names, input and output names, and information about the operating point.

Description

`linmod` and `dlinmod` compute a linear state space model by linearizing each block in a model individually. `linmod2` computes a linear state-space model by perturbing the model inputs and model states, and uses an advanced algorithm to reduce truncation error. `linmodv5` computes a linear state space model using the full model perturbation algorithm created prior to MATLAB 5.3.

`linmod` obtains linear models from systems of ordinary differential equations described as Simulink models. Inputs and outputs are denoted in Simulink block diagrams using Inport and Outport blocks.

The default algorithm uses preprogrammed analytic block Jacobians for most blocks which should result in more accurate linearization than numerical perturbation of block inputs and states. A list of blocks that have preprogrammed analytic Jacobians is available in the Simulink Control Design documentation along with a discussion of the block-by-block analytic algorithm for linearization. If you do not have Simulink Control Design software installed, you can access the documentation on The MathWorks™ Web site at http://www.mathworks.com/access/helpdesk/help/toolbox/slcontrol/.

The default algorithm also allows for special treatment of problematic blocks such as the Transport Delay and the Quantizer. See the mask dialog of these blocks for more information and options.

Discrete-Time System Linearization

The function `dlinmod` can linearize discrete, multirate, and hybrid continuous and discrete systems at any given sampling time. Use the

same calling syntax for `dlinmod` as for `linmod`, but insert the sample time at which to perform the linearization as the second argument. For example,

```
[Ad,Bd,Cd,Dd] = dlinmod('sys', Ts, x, u);
```

produces a discrete state-space model at the sampling time `Ts` and the operating point given by the state vector `x` and input vector `u`. To obtain a continuous model approximation of a discrete system, set `Ts` to `0`.

For systems composed of linear, multirate, discrete, and continuous blocks, `dlinmod` produces linear models having identical frequency and time responses (for constant inputs) at the converted sampling time `Ts`, provided that

- `Ts` is an integer multiple of all the sampling times in the system.
- The system is stable.

For systems that do not meet the first condition, in general the linearization is a time-varying system, which cannot be represented with the [*A*,*B*,*C*,*D*] state-space model that `dlinmod` returns.

Computing the eigenvalues of the linearized matrix `Ad` provides an indication of the stability of the system. The system is stable if `Ts>0` and the eigenvalues are within the unit circle, as determined by this statement:

```
all(abs(eig(Ad))) < 1
```

Likewise, the system is stable if `Ts = 0` and the eigenvalues are in the left half plane, as determined by this statement:

```
all(real(eig(Ad))) < 0
```

When the system is unstable and the sample time is not an integer multiple of the other sampling times, `dlinmod` produces `Ad` and `Bd` matrices, which can be complex. The eigenvalues of the `Ad` matrix in this case still, however, provide a good indication of stability.

You can use `dlinmod` to convert the sample times of a system to other values or to convert a linear discrete system to a continuous system or vice versa.

You can find the frequency response of a continuous or discrete system by using the `bode` command.

Notes

By default, the system time is set to zero. For systems that are dependent on time, you can set the variable `para` to a two-element vector, where the second element is used to set the value of `t` at which to obtain the linear model.

The ordering of the states from the nonlinear model to the linear model is maintained. For Simulink systems, a string variable that contains the block name associated with each state can be obtained using

```
[sizes,x0,xstring] = sys
```

where `xstring` is a vector of strings whose *i*th row is the block name associated with the `i`th state. Inputs and outputs are numbered sequentially on the diagram.

For single-input multi-output systems, you can convert to transfer function form using the routine `ss2tf` or to zero-pole form using `ss2zp`. You can also convert the linearized models to LTI objects using `ss`. This function produces an LTI object in state-space form that can be further converted to transfer function or zero-pole-gain form using `tf` or `zpk`.

The default algorithms in `linmod` and `dlinmod` handle Transport Delay blocks by replacing the linearization of the blocks with a Pade approximation. For the `'v5'` algorithm, linearization of a model that contains Derivative or Transport Delay blocks can be troublesome. For more information, see “Linearizing Models” in *Simulink User’s Guide*.

Purpose

Invisibly load 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`

model

Purpose

Execute particular phase of simulation of 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 software 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 states and outputs of a model, 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. • `'update'` computes the next values of the model's discrete states.

Examples

This command executes the compilation phase of the `vdp` model that comes with Simulink software.

```
vdp([], [], [], 'compile')
```

The following command terminates the simulation initiated in the previous example.

```
vdp([], [], [], 'term')
```

- `'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 software to terminate simulation of the model.

Note You must always terminate simulation of the model by invoking the model command with the `'term'` command. Simulink software does not let you close the model until you have terminated the simulation.

See Also

`sim`

Purpose

Open Model Advisor

Syntax

```
modeladvisor('model')
```

Arguments

model
A string specifying the name or handle to the model or subsystem.

Description

`modeladvisor(model)` opens the Model Advisor on the model or subsystem specified by model. 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.

See Also

"Consulting the Model Advisor"

Purpose

Create 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 8, "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

Convert number to nearest value representable by specified fixed-point data type

Syntax

```
outValue = num2fixpt(OrigValue, FixPtDataType, FixPtScaling,
    RndMeth, DoSatur)
```

Description

`num2fixpt(OrigValue, FixPtDataType, FixPtScaling, RndMeth, DoSatur)` returns the result of converting `OrigValue` to the nearest value representable by the fixed-point data type `FixPtDataType`. Both `OrigValue` and `outValue` are of data type `double`. As illustrated in the example that follows, you can use `num2fixpt` to investigate quantization error that might result from converting a number to a fixed-point data type. The arguments of `num2fixpt` include:

`OrigValue`	Value to be converted to a fixed-point representation. Must be specified using a `double` data type.
`FixPtDataType`	The fixed-point data type used to convert `OrigValue`.
`FixPtScaling`	Scaling of the output in either Slope or [Slope Bias] format. If `FixPtDataType` does not specify a generalized fixed-point data type using the `sfix` or `ufix` command, `FixPtScaling` is ignored.

`RndMeth`	Rounding technique used if the fixed-point data type lacks the precision to represent `OrigValue`. If `FixPtDataType` specifies a floating-point data type using the `float` command, `RndMeth` is ignored. Valid values are `Zero`, `Nearest`, `Ceiling`, or `Floor` (the default).
`DoSatur`	Indicates whether the output should be saturated to the minimum or maximum representable value upon underflow or overflow. If `FixPtDataType` specifies a floating-point data type using the `float` command, `DoSatur` is ignored. Valid values are `on` or `off` (the default).

Examples

Suppose you wish to investigate the quantization effect associated with representing the real-world value 9.875 as a signed, 8-bit fixed-point number. The command

```
num2fixpt(9.875, sfix(8), 2^-1)

ans =

   9.50000000000000
```

reveals that a slope of `2^-1` results in a quantization error of 0.375. The command

```
num2fixpt(9.875, sfix(8), 2^-2)

ans =

   9.75000000000000
```

demonstrates that a slope of `2^-2` reduces the quantization error to 0.125. But a slope of `2^-3`, as used in the command

```
num2fixpt(9.875, sfix(8), 2^-3)

ans =

   9.87500000000000
```

eliminates the quantization error entirely.

See Also

`fixptbestexp`, `fixptbestprec`

Purpose

Open Simulink system window or 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', 'destsys', 'replace')
open_system('sys', 'destsys', 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 opened system (for example, `engine/Combustion`). On UNIX systems, the fully qualified pathname of a model can start with a tilde (~), signifying your home directory.

Note If you use the open_system command to open a subsystem without first opening or loading the model that contains the subsystem, Simulink gives an error. Use `load_system` to load a model before you use `open_system` to open a subsystem in that model.

`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', 'destsys', 'replace')` replaces the window of the previously opened system `destsys` with the window of the subsystem `sys` opened by this command. The location of the new window is determined by the location of the destination system `destsys` while the size of the window will match that used by `sys`.

`open_system('sys', 'destsys', 'reuse')` reuses the window of the previously opened system `destsys` to display the contents of the subsystem `sys` opened by this command. In this case, `sys` will be scaled to fit within the window size determined by the destination system `destsys`.

Note Use the MATLAB `sprintf` command to insert carriage return or line feed characters into paths passed to the open_system command. For example, the path to the Aircraft Dynamics Model subsystem of the f14 demo model contains line feeds. To open the subsystem, enter the following command at the MATLAB command line:

```
open_system(['f14/Aircraft' sprintf('\n') 'Dynamics' sprintf('\n') 'Model'])
```

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');
```

This command opens `f14` into the `f14/Controller` window using reuse mode.

```
open_system('f14','f14/Controller','reuse');
```

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.

This command opens `f14` and `vdp` with a vectorized operation.

```
open_system( {'f14','vdp'} );
```

Open a subsystem after loading a model.

```
load_system('f14');
open_system('f14/Controller');
```

See Also

`close_system`, `load_system`, `new_system`, `save_system`

openDialog

Purpose

Open configuration parameters dialog

Syntax

```
openDialog(configObj)
```

Arguments

configObj
: A configuration set (`Simulink.ConfigSet`) or configuration reference (`Simulink.ConfigSetRef`)

Description

`openDialog` opens a configuration parameters dialog box. If *`configObj`* is a configuration set, the dialog box displays the configuration set. If *`configObj`* is a configuration reference, the dialog box displays the referenced configuration set, or generates an error if the reference does not specify a valid configuration set. If the dialog box is already open, its window becomes selected.

Example

The following example opens a configuration parameters dialog box that shows the current parameters for the current model. The parameter values derive from the active configuration set or configuration reference (configuration object). The code is the same in either case; the only difference is which type of configuration object is currently active.

```
myConfigObj = getActiveConfigSet(gcs);
openDialog(myConfigObj);
```

See Also

"Setting Up Configuration Sets", "Referencing Configuration Sets"

`attachConfigSet`, `attachConfigSetCopy`, `closeDialog`, `detachConfigSet`, `getActiveConfigSet`, `getConfigSet`, `getConfigSets`, `setActiveConfigSet`

Purpose

Replace blocks in Simulink model

Syntax

```
replace_block('sys', 'old_blk', 'new_blk')
replace_block('sys', 'Parameter', 'value', ..., 'blk')
```

Description

`replace_block('sys', 'old_blk', 'new_blk')` replaces all blocks in `'sys'` having the block or mask type `'old_blk'` with `'new_blk'`.

- If `'new_blk'` is a Simulink built-in block, only the block name is necessary.
- If `'old_blk'` or `'new_blk'` is in another system, its full block pathname is required.
- If `'noprompt'` is omitted, Simulink software 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. For information on block parameters, see Chapter 8, “Model and Block Parameters”

Note Because it may be difficult to undo the changes this command makes, it is a good idea to save your Simulink model 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 software 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 software 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')
```

This command replaces the `Small_Wheel` subsystem in the `wheel_analysis` model with the `Large_Wheel` subsystem from the `wheels` library.

```
replace_block('wheel_analysis','Name','Small_Wheel','wheels/Large_Wheel')
```

See Also

`find_system`, `set_param`

Purpose

Save Simulink system

Syntax

```
save_system
save_system('sys')
save_system('sys', 'newsysname')
save_system('sys', 'newsysname', 'BreakAllLinks', true)
save_system('sys', 'newsysname', 'BreakUserLinks', true)
save_system('sys', 'newsysname',
'SaveModelWorkspace', true)
save_system('sys', 'newsysname', 'ErrorIfShadowed', true)
save_system('sys', 'newsysname', 'SaveAsVersion',
'version')
save_system('sys', 'newsysname',
'OverWriteIfChangedOnDisk',
   true)
save_system('sys', 'exported_file_name.xml','ExportToXML',
   true)
save_system('sys', 'newsysname',
'SaveModelWorkspace', true,
   'BreakUserLinks', true, 'OverwriteIfChangedOnDisk',
true)
```

Description

`save_system` saves the current top-level system to a file using the current system name.

`save_system('sys')` saves the top-level system that you specify in `sys` to a file using the current system name. The system must be open. *'sys'* can be a string, a cell array of strings, a numeric handle, or an array of numeric handles. If you specify any options they apply to all the systems that you save.

`save_system('sys', 'newsysname')` saves the top-level system that you specify to a file using the new system name *newsysname*. The system must be open. *'newsysname'* can be a file name, in which case Simulink saves the system in the working folder, or a fully qualified path name. On UNIX systems, the fully qualified path name can start with a tilde (~), signifying your home folder.

`save_system(*'sys'*, *'newsysname'*, 'BreakAllLinks', true)` replaces links to library blocks with copies of the library blocks in the saved file.

`save_system(*'sys'*, *'newsysname'*, 'BreakUserLinks', true)` replaces links to user-defined library blocks with copies of the library blocks in the saved file.

`save_system(*'sys'*, *'newsysname'*, 'SaveModelWorkspace', true)` saves the system using a new name and, if the workspace **Data source** is a MAT-file, saves the contents of the model workspace. If the data source is not a MAT-file, `save_system` does not save the workspace. See "Model Workspace Dialog Box".

`save_system(*'sys'*, *'newsysname'*, 'ErrorIfShadowed', true)` generates an error if the new name already exists on the MATLAB path or workspace.

`save_system(*'sys'*, *'newsysname'*, 'SaveAsVersion', '*version*')` saves the system in a form that the Simulink version can load.

`save_system(*'sys'*, *'newsysname'*, 'OverWriteIfChangedOnDisk', true)` allows you to overwrite the system on disk.

`save_system(*'sys'*, '*exported_file_name*.xml','ExportToXML', true)` exports the system to a file in a simple XML format. Do not use `ExportToXML` with any other `save_system` options.

`save_system(*'sys'*, *'newsysname'*, 'SaveModelWorkspace', true, 'BreakUserLinks', true, 'OverwriteIfChangedOnDisk', true)` combines additional arguments as name-value pairs, in any order.

`save_system` can save only entire systems. To save a subsystem, use the `Simulink.SubSystem.copyContentsToBlockDiagram` function to copy the subsystem contents to a new block diagram and then save it using `save_system`. See `Simulink.SubSystem.copyContentsToBlockDiagram`.

If you set the `UpdateHistory` property of the model to `UpdateHistoryWhenSave`, you see the following behavior:

- When you save interactively, you see a dialog prompting for a comment to include in the model history.
- When you save using `save_system`, you do not see a prompt for a comment. `save_system` reuses the previous comment, unless you set `'ModifiedComment'` before saving:

```
set_param(mymodel,'ModifiedComment',mycomment)
```

Inputs

`'sys'`

Top-level system to save.

The system must be open. *`'sys'`* can be a string, a cell array of strings, a numeric handle, or an array of numeric handles.

`'newsysname'`

New system name. *`'newsysname'`* can be empty (`[]`), in which case the current name is used. If *`'sys'`* refers to more than one block diagram, *`'newsysname'`* must be a cell array of new names.

This command displays an error if you enter any of the following as the new system name:

- A MATLAB keyword
- `'simulink'`
- More than 63 characters

`BreakAllLinks`

Logical value that indicates whether the function replaces links to library blocks with copies of the library blocks in the saved file. The `'BreakAllLinks'` option affects any linked block, including user-defined and Simulink library blocks. Also accepts `on` or `off`.

```
true
```

`false` (default)

Note

The `'BreakAllLinks'` option can result in compatibility issues when upgrading to newer versions of Simulink software. For example:

- Any masks on top of library links to Simulink S-functions will not upgrade to the new version of the S-function.
- Any library links to masked subsystems in a Simulink library will not upgrade to the new subsystem behavior.
- Any broken links prevent the automatic library forwarding mechanism from upgrading the link.

If you have saved a model with broken links, use the `Check model, local libraries, and referenced models for known upgrade issues` option in the Model Advisor to scan the model for out-of-date blocks. You can then use the `slupdate` command to upgrade the Simulink blocks to their current versions. Subsequently running the Model Advisor lists any remaining third-party library and optional Simulink blockset blocks that are still out of date and need manual upgrading.

`BreakUserLinks`

Logical value that indicates whether the function replaces links to user-defined library blocks with copies of the library blocks in the saved file. Also accepts `on` or `off`.

`true`
`false` (default)

Default: false

`ErrorIfShadowed`

Logical value that indicates whether the function generates an error if the new name already exists on the MATLAB path or workspace. Also accepts `on` or `off`.

`true`
`false` (default)

`ExportToXML`

Logical value that indicates whether the function exports the specified block diagram to a file in a simple XML format. Specify the full name of the file, including an extension. The block diagram in memory does not change and no callbacks execute. Do not use this option with any other `save_system` options. Also accepts `on` or `off`.

`true`
`false` (default)

`SaveAsVersion`

MATLAB version name, that specifies the version of Simulink software to save the system for. Valid values include: R12, R12P1, R13, R13SP1, R14, R14SP1, R14SP2, R14SP3, R2006A, etc. These version names are not case sensitive. If the system to be saved contains blocks not supported by the specified Simulink software version, the command replaces the unsupported blocks with empty masked subsystem blocks colored yellow. As a result, the converted system may generate incorrect results.

Default: Your current version

`OverwriteIfChangedOnDisk`

Logical value that indicates whether to overwrite the system on disk (`true`). To save the model regardless of whether the file has been changed on disk, supply the `OverwriteIfChangedOnDisk` option with value `true`.

If the file has changed on disk since the model was loaded, `save_system` displays an error to prevent the changes on disk from being overwritten, unless you use the `OverwriteIfChangedOnDisk` option set to `true`. Also accepts `on` or `off`.

`true`
`false` (default)

You can control whether `save_system` displays an error if the file has changed on disk by using the **Saving the model** option in the **Model File Change Notification** section of the Simulink Preferences dialog box. This preference is on by default.

`SaveModelWorkspace`

Logical value that indicates whether the function saves the contents of the model workspace. The model workspace `DataSource` must be a MAT-file. Also accepts `on` or `off`.

`true`
`false` (default)

Outputs

`filename`

`save_system` returns the full name of the file that you saved, as a string. If you saved multiple files, the return value is a cell array of strings.

Examples

The following examples assume prerequisites such as: you have loaded a model and the folder where you want to save is writeable.

Save the current system.

```
save_system
```

Save the `vdp` system with the name `vdp`.

```
save_system('vdp')
```

Save the `vdp` system to a file with the name `'myvdp'`.

```
save_system('vdp', 'myvdp')
```

Save the `vdp` system to another folder.

```
save_system('vdp', 'C:\TMP\vdp.mdl')
```

Save the `vdp` system to a file with the name `'myvdp'` and replace links to library blocks with copies of the library blocks in the saved file.

```
save_system('vdp','myvdp','BreakAllLinks', true)
```

Save the current model (with its current name), and break any library links in it:

```
save_system('mymodel,'mymodel','BreakAllLinks',true)
```

or

```
save_system('mymodel,[],'BreakAllLinks',true)
```

Save the current model with a new name, but display an error (instead of saving) if something with this name already exists on the MATLAB path:

```
save_system('mymodel','mynewmodel','ErrorIfShadowed',true)
```

Prevent saving the `vdp` system with a new name if something with this name already exists on the MATLAB path. In this case `save_system` displays an error (instead of saving) because `'max'` is the name of a MATLAB function.

```
save_system('vdp', 'max', 'ErrorIfShadowed', true)
```

Save 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','SaveAsVersion','R13SP1')
```

Save the current model with a new name, save the model workspace, break any library links, and overwrite if the file has changed on disk:

```
save_system('mymodel, 'mynewmodel', 'SaveModelWorkspace',
true, 'BreakAllLinks',true, 'OverwriteIfChangedOnDisk', true)
```

Return the full path name of the file that you saved, as a string. If you saved multiple files, the return value is a cell array of strings.

```
filename = save_system('mymodel')
```

See Also

`close_system` | `new_system` | `open_system`

Purpose

Set Simulink system and block parameter values

Syntax

```
set_param(object, paramName1, Value1, ..., paramNameN,
   ValueN)
```

Description

`set_param(object, paramName1, Value1, ..., paramNameN, ValueN)` sets the specified parameter(s) to the specified value(s). Case is ignored for parameter names. Value strings are case-sensitive.

Most block parameter values must be specified as strings. Two exceptions are the `Position` and `UserData` parameters, which are common to all blocks. All parameters that correspond to dialog box entries have string values.

To change block parameter values in the workspace and then update the block diagram with the changes:

1. Use `set_param` to make the changes at the command prompt.
2. Make the model window the current window.
3. Select **Edit > Update Diagram**.

When you set multiple parameters on the same model or block, use a single `set_param` command with multiple pairs of arguments, rather than multiple `set_param` commands. This technique is more efficient because using a single call requires evaluating parameters only once, and more robust because it prevents dependency errors by setting all parameter values before checking the legality of the results.

Inputs

`object`

The model or block for which `set_param` is to set one or more parameter values.

`paramName1`, *`Value1`* `...`

One or more pairs, each consisting of a parameter name and a value to which `set_param` is to set that parameter in *`object`*.

Examples

Set the `Solver` and `StopTime` parameters of the `vdp` system:

```
set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000')
```

Set the `Gain` parameter of the Mu block in the `vdp` system to 1000:

```
set_param('vdp/Mu', 'Gain', '1000')
```

Set the position of the Fcn block in the `vdp` system:

```
set_param('vdp/Fcn', 'Position', [50 100 110 120])
```

Set 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]')
```

Set the `Gain` parameter `k` for a block in a masked subsystem.

```
set_param('mymodel/Subsystem', 'k', '10')
```

Set the `OpenFcn` callback parameter of the block named `Compute` in system `mymodel`.

```
set_param('mymodel/Compute', 'OpenFcn', 'my_open_fcn')
```

See Also

`find_system` | `gcb` | `get_param`

How To

- "Associating User Data with Blocks"
- Chapter 8, "Model and Block Parameters"

- “Using Callback Functions”
- “Using MATLAB Commands to Change Workspace Data”

setActiveConfigSet

Purpose

Specify model's active configuration set or configuration reference

Syntax

```
setActiveConfigSet('model', 'configObjName')
```

Arguments

model
: The name of an open model, or `gcs` to specify the current model

configObjName
: The name of a configuration set (`Simulink.ConfigSet`) or configuration reference (`Simulink.ConfigSetRef`)

Description

`setActiveConfigSet` specifies the active configuration set or configuration reference (configuration object) of *`model`* to be the configuration object specified by *`configObjName`*. If no such configuration object is attached to the model, an error occurs. The previously active configuration object becomes inactive.

Example

The following example makes `DevConfig` the active configuration object of the current model. The code is the same whether `DevConfig` is a configuration set or configuration reference.

```
setActiveConfigSet(gcs, 'DevConfig');
```

See Also

"Setting Up Configuration Sets", "Referencing Configuration Sets"

`attachConfigSet`, `attachConfigSetCopy`, `closeDialog`, `detachConfigSet`, `getActiveConfigSet`, `getConfigSet`, `getConfigSets`, `openDialog`

Purpose

Create MATLAB structure describing signed generalized fixed-point data type

Syntax

```
a = sfix(TotalBits)
```

Description

`sfix(TotalBits)` returns a MATLAB structure that describes the data type of a signed generalized fixed-point number with a word size given by `TotalBits`.

`sfix` is automatically called when a signed generalized fixed-point data type is specified in a block dialog box.

Note A default binary point is not included in this data type description. Instead, the scaling must be explicitly defined in the block dialog box.

Examples

Define a 16-bit signed generalized fixed-point data type:

```
a = sfix(16)
a =
       Class: 'FIX'
    IsSigned: 1
    MantBits: 16
```

See Also

`fixdt`, `float`, `sfrac`, `sint`, `ufix`, `ufrac`, `uint`

Purpose

Create MATLAB structure describing signed fractional data type

Syntax

```
a = sfrac(TotalBits)
a = sfrac(TotalBits, GuardBits)
```

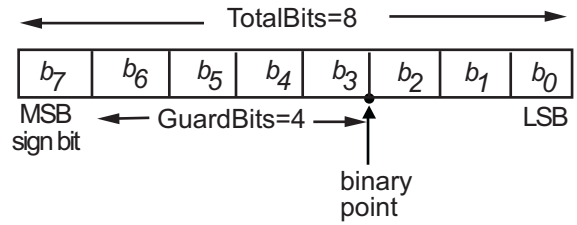
Description

`sfrac(TotalBits)` returns a MATLAB structure that describes the data type of a signed fractional number with a word size given by `TotalBits`.

`sfrac(TotalBits, GuardBits)` returns a MATLAB structure that describes the data type of a signed fractional number. The total word size is given by `TotalBits` with `GuardBits` bits located to the left of the binary point.

`sfrac` is automatically called when a signed fractional data type is specified in a block dialog box.

The most significant (leftmost) bit is the sign bit. The default binary point for this data type is assumed to lie immediately to the right of the sign bit. If guard bits are specified, they lie to the left of the binary point and to right of the sign bit. For example, the structure for an 8–bit signed fractional data type with 4 guard bits is:



Examples

Define an 8-bit signed fractional data type with 4 guard bits. Note that the range of this number is $-2^4 = -16$ to $(1 - 2^{(1 - 8)}).2^4 = 15.875$:

```
a = sfrac(8,4)
a =
        Class: 'FRAC'
     IsSigned: 1
```

```
     MantBits: 8
    GuardBits: 4
```

See Also

fixdt, float, sfix, sint, ufix, ufrac, uint

signalbuilder

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(path, 'create', time, data, siglabels,
grouplabels)
block = signalbuilder(path,'create', time, data, siglabels,
grouplabels, vis)
block = signalbuilder(path,'create', time, data, siglabels,
grouplabels, vis, pos)

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)

signalbuilder(block, 'print', [])
signalbuilder(block, 'print', config, printArgs)
figh = signalbuilder(block, 'print', config, 'figure')
```

Description

Use the `signalbuilder` command to interact programmatically with Signal Builder blocks.

- "Create and Access Signal Builder Blocks" on page 4-165
- "Get/Set Methods for Specific Signals and Groups" on page 4-166
- "Query and Set the Active Group" on page 4-167

• "Print Signal Groups" on page 4-167

Create and Access Signal Builder Blocks

`[time, data] = signalbuilder(block)` returns the time (x-coordinate) and amplitude (y-coordinate) data of the Signal Builder block, *`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. The preceding table describes the allowable formats of *`time`* and *`data`*. 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`. Always provide `time` and `data` when using the `create` command. These two parameters are required always.

`block = signalbuilder(path, 'create', time, data, siglabels, grouplabels)` creates a new Signal Builder block at *`path`* using the specified values. If *`path`* is empty, the function creates a block in a new model, which has a default name. If *`data`* is a cell array and *`time`* is a vector, the *`time`* values are duplicated for each element of *`data`*. Each vector within *`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`*. Always provide `time` and `data` when using the `create` command. These two parameters are required always.

`block = signalbuilder(path,'create', time, data, siglabels, grouplabels, vis)` creates a new Signal Builder block and sets the visible signals in each group based on the values of the matrix `vis`. This matrix must be the same size as the cell array, *`data`*. Always provide `time` and `data` when using the `create` command. These two parameters are required always.

`block = signalbuilder(path,'create', time, data, siglabels, grouplabels, vis, pos)` creates a new Signal Builder block and sets the block position to *`pos`*. Always provide `time` and `data` when using the `create` command. These two parameters are required always.

`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.

Note The `signalbuilder` function does not allow you to alter and delete data in the same invocation. It also does not allow you to delete all the signals and groups from the application.

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`*.

Print Signal Groups

`signalbuilder(block, 'print', [])` prints the currently active signal group.

`signalbuilder(block, 'print', config, printArgs)` prints the currently active signal group or the signal group that *`config`* specifies. The argument *`config`* is a structure that allows you to customize the printed appearance of a signal group. The *`config`* structure may contain any of the following fields:

Field	Description	Example Value
`groupIndex`	Scalar specifying index of signal group to print	`2`
`timeRange`	Two-element vector specifying the time range to print (must not exceed the block's time range)	`[3 6]`
`visibleSignals`	Vector specifying index of signals to print	`[1 2]`

Field	Description	Example Value
`yLimits`	Cell array specifying limits for each signal's *y*-axis	`{[-1 1], [0 1]}`
`extent`	Two-element vector of the form: `[width, height]` specifying the dimensions (in pixels) of the area in which to print the signals	`[500 300]`
`showTitle`	Logical value specifying whether to print a title; `true` (1) prints the title	`false`

The optional argument *`printArgs`* allows you to configure print options (see `print` in the MATLAB Function Reference).

`figh = signalbuilder(`*`block`*`, 'print', `*`config`*`, 'figure')` prints the currently active signal group or the signal group that *`config`* specifies to a new hidden figure handle, `figh`.

Examples

Example 1

The following command creates a new Signal Builder block in a new model editor window:

```
block = signalbuilder([], 'create', [0 5], {[2 2];[0 2]});
```

The Signal Builder block contains two signals in one group. To alter the second signal in the group, use the `set` keyword as follows:

```
signalbuilder(block, 'set', 2, 1, [0 5], [2 0])
```

To delete the first signal from the group, enter the following command:

```
signalbuilder(block, 'set', 1, 1, [], [])
```

To add a new signal in a new group, use the `append` keyword as follows:

```
signalbuilder(block, 'append', [0 2.5 5], [0 2 0], 'Signal 2', 'Gro
```

Example 2

The following command creates a new Signal Builder block in a new model editor window:

```
block = signalbuilder([], 'create', [0 2], {[0 1],[1 0]});
```

The Signal Builder block has two groups, each of which contains a signal. To delete the second group, simply delete its signal with the following command:

```
signalbuilder(block, 'set', 1, 2, [], [])
```

Example 3

The following command creates a new Signal Builder block in a new model editor window:

```
block = signalbuilder([], 'create', [0 1], ...
         {[0 0],[1 1];[1 0],[0 1];[1 1],[0 0]});
```

The Signal Builder block has two groups, each of which contains three signals.

Purpose

Simulate dynamic system

Syntax

```
simOut = sim('model', 'ParameterName1',Value1,
   'ParameterName2', Value2...);
simOut = sim('model', ParameterStruct);
simOut = sim('model', ConfigSet);
```

Description

`simOut = sim('model', 'ParameterName1',Value1,'ParameterName2', Value2...);` causes Simulink software to simulate the block diagram, *`model`*, using parameter name-value pairs `ParameterName1`, `Value1` and `ParameterName2`, `Value2`.

`simOut = sim('model', ParameterStruct);` causes Simulink software to simulate the block diagram, *`model`*, using the parameter values specified in the structure *`ParameterStruct`*.

`simOut = sim('model', ConfigSet);` causes Simulink software to simulate the block diagram, *`model`*, using the configuration settings specified in the model configuration set, *`ConfigSet`*.

Inputs

`ParameterNamek`

The name of a parameter to be specified for simulation.

`Valuek`

The value of the parameter, `ParameterNamek` (`Value1` is the value of `ParameterName1`.)

`ParameterStruct`

A structure containing parameter settings

`ConfigSet`

A configuration set

Outputs

`simOut`

A Simulink.SimulationOutput object that contains all of the simulation outputs—logged time, states, and signals

Definitions

For all three formats of the `sim` command, the input(s) are parameter specifications that override those defined on the Configuration Parameters dialog box. The software restores the original configuration values at the end of simulation. For additional details about the `sim` command, see "Using the sim Command".

Examples

Simulate the model, `vdp`, in Rapid Accelerator mode for an absolute tolerance of 1e-5 and save the states in *xoutNew* and the output in *youtNew*.

1 Specify parameter name-value pairs within the `sim` command:

```
simOut = sim('vdp','SimulationMode','rapid','AbsTol','1e-5',...
             'SaveState','on','StateSaveName','xoutNew',...
             'SaveOutput','on','OutputSaveName','youtNew');
```

2 Specify parameter values in a structure, `paramNameValStruct`:

```
paramNameValStruct.SimulationMode = 'rapid';
paramNameValStruct.AbsTol         = '1e-5';
paramNameValStruct.SaveState      = 'on';
paramNameValStruct.StateSaveName  = 'xoutNew';
paramNameValStruct.SaveOutput     = 'on';
paramNameValStruct.OutputSaveName = 'youtNew';
simOut = sim('vdp',paramNameValStruct);
```

3 Specify a configuration set containing the parameter values:

```
mdl = 'vdp';
load_system(mdl)
simMode = get_param(mdl, 'SimulationMode');
set_param(mdl, 'SimulationMode', 'rapid')
cs = getActiveConfigSet(mdl);
```

```
mdl_cs = cs.copy;
set_param(mdl_cs,'AbsTol','1e-5',...
          'SaveState','on','StateSaveName','xoutNew',...
          'SaveOutput','on','OutputSaveName','youtNew')
simOut = sim(mdl, mdl_cs);
set_param(mdl, 'SimulationMode', simMode)
```

Alternatives

See Also

`sldebug` | `parfor` | Rapid Accelerator Simulations Using PARFOR

Purpose

Plot simulation data in figure window

Syntax

```
simplot(data);
simplot(time, data);
simplot(data, ports);
simplot(data, 'diff')
simplot(time, data, ports, 'diff')
```

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.

The data to be plotted can be either a data structure or a matrix of the form produced by Simulink output blocks.

Specifying a Separate Time Vector

If `data` is a matrix or a structure without time, you can specify a separate time vector. `time` must be a vector with the same length as `data`.

Foe example:

```
simplot(time,data)
```

Specifying Specific Ports to Display

If `data` is a structure produced by a multi-port Scope block, the data from each port is displayed in a separate subplot. You can select specific ports to display by supplying a vector of port indices.

For example:

```
ports = [1,3];
simplot(data,ports)
```

plots the data from the first and third ports.

Overlaying Plots from Multiple Runs

If `data` is a cell array of structures or matrices, Simulink software overlays the plots from each element so that you can compare multiple runs. Each run is assumed to have identical structure. Line styles are used to differentiate between runs.

For example:

```
data = {run1, run2};
simplot(data)
```

overlays the data from `run1` and `run2`.

Note If `data` contains Matrices or Structures without time, the data sets for all runs must be the same size.

You can use the `'diff'` flag to display the differences between multiple runs. When you specify the `'diff'` flag, Simulink software subtracts the first run from subsequent runs, and plots the results with the line style of the final run being compared.

For example:

```
data = {run1,run2};
simplot(data, 'diff')
```

plots `run2 — run 1` using the line style of `run2`.

Note Simulink software uses linear interpolation if the time vectors are not identical.

If the start and stop times differ between runs, the difference is only plotted for the region of overlap.

Combining Input Argument Options:

The options described above can be used in various combinations. All input arguments except for `data` are optional but when included must be entered in the following order:

```
simplot(time, data, ports, 'diff')
```

Obtaining Object Handles

You can obtain the handles for the plotted figure, its axes and lines using the `simplot` command:

- `hfig = simplot(data)` — returns the figure handles.
- `haxes = simplot(data)` — returns the handles for the figure axes.
- `hlins = simplot(data)` — returns the handles for the lines in the figure.

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").
`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.

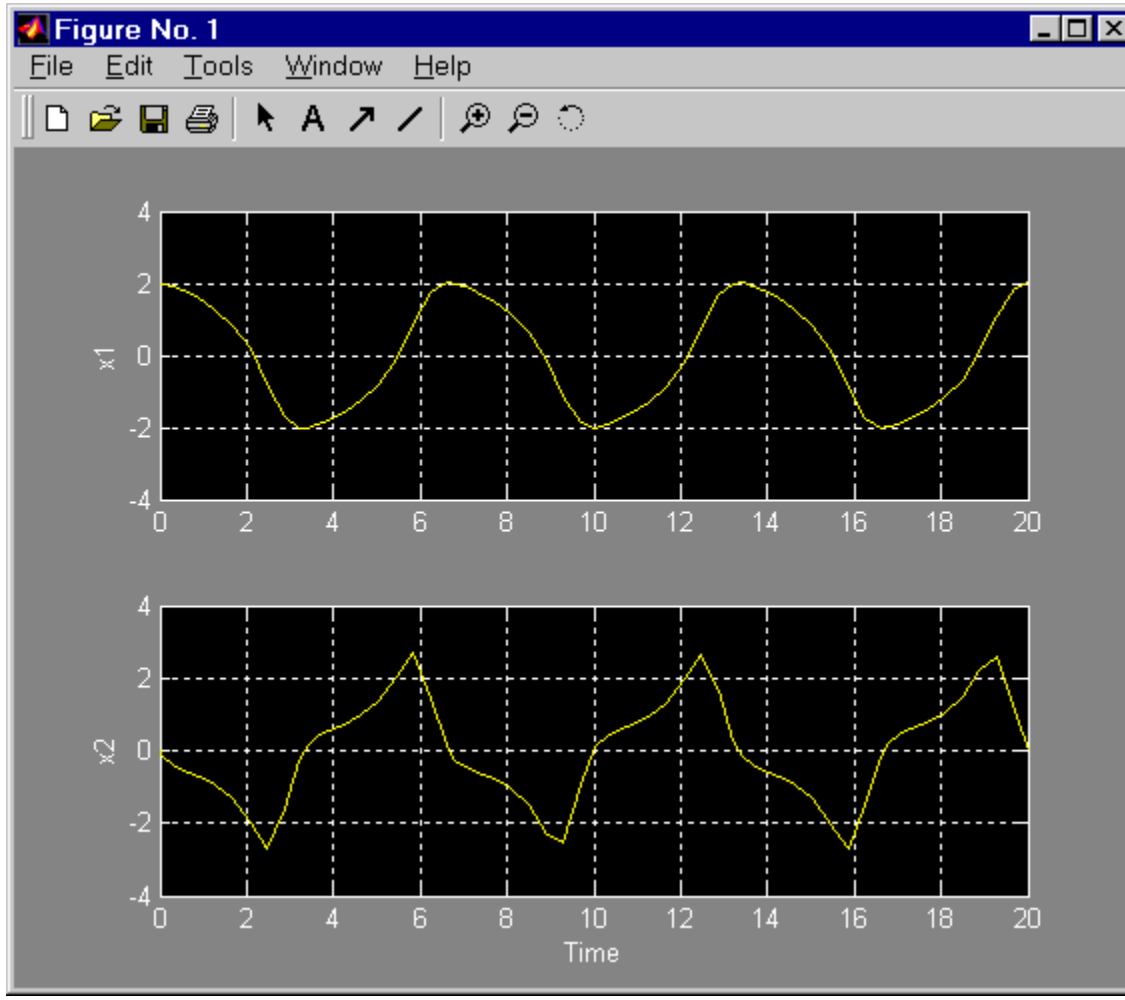
`ports`	The vector of port indices from which to display data. If the data is a structure produced by a multi-port Scope block, the data from each port is displayed in a separate subplot.
`'diff'`	Displays the differences between multiple runs. When you specify the 'diff' flag, Simulink software subtracts the first run from subsequent runs, and plots the results with the line style of the final run being compared.

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`, `plot`

simulink

Purpose

Open Simulink block library

Syntax

```
simulink
simulink('open')
simulink('close')
```

Description

`simulink` or `simulink('open')` opens the Simulink Library Browser. `simulink('close')` closes the Library Browser.

Purpose

Return sample time information for a block

Syntax

```
ts = Simulink.Block.getSampleTimes(block)
```

Arguments

block

Full name or handle of a Simulink block

Returns

`ts`

The command returns `ts` which is a `1xn` array of Simulink.SampleTime objects associated with the model passed to Simulink.Block.getSampleTimes. Here `n` is the number of sample times associated with the block. The format of the returns is:

```
1xn Simulink.SampleTime
Package: Simulink

value: [1x2 double]
Description: [char string]
ColorRGBValue: [1x3 double]
Annotation: [char string]
OwnerBlock: [char string]
ComponentSampleTimes: [1x2 struct]

Methods
```

- `value` — A two-element array of doubles that contains the sample time period and offset
- `Description` — A character string that describes the sample time type
- `ColorRGBValue` — A 1x3 array of doubles that contains the red, green and blue (RGB) values of the sample time color
- `Annotation` — A character string that represents the annotation of a specific sample time (e.g., 'D1')

- `OwnerBlock` — For asynchronous and variable sample times, a string containing the full path to the block that controls the sample time. For all other types of sample times, an empty string.
- `ComponentSampleTimes` — A structure array of elements of the same type as `Simulink.BlockDiagram.getSampleTimes` if the sample time is an async union or if the sample time is hybrid and the component sample times are available.

Description

`ts = Simulink.Block.getSampleTimes(`*`block`*`)` performs an update diagram and then returns the sample times of the block connected to the input argument *`mdl`*`/`*`signal`*. This method performs an update diagram to ensure that the sample time information returned is up-to-date. If the model is already in the compiled state via a call to the model API, then an update diagram is not necessary.

Using this method allows you to access all information in the Sample Time Legend programmatically.

See Also

`Simulink.BlockDiagram.getSampleTimes`

Purpose

Add Bus to Vector blocks to convert virtual bus signals into vector signals

Syntax

```
[DstBlocks,
   BusToVectorBlocks] = Simulink.BlockDiagram.addBusToVector(
   model)
[DstBlocks,
   BusToVectorBlocks] = Simulink.BlockDiagram.addBusToVector(
   model, includeLibs)
[DstBlocks,
   BusToVectorBlocks] = Simulink.BlockDiagram.addBusToVector(
   model, includeLibs, reportOnly)
```

Description

`[DstBlocks, BusToVectorBlocks] = Simulink.BlockDiagram.addBusToVector(model)` searches a model, excluding any library blocks, for bus signals used implicitly as vectors, and returns the results of the search. Before executing this function, you must do the following:

1. Set **Configuration Parameters > Diagnostics > Connectivity > Buses > Mux blocks used to create bus signals** to `error`, or equivalently, execute `set_param (model, 'StrictBusMsg', 'ErrorLevel1')`.
2. Ensure that the model compiles without error.
3. Save the model.

`[DstBlocks, BusToVectorBlocks] = Simulink.BlockDiagram.addBusToVector(model, includeLibs)` is equivalent to `[DstBlocks, BusToVectorBlocks] = Simulink.BlockDiagram.addBusToVector(model)` if *includeLibs* is `false`.

If *includeLibs* is `true`, the function searches library blocks rather than excluding them.

`[DstBlocks, BusToVectorBlocks] = Simulink.BlockDiagram.addBusToVector(model, includeLibs, reportOnly)` is equivalent to `[DstBlocks, BusToVectorBlocks] = Simulink.BlockDiagram.addBusToVector(model, includeLibs)` if `reportOnly` is `true`.

If `reportOnly` is `false`, the function inserts a Bus to Vector block into each bus that is used as a vector in any block that it searches. The search excludes or includes library blocks as specified by `includeLibs`. The insertion replaces the implicit use of a bus as a vector with an explicit conversion of the bus to a vector. The signal's source and destination blocks are unchanged by this insertion.

If `Simulink.BlockDiagram.addBusToVector` adds Bus to Vector blocks to the model or any library, the function permanently changes the saved copy of the diagram. Be sure to back up the model and any libraries before calling the function with `reportOnly` specified as `false`.

If `Simulink.BlockDiagram.addBusToVector` changes a library block, the change affects every instance of that block in every Simulink model that uses the library. To preview the effects of the change on blocks in all models, call `Simulink.BlockDiagram.addBusToVector` with `includeLibs` = `true` and `reportOnly` = `true`, then examine the information returned in `DstBlocks`.

The Bus to Vector block is intended only for use in existing models to facilitate the elimination of implicit conversion of buses into vectors. New models and new parts of existing models should avoid implicitly using buses as vectors, and should not use Bus to Vector blocks for any purpose. See Avoiding Mux/Bus Mixtures for more information about using `Simulink.BlockDiagram.addBusToVector`.

Inputs

`model`

Model name or handle

`includeLibs`

Boolean specifying whether to search library blocks (`true`) or only the top-level model (`false`).

Default: `false`

`reportOnly`

Boolean specifying whether to change the model (`false`) or just generate a report (`true`).

Default: `true`

Outputs

`DstBlocks`

An array of structures that contain information about blocks that are connected to buses but treat the buses as vectors. If no such blocks exist the array has 0 length. Each structure in the array contains the following fields:

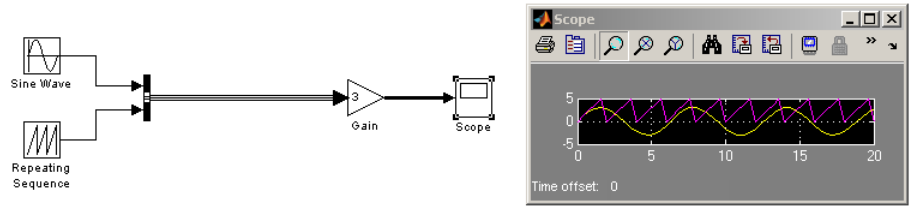
`BlockPath`	String specifying the path to the block to which the bus connects
`InputPort`	Integer specifying the input port to which the bus connects
`LibPath`	If the block is a library block instance, and *`includeLibs`* is `true`, the path to the source library block. Otherwise, `LibPath` is empty ([]).

`BusToVectorBlocks`

If *`reportOnly`* is `false`, and *`model`* contains any buses used as vectors, a cell array containing the path to each Bus to Vector block that was added to the model. Otherwise, *`BusToVectorBlocks`* is empty ([]).

Example

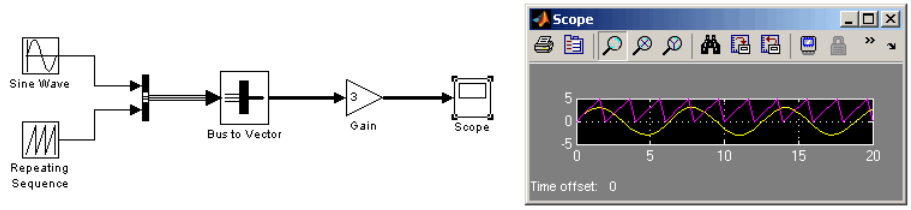
The following model simulates correctly, but the input to the Gain block is a bus, while the output is a vector. Thus the Gain block uses a block as a vector.



If the model shown is open as the current model, you can eliminate the implicit conversion with the following command:

```
Simulink.BlockDiagram.addBusToVector(gcs, false, false)
```

Rebuilding and simulating the model then gives this result:



The Gain block no longer implicitly converts the bus to a vector; the inserted Bus to Vector block performs the conversion explicitly. Note that the results of simulation are the same for both models. The Bus to Vector block is virtual, and never affects simulation results, code generation, or performance.

See Also

Bus Assignment | Bus Creator | Bus Selector | Bus to Vector | `Simulink.Bus` | `Simulink.Bus.cellToObject` | `Simulink.Bus.createObject` | `Simulink.BusElement` | `Simulink.Bus.objectToCell` | `Simulink.Bus.save`

How To

- “Mux Signals”
- “Using Composite Signals”
- Avoiding Mux/Bus Mixtures

Purpose

Copy contents of block diagram to empty subsystem

Syntax

```
Simulink.BlockDiagram.copyContentsToSubSystem(bdiag,
subsys)
```

Description

`Simulink.BlockDiagram.copyContentsToSubSystem(`*`bdiag`*`,` *`subsys`*`)` copies the contents of the block diagram *`bdiag`* to the subsystem *`subsys`*. The block diagram and subsystem must have already been loaded. The subsystem cannot be part of the block diagram.

The function affects only blocks, lines, and annotations; it does not affect nongraphical information such as configuration sets. You can use this function to convert a referenced model derived from an atomic subsystem into an atomic subsystem that is equivalent to the original subsystem.

This function cannot be used if the destination subsystem contains any blocks or signals. Other types of information can exist in the destination subsystem and are not affected by the function. Use `Simulink.SubSystem.deleteContents` if necessary to empty the subsystem before using `Simulink.BlockDiagram.copyContentsToSubSystem`.

Inputs

`bdiag`

Block diagram name or handle

`subsys`

Subsystem name or handle

Examples

Copy the contents of `f14`, including all nested subsystems, to an empty subsystem named `f14cp` that already exists in the model named `f16`:

```
% open f14 and f16
open_system('f14');
open_system('f16');
```

```
% copy the block diagram f14 to an empty subsystem of f16 named f14cp
Simulink.BlockDiagram.copyContentsToSubSystem('f14', 'f16/f14cp');

% close f14 and f16
close_system('f14', 0);
close_system('f16', 0);
```

See Also

Simulink.BlockDiagram.deleteContents |
Simulink.SubSystem.convertToModelReference |
Simulink.SubSystem.copyContentsToBlockDiagram |
Simulink.SubSystem.deleteContents

How To

- "Systems and Subsystems"
- "Creating Subsystems"

Purpose

Create a subsystem containing a specified set of blocks

Syntax

```
Simulink.BlockDiagram.createSubSystem(blocks)
Simulink.BlockDiagram.createSubSystem()
```

Description

`Simulink.BlockDiagram.createSubSystem(blocks)` creates a new subsystem and moves the specified blocks into the subsystem. All of the specified blocks must originally reside at the top level of the model or in the same existing subsystem within the model.

If any of the blocks have unconnected input ports, the command creates input port blocks for each unconnected input port in the subsystem and connects the input port block to the unconnected input port. The command similarly creates and connects output port blocks for unconnected output ports on the specified blocks. If any of the specified blocks is an input port, the command creates an input port block in the parent system and connects it to the corresponding input port of the newly created subsystem. The command similarly creates and connects output port blocks for each of the specified blocks that is an output port block.

`Simulink.BlockDiagram.createSubSystem()` creates a new subsystem in the currently selected model and moves the currently selected blocks within the current model to the new subsystem.

Inputs

`blocks`

An array of block handles

Default: `[]`

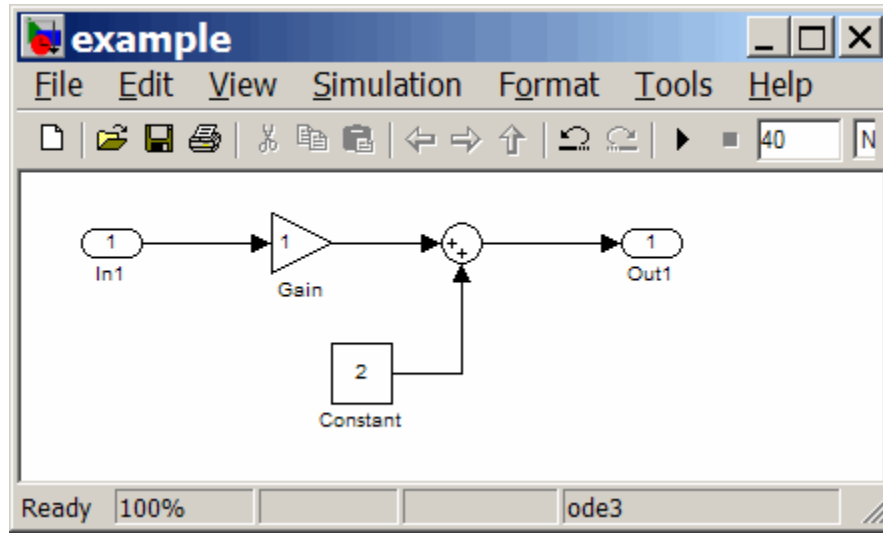
Examples

The following function converts the contents of a model or subsystem into a subsystem.

```
function convert2subsys(sys)
  blocks = find_system(sys, 'SearchDepth', 1);
  bh = [];
  for i = 2:length(blocks)
```

```
    bh = [bh get_param(blocks{i}, 'handle')];
  end
  Simulink.BlockDiagram.createSubSystem(bh);
end
```

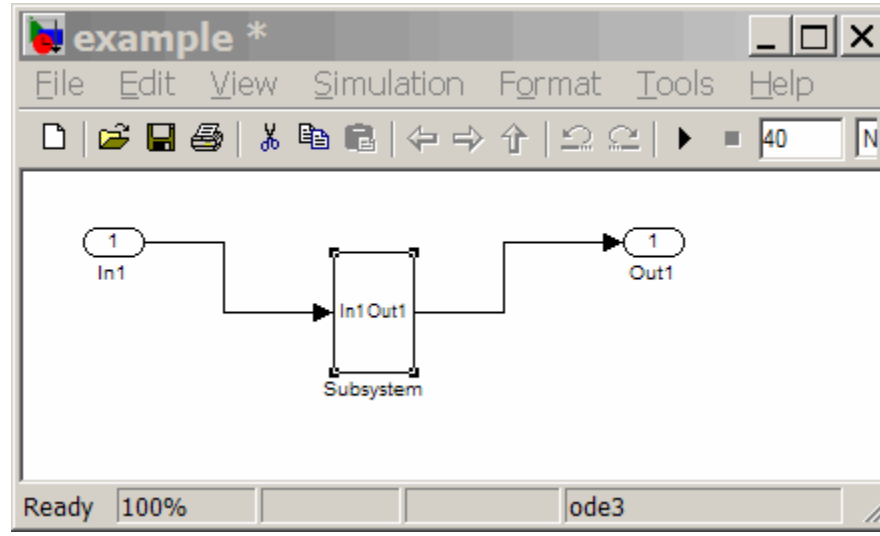
Consider, for example, the following model:



Executing

```
convert2subsys('example');
```

converts this model to:



See Also

`Simulink.BlockDiagram.copyContentsToSubSystem | Simulink.BlockDiagram.deleteContents | Simulink.SubSystem.convertToModelReference | Simulink.SubSystem.copyContentsToBlockDiagram`

How To

- "Systems and Subsystems"
- "Creating Subsystems"

Simulink.BlockDiagram.deleteContents

Purpose

Delete contents of block diagram

Syntax

```
Simulink.BlockDiagram.deleteContents(bdiag)
```

Description

`Simulink.BlockDiagram.deleteContents(bdiag)` deletes the contents of the block diagram *`bdiag`*. The block diagram must have already been loaded. The function affects only blocks, lines, and annotations. It does not affect nongraphical information such as configuration sets.

Inputs

`bdiag`

Block diagram name or handle

Example

Delete the graphical content of an open block diagram named `f14`, including all subsystems:

```
Simulink.BlockDiagram.deleteContents('f14');
```

See Also

`Simulink.BlockDiagram.copyContentsToSubSystem` | `Simulink.SubSystem.convertToModelReference` | `Simulink.SubSystem.copyContentsToBlockDiagram` | `Simulink.SubSystem.deleteContents`

How To

- "Creating a Model"
- "Creating Subsystems"

Purpose

Return checksum of model

Syntax

```
[checksum,details] = Simulink.BlockDiagram.getChecksum(mdl)
```

Description

`[checksum,details] = Simulink.BlockDiagram.getChecksum(mdl)` returns the checksum of the specified model. Simulink software computes the checksum based on attributes of the model and the blocks the model contains.

One use of this command is to determine why the Accelerator mode in Simulink software regenerates code. For an example, see the demo `slAccelDemoWhyRebuild`.

Note `Simulink.BlockDiagram.getChecksum` compiles the specified model, using the command *`model`*`([], [], [], 'compileForRTW')`, if the model is not already in a compiled state. To get the checksum for the model when Simulink software compiles it for simulation, use the command *`model`*`([], [], [], 'compile')` to place the model in a compiled state before using `Simulink.BlockDiagram.getChecksum`.

This command accepts the argument *`mdl`*, which is the full name or handle of the model for which you are returning checksum data.

This command returns the following output:

- *`checksum`* — Array of four 32-bit integers that represents the model's 128-bit checksum.
- *`details`* — Structure of the form

```
ContentsChecksum: [1x1 struct]
InterfaceChecksum: [1x1 struct]
ContentsChecksumItems: [nx1 struct]
InterfaceChecksumItems: [mx1 struct]
```

- `ContentsChecksum` — Structure of the following form that represents a checksum that provides information about all blocks in the model.

  ```
  Value: [4x1 uint32]
  MarkedUnique: [bool]
  ```

  - `Value` — Array of four 32-bit integers that represents the model's 128-bit checksum.
  - `MarkedUnique` — True if any blocks in the model have a property that prevents code reuse.
- `InterfaceChecksum` — Structure of the following form that represents a checksum that provides information about the model.

  ```
  Value: [4x1 uint32]
  MarkedUnique: [bool]
  ```

  - `Value` — Array of four 32-bit integers that represents the model's 128-bit checksum.
  - `MarkedUnique` — Always true. Present for consistency with `ContentsChecksum` structure.
- `ContentsChecksumItems` and `InterfaceChecksumItems` — Structure arrays of the following form that contain information that Simulink software uses to compute the checksum for `ContentsChecksum` and `InterfaceChecksum`, respectively:

  ```
  Handle: [char array]
  Identifier: [char array]
  Value: [type]
  ```

  - `Handle` — Object for which Simulink software added an item to the checksum. For a block, the handle is a full block path. For a block port, the handle is the full block path and a string that identifies the port.

- `Identifier` — Descriptor of the item Simulink software added to the checksum. If the item is a documented parameter, the identifier is the parameter name.
- `Value` — Value of the item Simulink software added to the checksum. If the item is a parameter, `Value` is the value returned by

  ```
  get_param(handle, identifier)
  ```

See Also

`Simulink.SubSystem.getChecksum`

Simulink.BlockDiagram.getInitialState

Purpose

Return initial state structure of block diagram

Syntax

```
x0 = Simulink.BlockDiagram.getInitialState(mdl)
```

Description

`x0 = Simulink.BlockDiagram.getInitialState(mdl)` returns the initial state structure of the block diagram specified by the input argument *`mdl`*. This state structure can be used to specify the initial state vector in the **Configuration Parameters** dialog box or to provide an initial state condition to the linearization commands.

The command returns `x0`, a structure of the form

```
time: 0
signals: [1xn struct]
```

where `n` is the number of states contained in the model, including any models referenced by Model blocks. The `signals` field is a structure of the form

```
values: [1xm double]
dimensions: [1x1 double]
label: [char array]
blockName: [char array]
inReferencedModel: [bool]
sampleTime: [1x2 double]
```

- `values` — Numeric array of length `m`, where `m` is the number of states in the signal
- `dimensions` — Length of the `values` vector
- `label` — Indication of whether the state is continuous (`CSTATE`) or discrete. If the state is discrete:

  The name of the discrete state will be shown for S-function blocks

  The name of the discrete state will be shown for those built-in blocks that assign their own names to discrete states

  `DSTATE` is used in all other cases

- `blockName` — Full path to block associated with this state
- `inReferencedModel` — Indication of whether the state originates in a model referenced by a Model block (`1`) or in the top model (`0`)
- `sampleTime` — Array containing the sample time and offset of the state

Using the state structure simplifies specifying initial state values for models with multiple states, as each state is associated with the full path to its parent block.

See Also

`linmod`

Simulink.BlockDiagram.getSampleTimes

Purpose

Return all sample times associated with a model

Syntax

```
ts = Simulink.BlockDiagram.getSampleTimes(mdl)
```

Arguments

mdl

Name or handle of a Simulink model

Returns

`ts`

The command returns `ts` which is a `1xn` array of Simulink.SampleTime objects associated with the model passed to Simulink.BlockDiagram.getSampleTimes. Here `n` is the number of sample times associated with the block diagram. The format of the returns is as follows:

```
1xn Simulink.SampleTime
Package: Simulink

value: [1x2 double]
Description: [char string]
ColorRGBValue: [1x3 double]
Annotation: [char string]
OwnerBlock: [char string]
ComponentSampleTimes: [1x2 struct]

Methods
```

- `value` — A two-element array of doubles that contains the sample time period and offset
- `Description` — A character string that describes the sample time type
- `ColorRGBValue` — A 1x3 array of doubles that contains the red, green and blue (RGB) values of the sample time color
- `Annotation` — A character string that represents the annotation of a specific sample time (e.g., 'D1')

- `OwnerBlock` — For asynchronous and variable sample times, a string containing the full path to the block that controls the sample time. For all other types of sample times, an empty string.
- `ComponentSampleTimes` — A structure array of elements of the same type as `Simulink.BlockDiagram.getSampleTimes` if the sample time is an async union or if the sample time is hybrid and the component sample times are available.

Description

`ts = Simulink.BlockDiagram.getSampleTimes(`*`mdl`*`)` performs an update diagram and then returns the sample times associated with the block diagram specified by the input argument *`mdl`*. The update diagram ensures that the sample time information returned is up-to-date. If the model is already in the compiled state via a call to the model API, then an update diagram is not necessary.

Using this method allows you to access all information in the Sample Time Legend programmatically.

See Also

`Simulink.Block.getSampleTimes`

Simulink.Bus.cellToObject

Purpose

Convert cell array containing bus information to bus objects

Syntax

```
Simulink.Bus.cellToObject(busCells)
```

Description

`Simulink.Bus.cellToObject(busCells)` creates a set of bus objects in the MATLAB base workspace from a cell array of bus information. The inverse function is `Simulink.Bus.objectToCell`.

Input

`busCells`

A cell array of cell arrays in which each subordinate array represents a bus object and contains the following data:

```
{BusName, HeaderFile, Description, BusElements}
```

The *`BusElements`* field is an array containing the following data for each element:

```
{ElementName, Dimensions, DataType,
SampleTime, Complexity, SamplingMode}
```

See Also

Bus Assignment | Bus Creator | Bus Selector | Bus to Vector | `Simulink.Bus` | `Simulink.Bus.createObject` | `Simulink.BusElement` | `Simulink.Bus.objectToCell` | `Simulink.Bus.save`

How To

- "Using Composite Signals"

Purpose

Create bus objects for blocks

Syntax

```
busInfo = Simulink.Bus.createObject(modelName, blks)
busInfo = Simulink.Bus.createObject(modelName, blks,
   fileName)
busInfo = Simulink.Bus.createObject(modelName, blks,
   fileName, 'format')
```

Description

`busInfo = Simulink.Bus.createObject(modelName, blks)` creates bus objects (instances of `Simulink.Bus` class in the MATLAB base workspace) for specified blocks, and returns information about the objects that it created.

`busInfo = Simulink.Bus.createObject(modelName, blks, fileName)` saves the bus objects in an M-file that contains a cell array of cell arrays. Each subordinate array represents a bus object and contains the following data:

```
{BusName, HeaderFile, Description, BusElements}
```

The *BusElements* field is an array containing the following data for each element:

```
{ElementName, Dimensions, DataType,
SampleTime, Complexity, SamplingMode}
```

`busInfo = Simulink.Bus.createObject(modelName, blks, fileName, 'format')` saves the bus objects in an M-file that contains either a cell array of bus information, or the bus objects themselves.

Inputs

modelName

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. By default, the function saves the objects in a cell array.

format

Format used to store the bus objects. The value can be `'cell'` or `'object'`. Use cell array format to save the objects in a compact form.

Default: `'cell'`

Outputs

busInfo

A structure array containing bus information for the specified blocks. Each element of the structure array corresponds to one block and contains the following fields:

`block`	Handle of the block
`busName`	Name of the bus object associated with the block

See Also

Bus Assignment | Bus Creator | Bus Selector | Bus to Vector | `Simulink.Bus` | `Simulink.Bus.cellToObject` | `Simulink.BusElement` | `Simulink.Bus.objectToCell` | `Simulink.Bus.save`

How To

- "Using Composite Signals"

Purpose

Convert bus objects to cell array containing bus information

Syntax

```
busCells = Simulink.Bus.objectToCell(busNames)
```

Description

`busCells = Simulink.Bus.objectToCell(busNames)` inputs a cell array of names of bus objects in the MATLAB base workspace, and returns a cell array of cell arrays in which each subordinate array contains the bus information defined by one of the bus objects. The order of the elements in the output array corresponds to the order of the names in the input array. If *busNames* is empty, the function converts all bus objects in the base workspace. The inverse function is `Simulink.Bus.cellToObject`.

Input

busNames

A cell array of names of bus objects in the MATLAB base workspace

Output

busCells

A cell array of cell arrays in which each subordinate array represents a bus object and contains the following data:

```
{BusName, HeaderFile, Description, BusElements}
```

The *BusElements* field is an array containing the following data for each element:

```
{ElementName, Dimensions, DataType,
SampleTime, Complexity, SamplingMode}
```

See Also

Bus Assignment | Bus Creator | Bus Selector | Bus to Vector | `Simulink.Bus` | `Simulink.Bus.cellToObject` | `Simulink.Bus.createObject` | `Simulink.BusElement` | `Simulink.Bus.save`

How To

- "Using Composite Signals"

Simulink.Bus.save

Purpose

Save bus objects in M-file

Syntax

```
Simulink.Bus.save(fileName)
Simulink.Bus.save(fileName, format)
Simulink.Bus.save(fileName, format, busNames)
```

Description

`Simulink.Bus.save(fileName)` saves all bus objects (instances of `Simulink.Bus` class residing in the MATLAB base workspace) in an M-file that contains a cell array of cell arrays. Each subordinate array represents a bus object and contains the following data:

```
{BusName, HeaderFile, Description, BusElements}
```

The *`BusElements`* field is an array containing the following data for each element:

```
{ElementName, Dimensions, DataType,
SampleTime, Complexity, SamplingMode}
```

Executing an M-file created by Simulink.Bus.save in cell array format calls `Simulink.Bus.cellToObject` to recreate the bus objects and returns the new bus objects in the cell array. To suppress the creation of bus objects, specify the optional argument `'false'` when you execute the M-file.

`Simulink.Bus.save(fileName, format)` saves the bus objects in an M-file that contains either a cell array of bus information or the bus objects themselves.

`Simulink.Bus.save(fileName, format, busNames)` saves only those bus objects whose names appear in *`busNames`*.

Inputs

`fileName`

Name of the file in which to store the bus objects

`format`

Format used to store the bus objects. The value can be `'cell'` or `'object'`. Use cell array format to save the objects in a compact form.

Default: `'cell'`

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.

Default: {}

See Also

Bus Assignment | Bus Creator | Bus Selector | Bus to Vector | `Simulink.Bus` | `Simulink.Bus.cellToObject` | `Simulink.Bus.createObject` | `Simulink.BusElement` | `Simulink.Bus.objectToCell`

How To

- "Using Composite Signals"

Simulink.ModelReference.protect

Purpose

Obscure referenced model contents to hide intellectual property embodied by the model

Syntax

```
Simulink.ModelReference.protect(model)
Simulink.ModelReference.protect(model, 'Path', pathname)
[harnessHandle] = Simulink.ModelReference.protect(model,
   'Harness', true)
[~ ,neededVars] = Simulink.ModelReference.protect(model)
[harnessHandle,
   neededVars] = Simulink.ModelReference.protect(model,
   'Path', pathname, 'Harness', true)
```

Description

`Simulink.ModelReference.protect(model)` creates a protected model from *`model`* and puts it in the current folder. The protected model has the same name as the source model, and the suffix `.mdlp`.

`Simulink.ModelReference.protect(model, 'Path', pathname)` puts the protected model in the folder specified by *`pathname`*.

`[harnessHandle] = Simulink.ModelReference.protect(model, 'Harness', true)` creates a harness model for the protected model and returns the handle of the harnessed model in *`harnessHandle`*.

`[~ ,neededVars] = Simulink.ModelReference.protect(model)` returns in *`neededVars`* a cell array that includes the names of all base workspace entities that the protected model might need in order to execute.

`[harnessHandle,neededVars] = Simulink.ModelReference.protect(model, 'Path', pathname, 'Harness', true)` combines the effect of all the preceding syntaxes into a single call. See “Protecting Referenced Models” for more information.

Inputs

`model`

The name of the model to be protected, or the path name of a Model block that references the model to be protected.

`'Path', pathname`

Specifies that the protected version of the model should be placed in the folder named *pathname*. Omitting the keyword-value pair places the protected model in the current directory.

`'Harness', true`

Specifies that the function will create a harness model and return its handle in *harnessHandle*. Specifying `false` or omitting the keyword-value pair suppresses creation of a harness model.

Outputs

harnessHandle

The handle of the harness model created when the input arguments include `'Harness', true`, or 0 otherwise. You can specify ~ instead of a name if you don't need a harness model but want to obtain the value of the second argument.

neededVars

A cell array that includes the names of all base workspace entities that the protected model might need in order to execute. The function always creates the cell array (though it may be empty) but the array is discarded unless captured by *neededVars*. The array may also include the names of some variables that the model does not need.

How To

- "Creating a Model"
- "Referencing a Model"
- "Protecting Referenced Models"

Simulink.SubSystem.convertToModelReference

Purpose

Convert atomic subsystem or function call subsystem to model reference

Syntax

```
[success,
   mdlRefBlkH] = Simulink.SubSystem.convertToModelReference(su
   bsys, mdlRef, 'opt1', 'val1', ... , 'optN', 'valN')
```

Description

`[success,mdlRefBlkH] = Simulink.SubSystem.convertToModelReference(subsys, mdlRef, 'opt1', 'val1', ... , 'optN', 'valN')` converts an atomic subsystem or function call subsystem to a referenced model. The function creates a new model, copies the contents of the subsystem into the model, sets the new model's configuration parameters, and configures the model's root level `Inport` and `Outport` blocks. The function then replaces the subsystem block with a Model block that references the new model, or creates another, temporary model containing a Model block that references the new model, depending on the input option `ReplaceSubsystem`.

Converting a subsystem to a referenced model requires your model to have the following configuration parameter settings:

- **Configuration Parameters > Optimization > Inline parameters** must be `On`.
- **Configuration Parameters > Diagnostics > Data Validity > Signal resolution** must be `Explicit only`.
- **Configuration Parameters > Diagnostics > Connectivity > Mux blocks used to create bus signals** must be `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');
```

This function produces error or warning messages for models and subsystems that it cannot handle.

- For some errors, a message box appears that gives you the choice of cancelling or continuing.
- If continuing is impossible, Simulink cancels the conversion without offering a choice to continue.

Even if conversion is successful, you may still need to reconfigure the resulting model to meet your requirements.

In the new model, the **Interpolate data** parameter of each root input port is selected by default. You can clear the parameter wherever this default is not appropriate. See the Inport block documentation for information about **Interpolate data**.

Inputs

subsys

Full name or handle of the atomic subsystem block to be converted

mdlRef

Name of the new model to which the subsystem is to be converted

opt1, *val1*, ... , *optN*, *valN*

Zero or more pairs of options and values that control the conversion process. The options and values are:

`'ReplaceSubsystem'`	The option value is a Boolean. If the value is `true`, the function replaces the subsystem block with a Model block that references the model created from the subsystem. If this option is omitted or specified as `false`, the function creates and opens a model containing a Model block that references the model derived from the subsystem block. Default: `false`.
`'BusSaveFormat'`	The option value can be `'Cell'` or `'Object'`. If the option is specified, the function saves any bus objects that it creates in an M-file. The value of the option specifies the format of the file. Use cell array format to save the objects in a compact form. If this option is omitted, the function does not save bus objects to a file.
`'BuildTarget'`	The option value can be `'Sim'` or `'RTW'`. If the option is specified, the function generates a model reference Sim or RTW target for the new model, depending on the option value. If this option is omitted, the function does not generate a model reference target.
`'Force'`	The option value is a Boolean. If the value is `true`, the function reports some errors that would halt the conversion process as warnings and continues with the conversion. This allows you to use the function to do the initial steps of conversion, and then complete the conversion process yourself. If this option is omitted or specified as `false`, the function halts the conversion if an error occurs. Default: `false`

Outputs

success

The value is `true` if the conversion completed successfully, and `false` otherwise.

mdlRefBlkH

Handle of the `Model` block that references the new model.

See Also

`Simulink.BlockDiagram.copyContentsToSubSystem` | `Simulink.Bus.save` | `Simulink.SubSystem.copyContentsToBlockDiagram` | "Converting a Subsystem to a Referenced Model"

Tutorials

- `sldemo_mdlref_conversion`

How To

- "Creating a Model"
- "Creating Subsystems"
- "Referencing a Model"
- "Generating Code for a Referenced Model"

Simulink.SubSystem.copyContentsToBlockDiagram

Purpose

Copy contents of subsystem to empty block diagram

Syntax

```
Simulink.SubSystem.copyContentsToBlockDiagram(subsys,
bdiag)
```

Description

`Simulink.SubSystem.copyContentsToBlockDiagram(`*`subsys`*`,` *`bdiag`*`)` copies the contents of the subsystem *`subsys`* to the block diagram *`bdiag`*. The subsystem and block diagram must have already been loaded. The subsystem cannot be part of the block diagram. The function affects only blocks, lines, and annotations; it does not affect nongraphical information such as configuration sets.

This function cannot be used if the destination block diagram contains any blocks or signals. Other types of information can exist in the destination block diagram and are unaffected by the function. Use `Simulink.BlockDiagram.deleteContents` if necessary to empty the block diagram before using `Simulink.SubSystem.copyContentsToBlockDiagram`.

Inputs

`subsys`

Subsystem name or handle

`bdiag`

Block diagram name or handle

Example

Copy the graphical contents of `f14/Controller`, including all nested subsystems, to a new block diagram:

```
% open f14
open_system('f14');

% create a new model
newbd = new_system;
open_system(newbd);

% copy the subsystem
```

```
Simulink.SubSystem.copyContentsToBlockDiagram('f14/Controller', newbd);

% close f14 and the new model
close_system('f14', 0);
close_system(newbd, 0);
```

See Also

Simulink.BlockDiagram.copyContentsToSubSystem | Simulink.BlockDiagram.deleteContents | Simulink.SubSystem.convertToModelReference | Simulink.SubSystem.deleteContents

How To

- "Creating a Model"
- "Creating Subsystems"

Simulink.SubSystem.deleteContents

Purpose

Delete contents of subsystem

Syntax

```
Simulink.SubSystem.deleteContents(subsys)
```

Description

`Simulink.SubSystem.deleteContents(subsys)` deletes the contents of the subsystem *`subsys`*. The subsystem must have already been loaded. The function affects only blocks, lines, and annotations; it does not affect nongraphical information such as configuration sets.

Inputs

`subsys`

Subsystem name or handle

Example

Delete the graphical contents of `Controller`, including all nested subsystems:

```
Simulink.SubSystem.deleteContents('f14/Controller');
```

See Also

`Simulink.BlockDiagram.copyContentsToSubSystem` | `Simulink.BlockDiagram.deleteContents` | `Simulink.SubSystem.convertToModelReference` | `Simulink.SubSystem.copyContentsToBlockDiagram`

How To

- "Creating a Model"
- "Creating Subsystems"

Purpose

Return checksum of subsystem

Syntax

```
[checksum,details] = Simulink.SubSystem.getChecksum(subsys)
```

Description

`[checksum,details] = Simulink.SubSystem.getChecksum(subsys)` returns the checksum of the specified subsystem. Simulink software computes the checksum based on subsystem parameter settings and the blocks the subsystem contains.

One use of this command is to determine why code generated for a subsystem is not being reused. For an example, see "Determining Why Subsystem Code Is Not Reused" in the Real-Time Workshop documentation.

Note `Simulink.SubSystem.getChecksum` compiles the model that contains the specified subsystem, using the command *`model`*`([], [], [], 'compileForRTW')`, if the model is not already in a compiled state. To get the checksum for the model when Simulink software compiles it for simulation, use the command *`model`*`([], [], [], 'compile')` to place the model in a compiled state before using `Simulink.SubSystem.getChecksum`.

This command accepts the argument *`subsys`*, which is the full name or handle of the atomic subsystem block for which you are returning checksum data.

This command returns the following output:

- *`checksum`* — Structure of the form

  ```
  Value: [4x1 uint32]
  MarkedUnique: [bool]
  ```

  - `Value` — Array of four 32-bit integers that represents the subsystem's 128-bit checksum.

- `MarkedUnique` — True if the subsystem or the blocks it contains have properties that would prevent the code generated for the subsystem from being reused; otherwise, false.

- *`details`* — Structure of the form

```
ContentsChecksum: [1x1 struct]
InterfaceChecksum: [1x1 struct]
ContentsChecksumItems: [nx1 struct]
InterfaceChecksumItems: [mx1 struct]
```

  - `ContentsChecksum` — Structure of the same form as *`checksum`*, representing a checksum that provides information about all blocks in the system.
  - `InterfaceChecksum` — Structure of the same form as *`checksum`*, representing a checksum that provides information about the subsystem's block parameters and connections.
  - `ContentsChecksumItems` and `InterfaceChecksumItems` — Structure arrays of the following form that Simulink software uses to compute the checksum for `ContentsChecksum` and `InterfaceChecksum`, respectively:

```
Handle: [char array]
Identifier: [char array]
Value: [type]
```

    - `Handle` — Object for which Simulink software added an item to the checksum. For a block, the handle is a full block path. For a block port, the handle is the full block path and a string that identifies the port.
    - `Identifier` — Descriptor of the item Simulink software added to the checksum. If the item is a documented parameter, the identifier is the parameter name.
    - `Value` — Value of the item Simulink software added to the checksum. If the item is a parameter, `Value` is the value returned by

```
get_param(handle, identifier)
```

See Also

Simulink.BlockDiagram.getChecksum

sint

Purpose

Create MATLAB structure describing signed integer data type

Syntax

```
a = sint(TotalBits)
```

Description

`sint(TotalBits)` returns a MATLAB structure that describes the data type of a signed integer with a word size given by `TotalBits`.

`sint` is automatically called when a signed 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 signed integer data type:

```
a = sint(16)
a =
        Class: 'INT'
    IsSigned: 1
    MantBits: 16
```

See Also

`fixdt`, `float`, `sfix`, `sfrac`, `ufix`, `ufrac`, `uint`

Purpose

Build standalone and model reference targets

Syntax

```
slbuild('model')
slbuild('model', 'ModelReferenceSimTarget')
slbuild('model', 'ModelReferenceRTWTarget')
slbuild('model', 'ModelReferenceRTWTargetOnly')
```

Description

Note Except where noted, this command requires a Real-Time Workshop license.

`slbuild('model')` builds a standalone executable from *`model`*, using the model's Real-Time Workshop configuration settings.

Note The following commands honor the setting of the **Rebuild options** on the **Model Referencing** pane of the **Configuration Parameters** dialog for rebuilding the model reference target for this model and its referenced models.

`slbuild('model', 'ModelReferenceSimTarget')` builds a model reference simulation target for the model. This command does not require a Real-Time Workshop license.

`slbuild('model', 'ModelReferenceRTWTarget')` builds model reference simulation and Real-Time Workshop targets for *`model`*.

`slbuild('model', 'ModelReferenceRTWTargetOnly') builds a model reference RTW target for the model.`

If the **Rebuild option** on the **Model Referencing** pane of the **Configuration Parameters** dialog is set to `Never`, you can use two additional arguments, `'UpdateThisModelReferenceTarget'` and *`'Buildcond'`*, to specify a rebuild option for building a model reference target for this *`'model'`*. For example,

```
slbuild('model','ModelReferenceSimTarget', ...
        'UpdateThisModelReferenceTarget', Buildcond)
```

conditionally builds the simulation target for this *'model'* based on the value of *Buildcond*.

Note This option does not rebuild model reference targets for models referenced by this model.

'Buildcond' must be one of the following:

- `'IfOutOfDateOrStructuralChange'`

  Causes `slbuild` to rebuild this model if it detects any changes. This option is equivalent to the `If any changes detected` rebuild option on the **Model Referencing** pane of the **Configuration Parameters** dialog box.

- `'IfOutOfDate'`

  Causes `slbuild` to rebuild this model if it detects any changes in known dependencies of this model. This option is equivalent to the `If any changes in known dependencies detected` rebuild option on the **Model Referencing** pane of the **Configuration Parameters** dialog box.

- `'Force'`

  Causes `slbuild` to always rebuild the model. This option is equivalent to the `"Always"` rebuild option on the **Model Referencing** pane of the **Configuration Parameters** dialog box.

Purpose

Change 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 developed by IBM® Corp. and available on the Internet. 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

Start simulation in debug mode

Syntax

```
sldebug('sys')
```

Description

`sldebug('sys')` starts a simulation in debug mode. See "Simulink Debugger" in the Simulink documentation and Chapter 6, "Simulink Debugger Commands" in the Simulink Reference for information about using the debugger.

Examples

The following command:

```
sldebug('vdp')
```

loads the Simulink demo model `vdp` into memory and starts the simulation in debug mode. Alternatively, you can achieve the same result by using both the `sim` and `simset` commands:

```
sim('vdp', [0,10], simset('debug', 'on'))
```

See Also

`sim`, `simset`

sldiagnostics

Purpose

Display diagnostic information about Simulink system

Syntax

```
[txtRpt, sRpt] = sldiagnostics('sys')
[txtRpt, sRpt] = sldiagnostics('sys', options)
```

Description

`sldiagnostics('sys')` displays the following diagnostic information associated with the model or subsystem specified by `sys`:

- Number of each type of block
- Number of each type of Stateflow object
- Number of states, outputs, inputs, and sample times
- Names of libraries referenced and instances of the referenced blocks
- Time and additional memory used for each compilation phase of the root model

If the model specified by `sys` is not loaded, `sldiagnostics` loads the model, completes the diagnostics, and then closes the model. If `sys` is a subsystem, the root model must be loaded for `sldiagnostics` to operate successfully.

`sldiagnostics('sys', options)` displays only the diagnostic information associated with the specific operations listed as *`options`* strings. The available options and their output are as follows:

Option	Description
`CountBlocks`	Lists all unique blocks in the system and the number of occurrences of each. This includes blocks that are nested in masked subsystems or hidden blocks.
`CountSF`	Lists all unique Stateflow objects in the system and the number of occurrences of each.

Option	Description
`Sizes`	Lists the number of states, outputs, inputs, and sample times, as well as a flag indicating direct feedthrough, used in the root model.
`Libs`	Lists all unique libraries referenced in the root model, as well as the names and numbers of the library blocks.
`CompileStats`	Lists the time and additional memory used for each compilation phase of the root model. This information helps users troubleshoot model compilation speed and memory issues.
`Verbose`	Lists the results of the `CompileStats` diagnostic during the compilation phase. This is useful for diagnosing the compilation itself if it takes an unreasonable amount of time or hangs.
`RTWBuildStats`	Lists the same information as the `CompileStats` diagnostic. When issued with the second output argument `sRpt`, it captures the Real-Time Workshop build statistics in `sRpt.rtwbuild`.
`All`	Performs all diagnostics.

Note Running the `CompileStats` diagnostic before simulating a model for the first time will show greater memory usage. However, subsequent runs of the `CompileStats` diagnostic on the model will return a lesser amount of memory usage.

`[txtRpt, sRpt] = sldiagnostics('sys')` or `[txtRpt, sRpt] = sldiagnostics('sys', options)` returns the diagnostic information as a textual report `txtRpt` and a structure array `sRpt`, which contains the following fields that correspond to the diagnostic options:

- `blocks`
- `stateflow`
- `sizes`
- `links`
- `compilestats`
- `rtwbuild`

Examples

The following command counts and lists each type of block used in the `sldemo_bounce` model that comes with Simulink software.

```
sldiagnostics('sldemo_bounce', 'CountBlocks')
```

The following command counts and lists both the unique blocks and Stateflow objects used in the `sf_boiler` model that comes with Stateflow software; the textual report returned is captured as `myReport`.

```
myReport = sldiagnostics('sf_boiler', 'CountBlocks', 'CountSF')
```

The following commands open the `f14` model that comes with Simulink software, and counts the number of blocks used in the `Controller` subsystem.

```
f14; sldiagnostics('f14/Controller', 'CountBlocks')
```

The following command runs the `Sizes` and `CompileStats` diagnostics on the `f14` model, capturing the results as both a textual report and structure array.

```
[txtRpt, sRpt] = sldiagnostics('f14', 'Sizes', 'CompileStats')
```

See Also

`find_system`, `get_param`

Purpose

Discretize model that contains continuous blocks

Syntax

```
sldiscmdl('model_name',sample_time)
sldiscmdl('model_name',sample_time,method)
sldiscmdl('model_name',sample_time,options)
sldiscmdl('model_name',sample_time,method,freq)
sldiscmdl('model_name',sample_time,method,options)
sldiscmdl('model_name',sample_time,method,freq,options)
[old_blks,new_blks] = sldiscmdl('model_name',sample_time,
   method,freq,options)
```

Description

`sldiscmdl('model_name',sample_time)` discretizes the model named `'model_name'` using the specified *`sample_time`*. The model does not need to be open, and the units for *`sample_time`* are simulation seconds.

`sldiscmdl('model_name',sample_time,method)` discretizes the model using *`sample_time`* and the transform method specified by *`method`*.

`sldiscmdl('model_name',sample_time,options)` discretizes the model using *`sample_time`* and criteria specified by the *`options`* cell array. This array consists of four elements: {*`target`*, *`replace_with`*, *`put_into`*, *`prompt`*}.

`sldiscmdl('model_name',sample_time,method,freq)` discretizes the model using *`sample_time`*, *`method`*, and the critical frequency specified by *`freq`*. The units for *`freq`* are Hz. When you specify *`freq`*, *`method`* must be `'prewarp'`.

`sldiscmdl('model_name',sample_time,method,options)` discretizes the model using *`sample_time`*, *`method`*, and *`options`*.

`sldiscmdl('model_name',sample_time,method,freq,options)` discretizes the model using *`sample_time`*, *`method`*, *`freq`*, and *`options`*. When you specify *`freq`*, *`method`* must be `'prewarp'`.

`[old_blks,new_blks] = sldiscmdl('model_name',sample_time,method,freq,options)` discretizes the model using *`sample_time`*, *`method`*, *`freq`*, and *`options`*. When you specify *`freq`*, *`method`* must be `'prewarp'`. The function also

returns two cell arrays that contain full path names of the original, continuous blocks and the new, discretized blocks.

Inputs

`model_name`

Name of the model to discretize.

`sample_time`

Sample-time specification for the model:

Scalar value	Sample time with zero offset, such as `1`
Two-element vector	Sample time with nonzero offset, such as `[1 0.1]`

`method`

Method of converting blocks from continuous to discrete mode:

`'zoh'` (default)	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 single-input, single-output (SISO) systems only

`freq`

Critical frequency in Hz. This input applies only when the *`method`* input is `'prewarp'`.

`options`

Cell array {*target*, *replace_with*, *put_into*, *prompt*}, where each element can take the following values:

target	`'all'` (default)	Discretize all continuous blocks
	`'selected'`	Discretize only selected blocks in the model
	`'full_blk_path'`	Discretize specified block
replace_with	`'parammask'` (default)	Create discrete blocks whose parameters derive from the corresponding continuous blocks
	`'hardcoded'`	Create discrete blocks with hard-coded parameters placed directly into each block dialog box
put_into	`'copy'` (default)	Create discretization in a copy of the original model
	`'configurable'`	Create discretization candidate in a configurable subsystem
	`'current'`	Apply discretization to the current model
	`'untitled'`	Create discretization in a new untitled window
prompt	`'on'` (default)	Show discretization information at the command prompt
	`'off'`	Do not show discretization information at the command prompt

Examples

Discretize all continuous blocks in the `sldemo_f14` model using a 1-second sample time:

```
sldiscmdl('sldemo_f14',1);
```

Discretize the `Aircraft Dynamics Model` subsystem in the `sldemo_f14` model using a 1-second sample time, a 0.1-second offset, and a first-order hold transform method:

```
sldiscmdl('sldemo_f14',[1 0.1],'foh',...
{'sldemo_f14/Aircraft Dynamics Model',...
'parammask','copy','on'});
```

Discretize the `Aircraft Dynamics Model` subsystem in the `sldemo_f14` model and retrieve the full path name of the second discretized block:

```
[old_blks,new_blks] = sldiscmdl('sldemo_f14',[1 0.1],...
'foh',{'sldemo_f14/Aircraft Dynamics Model','parammask',...
'copy','on'});
% Get full path name of the second discretized block
new_blks{2}
```

See Also

`slmdldiscui`

How To

- "How to Discretize a Model from the MATLAB Command Window"

Purpose

Determine whether model has changed since it was loaded

Syntax

```
Changed = slIsFileChangedOnDisk(sys)
```

Description

`Changed = slIsFileChangedOnDisk(sys)` Returns true if the file which contains block diagram *`sys`* was changed on disk since the block diagram was loaded.

Example

To ensure that code is not generated for a model whose file has changed on disk since it was loaded, include the following in the `'entry'` section of the *`STF`*`_make_rtw_hook.m` file:

```
if (slIsFileChangedOnDisk(sys))
 error('File has changed on disk since it was loaded. Aborting code generation.');
end
```

See Also

"Customizing the Target Build Process with the STF_make_rtw Hook File"

"Model File Change Notification"

slmdldiscui

Purpose

Open Model Discretizer GUI

Syntax

```
slmdldiscui
slmdldiscui('name')
```

Description

`slmdldiscui` opens the Model Discretizer. A model does not need to be open.

`slmdldiscui('name')` opens the Model Discretizer for the model or library called `'name'`.

To use the Model Discretizer, you must have a Control System Toolbox license, version 5.2 or later.

Examples

Open the Model Discretizer for the `sldemo_f14` model:

```
slmdldiscui('sldemo_f14')
```

Open the Model Discretizer for the `discretizing` library:

```
slmdldiscui('discretizing')
```

See Also

`sldiscmdl`

How To

- "How to Discretize a Model from the Model Discretizer GUI"

Purpose

Replace Mux blocks used to create buses with Bus Creator blocks

Syntax

```
[muxes, uniqueMuxes, uniqueBds] = slreplace_mux(model)
[muxes, uniqueMuxes, uniqueBds] = slreplace_mux(model,
   reportonly)
```

Description

`[muxes, uniqueMuxes, uniqueBds] = slreplace_mux(model)` 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.

Before running this command in any form, you should set the **Mux blocks used to create bus signals** connectivity diagnostic to `warning` or `none`. See "Connectivity Diagnostics Overview" for more information.

`[muxes, uniqueMuxes, uniqueBds] = slreplace_mux(model, reportonly)` is equivalent to `[muxes, uniqueMuxes, uniqueBds] = slreplace_mux(model)` if *`reportonly`* is `true`.

If *`reportonly`* is `false`, the function reports all Mux blocks that create buses in `model` and in libraries referenced by `model`, and replaces all such Mux blocks with Bus Creator blocks. The function saves the model, if changed, and saves and closes any changed library. You should make a backup copy of your model and libraries before using this form of the command.

Inputs

`model`

The model for which `slreplace_mux` is to report and optionally replace muxes used as buses.

`reportOnly`

Whether to just generate a report (`true`) or also change the model(s) (`false`).

Default: `true`

Outputs

`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`

All models and libraries that use Mux blocks as Bus Creators

See Also

Bus Creator | Mux

How To

- "Connectivity Diagnostics Overview"
- "Mux Signals"
- "Using Composite Signals"
- "Avoiding Mux/Bus Mixtures"

Purpose

Replace blocks from previous releases with latest versions

Syntax

```
slupdate('sys')
slupdate('sys', prompt)
slupdate('sys', 'OperatingMode', 'Analyze')
```

Description

`slupdate('sys')` replaces blocks in model *sys* from a previous release of Simulink software with the latest versions.

Note Best practice is to first open the model before you call `slupdate`.

`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.

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.

- Updates obsolete configuration settings for the model.

`slupdate('sys', 'OperatingMode', 'Analyze')` performs only the analysis portion without updating or changing the model. This command analyzes referenced models, linked libraries, and S-functions, and then returns a data structure with the following fields:

- `Message` — string containing a message summarizing the results
- `blockList` — cell array listing blocks that need to be updated
- `blockReasons` — cell array listing reasons for updating the corresponding blocks
- `modelList` — cell array listing referenced models and the parent model
- `libraryList` — cell array listing non-MathWorks libraries referenced
- `configSetList` — for internal use
- `sfunList` — cell array listing S-functions referenced
- `sfunOK` — logical array representing S-function status, where `false` indicates that an S-function needs updating and `true` indicates otherwise
- `sfunType` — cell array listing apparent S-function type (e.g., m, mex)

Purpose

Find trim point of dynamic system

Syntax

```
[x,u,y,dx] = trim('sys')
[x,u,y,dx] = trim('sys',x0,u0,y0)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx)
[x,u,y,dx,options] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,
   options)
[x,u,y,dx,options] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,
   options,t)
```

Description

A trim point, also known as an equilibrium point, is a point in the parameter space of a dynamic system at which the system is in a steady state. For example, a trim point of an aircraft is a setting of its controls that causes the aircraft to fly straight and level. Mathematically, a trim point is a point where the system's state derivatives equal zero. `trim` starts from an initial point and searches, using a sequential quadratic programming algorithm, until it finds the nearest trim point. You must supply the initial point implicitly or explicitly. If `trim` cannot find a trim point, it returns the point encountered in its search where the state derivatives are closest to zero in a min-max sense; that is, it returns the point that minimizes the maximum deviation from zero of the derivatives. `trim` can find trim points that meet specific input, output, or state conditions, and it can find points where a system is changing in a specified manner, that is, points where the system's state derivatives equal specific nonzero values.

`[x,u,y,dx] = trim('sys')` finds the equilibrium point nearest to the system's initial state, `x0`. Specifically, `trim` finds the equilibrium point that minimizes the maximum absolute value of `[x-x0,u,y]`. If `trim` cannot find an equilibrium point near the system's initial state, it returns the point at which the system is nearest to equilibrium. Specifically, it returns the point that minimizes `abs(dx-0)`. You can obtain `x0` using this command.

```
[sizes,x0,xstr] = sys([],[],[],0)
```

`[x,u,y,dx] = trim('sys',x0,u0,y0)` finds the trim point nearest to `x0`, `u0`, `y0`, that is, the point that minimizes the maximum value of

```
abs([x-x0; u-u0; y-y0])
```

`[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy)` finds the trim point closest to `x0`, `u0`, `y0` that satisfies a specified set of state, input, and/or output conditions. The integer vectors `ix`, `iu`, and `iy` select the values in `x0`, `u0`, and `y0` that must be satisfied. If `trim` cannot find an equilibrium point that satisfies the specified set of conditions exactly, it returns the nearest point that satisfies the conditions, namely,

```
abs([x(ix)-x0(ix); u(iu)-u0(iu); y(iy)-y0(iy)])
```

`[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx)` finds specific nonequilibrium points, that is, points at which the system's state derivatives have some specified nonzero value. Here, `dx0` specifies the state derivative values at the search's starting point and `idx` selects the values in `dx0` that the search must satisfy exactly.

`[x,u,y,dx,options] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options)` specifies an array of optimization parameters that `trim` passes to the optimization function that it uses to find trim points. The optimization function, in turn, uses this array to control the optimization process and to return information about the process. `trim` returns the `options` array at the end of the search process. By exposing the underlying optimization process in this way, `trim` allows you to monitor and fine-tune the search for trim points.

The following table describes how each element affects the search for a trim point. Array elements 1, 2, 3, 4, and 10 are particularly useful for finding trim points.

No.	Default	Description
1	0	Specifies display options. 0 specifies no display; 1 specifies tabular output; -1 suppresses warning messages.

No.	Default	Description
2	10^{-4}	Precision the computed trim point must attain to terminate the search.
3	10^{-4}	Precision the trim search goal function must attain to terminate the search.
4	10^{-6}	Precision the state derivatives must attain to terminate the search.
5	N/A	Not used.
6	N/A	Not used.
7	N/A	Used internally.
8	N/A	Returns the value of the trim search goal function (λ in goal attainment).
9	N/A	Not used.
10	N/A	Returns the number of iterations used to find a trim point.
11	N/A	Returns the number of function gradient evaluations.
12	0	Not used.
13	0	Number of equality constraints.
14	100*(Number of variables)	Maximum number of function evaluations to use to find a trim point.
15	N/A	Not used.
16	10^{-8}	Used internally.
17	0.1	Used internally.
18	N/A	Returns the step length.

`[x,u,y,dx,options] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options,t)` sets the time to `t` if the system is dependent on time.

Examples

Consider a linear state-space model

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

The A, B, C, and D matrices are as follows in a system called `sys`.

```
A = [-0.09 -0.01;  1   0];
B = [ 0    -7;     0  -2];
C = [ 0     2;     1  -5];
D = [-3     0;     1   0];
```

Example 1

To find an equilibrium point, use

```
[x,u,y,dx,options] = trim('sys')
x =
   0
   0
u =
   0
y =
   0
   0
dx =
   0
   0
```

The number of iterations taken is

```
options(10)
ans =
```

```
       7
```

Example 2

To find an equilibrium point near `x = [1;1], u = [1;1]`, enter

```
x0 = [1;1];
u0 = [1;1];
[x,u,y,dx,options] = trim('sys', x0, u0);
x =
    1.0e-11 *
   -0.1167
   -0.1167
u =
    0.3333
    0.0000
y =
   -1.0000
    0.3333
dx =
    1.0e-11 *
    0.4214
    0.0003
```

The number of iterations taken is

```
options(10)
ans =
      25
```

Example 3

To find an equilibrium point with the outputs fixed to `1`, use

```
y = [1;1];
iy = [1;2];
[x,u,y,dx] = trim('sys', [], [], y, [], [], iy)
x =
```

```
    0.0009
   -0.3075
u =
   -0.5383
    0.0004
y =
    1.0000
    1.0000
dx =
    1.0e-16 *
   -0.0173
    0.2396
```

Example 4

To find an equilibrium point with the outputs fixed to 1 and the derivatives set to 0 and 1, use

```
y = [1;1];
iy = [1;2];
dx = [0;1];
idx = [1;2];
[x,u,y,dx,options] = trim('sys',[],[],y,[],[],iy,dx,idx)
x =
    0.9752
   -0.0827
u =
   -0.3884
   -0.0124
y =
    1.0000
    1.0000
dx =
    0.0000
    1.0000
```

The number of iterations taken is

```
options(10)
ans =
      13
```

Limitations

The trim point found by `trim` starting from any given initial point is only a local value. Other, more suitable trim points may exist. Thus, if you want to find the most suitable trim point for a particular application, it is important to try a number of initial guesses for `x`, `u`, and `y`.

Algorithm

`trim` uses a sequential quadratic programming algorithm to find trim points. See the *Optimization Toolbox™ User's Guide* for a description of this algorithm.

tunablevars2parameterobjects

Purpose

Create Simulink parameter objects from tunable parameters

Syntax

```
tunablevars2parameterobjects (modelName)
tunablevars2parameterobjects (modelName, class)
```

Description

`tunablevars2parameterobjects (modelName)` creates `Simulink.Parameter` objects in the base workspace for the variables listed in the specified model's Tunable Parameters dialog, then deletes the source information from the dialog. To preserve the information, save the resulting Simulink parameter objects into a MAT-file.

If a tunable variable is already defined as a numeric variable in the base workspace, the variable will be replaced by a parameter object and the original variable will be copied to the object's Value property.

If a tunable variable is already defined as a Simulink parameter object, the object will not be modified but the information for the variable will still be deleted from the Tunable Parameters dialog.

If a tunable variable is defined as any other class of variable, the variable will not be modified and the information for the variable will not be deleted from the Tunable Parameters dialog.

`tunablevars2parameterobjects (modelName, class)` creates objects of the specified class rather than `Simulink.Parameter` objects.

Inputs

`modelName`

Model name or handle

`class`

Parameter class to use for creating objects

Default: `Simulink.Parameter`

See Also

`Simulink.Parameter`

How To

- "Tunable Parameters"

Purpose

Create MATLAB structure describing unsigned generalized fixed-point data type

Syntax

```
a = ufix(TotalBits)
```

Description

`ufix(TotalBits)` returns a MATLAB structure that describes the data type of an unsigned generalized fixed-point data type with a word size given by `TotalBits`.

`ufix` is automatically called when an unsigned generalized fixed-point data type is specified in a block dialog box.

Note The default binary point is not included in this data type description. Instead, the scaling must be explicitly defined in the block dialog box.

Examples

Define a 16-bit unsigned generalized fixed-point data type:

```
a = ufix(16)
a =
       Class: 'FIX'
    IsSigned: 0
    MantBits: 16
```

See Also

`fixdt`, `float`, `sfix`, `sfrac`, `sint`, `ufrac`, `uint`

ufrac

Purpose

Create MATLAB structure describing unsigned fractional data type

Syntax

```
a = ufrac(TotalBits)
a = ufrac(TotalBits, GuardBits)
```

Description

`ufrac(TotalBits)` returns a MATLAB structure that describes the data type of an unsigned fractional number with a word size given by `TotalBits`.

`ufrac(TotalBits, GuardBits)` returns a MATLAB structure that describes the data type of an unsigned fractional number. The total word size is given by `TotalBits` with `GuardBits` bits located to the left of the binary point.

`ufrac` is automatically called when an unsigned fractional data type is specified in a block dialog box.

The default binary point for this data type is assumed to lie immediately to the left of all bits. If guard bits are specified, then they lie to the left the default binary point.

Examples

Define an 8-bit unsigned fractional data type with 4 guard bits. Note that the range of this number is from 0 to $(1 - 2^{-8}).2^4 = 15.9375$:

```
a = ufrac(8,4)
a =
        Class: 'FRAC'
     IsSigned: 0
     MantBits: 8
    GuardBits: 4
```

See Also

`fixdt`, `float`, `sfix`, `sfrac`, `sint`, `ufix`, `uint`

Purpose

Create MATLAB structure describing 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`

unpack

Purpose

Extract signal logging objects from signal logs and write them into MATLAB workspace

Syntax

```
log.unpack
tsarray.unpack
log.unpack('systems')
log.unpack('all')
```

Description

`log.unpack` or `unpack(log)` extracts the top level elements of the `Simulink.ModelDataLogs` or `Simulink.SubsysDataLogs` object named **`log`** (e.g., `logsout`).

`log.unpack('systems')` or `unpack(`*`log`*`, 'systems')` extracts `Simulink.Timeseries` and `Simulink.TsArray` objects from the `Simulink.ModelDataLogs` or `Simulink.SubsysDataLogs` object named `log` . This command does not extract `Simulink.Timeseries` objects from `Simulink.TsArray` objects nor does it write intermediate `Simulink.ModelDataLogs` or `Simulink.SubsysDataLogs` objects to the MATLAB workspace.

`log.unpack('all')` or `unpack(`*`log`*`, 'all')` extracts all the `Simulink.Timeseries` objects contained by the `Simulink.ModelDataLogs`, `Simulink.TsArray`, or `Simulink.SubsysDataLogs` object named `log`.

`tsarray.unpack` extracts the time-series objects of class `Simulink.Timeseries` from the `Simulink.TsArray` object named `tsarray`.

See Also

"Importing and Exporting Data", `Simulink.ModelDataLogs`, `Simulink.SubsysDataLogs`, `Simulink.ScopeDataLogs`, `Simulink.Timeseries`, `Simulink.TsArray`, `who`, `whos`

Purpose

Display graph of model reference dependencies

Syntax

```
view_mdlrefs(modelName)
```

Description

`view_mdlrefs(modelName)` launches the Model Dependency Viewer, which displays a graph of model reference dependencies for the model specified by *`modelName`*. The nodes in the graph represent Simulink models. The directed lines indicate model dependencies.

The default display omits library blocks. You could see this same display by opening *`modelName`* and choosing **Tools > Model Dependencies > Model Dependency Viewer > .mdl File Dependencies Excluding Libraries** from the model menu. Use **View > Dependency Type** to see other dependency displays.

The Model Dependency Viewer is the same tool, and provides the same options, whether you launch it by typing `view_mdlrefs(`*`modelName`* or by using the Simulink GUI. To see a demo of the Model Dependency Viewer, type `sldemo_mdlref_depgraph` in the MATLAB Command Window.

See Also

Model | `find_mdlrefs`

Tutorials

- `sldemo_mdlref_depgraph`

How To

- "Referencing a Model"
- "Using the Model Dependency Viewer"

who

Purpose

List names of top-level data logging objects in Simulink data log

Syntax

```
log.who
tsarray.who
log.who('systems')
log.who('all')
```

Description

`log.who` or `who(log)` lists the names of the top-level signal logging objects contained by `log`, where `log` is the handle of a `Simulink.ModelDataLogs` object name.

`tsarray.who` or `who(tsarray)` lists the names of `Simulink.TimeSeries` objects contained by the `Simulink.TsArray` object named `tsarray`.

`log.who('systems')` or `who(log, 'systems')` lists the names of all signal logging objects contained by `log` except for `Simulink.Timeseries` objects stored in `Simulink.TsArray` objects contained by `log`.

`log.who('all')` or `who(log, 'all')` lists the names of all the `Simulink.Timeseries` objects contained by the `Simulink.ModelDataLogs`, `Simulink.TsArray`, or `Simulink.SubsysDataLogs` object named `log`.

For information about other uses of `who`, execute help whoin the MATLAB Command Window.

See Also

"Importing and Exporting Data", `Simulink.ModelDataLogs`, `Simulink.SubsysDataLogs`, `Simulink.ScopeDataLogs`, `Simulink.Timeseries`, `Simulink.TsArray`, `whos`, `unpack`

Purpose

List names and types of top-level data logging objects in Simulink data log

Syntax

```
log.whos
tsarray.whos
log.whos('systems')
log.whos('all')
```

Description

`log.whos` or `whos(log)` lists the names and types of the top-level signal logging objects contained by `log`, where `log` is the handle of a `Simulink.ModelDataLogs` object name.

`tsarray.whos` or `whos(tsarray)` lists the names and types of `Simulink.TimeSeries` objects contained by the `Simulink.TsArray` object named `tsarray`.

`log.whos('systems')` or `whos(log, 'systems')` lists the names and types of all signal logging objects contained by `log` except for `Simulink.Timeseries` objects stored in `Simulink.TsArray` objects contained by `log`.

`log.whos('all')` or `whos(log, 'all')` lists the names and types of all the `Simulink.Timeseries` objects contained by the `Simulink.ModelDataLogs`, `Simulink.TsArray`, or `Simulink.SubsysDataLogs` object named `log`.

For information about other uses of `whos`, execute help whos in the MATLAB Command Window.

See Also

"Importing and Exporting Data", `Simulink.ModelDataLogs`, `Simulink.SubsysDataLogs`, `Simulink.ScopeDataLogs`, `Simulink.Timeseries`, `Simulink.TsArray`, `who`, `unpack`

5

Mask Icon Drawing Commands

color	Change drawing color of subsequent mask icon drawing commands
disp	Display text on icon of masked subsystem
dpoly, droots	Display transfer function or zero-pole representation on icon of masked subsystem
fprintf	Display variable text centered on icon of masked subsystem
image	Display image on icon of masked subsystem
patch	Draw color patch of specified shape on icon of masked subsystem
plot	Draw graph connecting series of points
port_label	Draw port label on icon of masked subsystem
text	Display text at specific location on icon of masked subsystem

color

Purpose

Change drawing color of subsequent mask icon drawing commands

Syntax

```
color(colorstr)
```

Description

`color(colorstr)` sets the drawing color of all subsequent mask drawing commands to the color specified by the string `colorstr`.

`colorstr` must be one of the following supported color strings.

```
blue
green
red
cyan
magenta
yellow
black
```

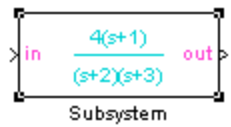
Entering any other string or specifying the color using RGB values results in a warning at the MATLAB command prompt and the color change is ignored. The specified drawing color does not influence the color used by the `patch` or `image` drawing commands.

Examples

The following commands

```
color('cyan');
droots([-1], [-2 -3], 4)
color('magenta')
port_label('input',1,'in')
port_label('output',1,'out')
```

draw the following mask icon.



See Also droots, port_label

disp

Purpose

Display text on icon of masked subsystem

Syntax

```
disp(text)
disp(text, 'texmode', 'on')
```

Description

`disp(text)` displays *`text`* centered on the icon where *`text`* is any MATLAB expression that evaluates to a string.

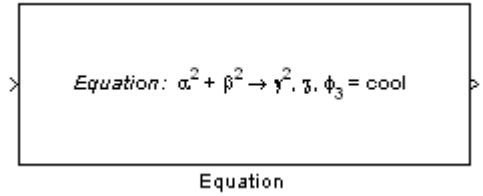
`disp``(text, 'texmode', 'on')` allows you to use TeX formatting commands in *`text`*. The TeX formatting commands in turn allow you to include symbols and Greek letters in icon text. See "Mathematical Symbols, Greek Letters, and TEX Characters" in the MATLAB documentation for information on the TeX formatting commands supported by Simulink software.

Examples

The following command

```
disp('{\itEquation:}  \alpha^2 + \beta^2 \rightarrow \gamma^2,
\chi, \phi_3 = {\bfcool}', 'texmode','on')
```

draws the equation that appears on this masked block icon.



See Also

`fprintf`, `port_label`, `text`

Purpose

Display transfer function or zero-pole representation on icon of masked subsystem

Syntax

```
dpoly(num, den)
dpoly(num, den, 'character')

droots(zero, pole, gain)
droots(zero, pole, gain,'z')
droots(zero, pole, gain,'z-')
```

Description

`dpoly(num, den)` displays the transfer function whose numerator is *num* and denominator is `den`.

`dpoly(num, den, 'character')` allows you to specify the name of the transfer function's independent variable. The default is *s*.

`droots(zero, pole, gain)` displays the transfer function whose zero is a *zero*, pole is *pole*, and gain is *gain*.

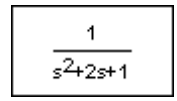
`droots(zero, pole, gain,'z')` and `droots(zero, pole, gain,'z-')` express the equation in terms of *z* or *1/z*.

When the icon is drawn, the initialization commands are executed and the resulting equation is drawn on the icon:

- To display a continuous transfer function in descending powers of *s*, enter

  ```
  dpoly(num, den)
  ```

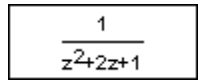
  For example, for `num = [0 0 1];` and `den = [1 2 1]` the icon looks like this:

- To display a discrete transfer function in descending powers of *z*, enter

```
dpoly(num, den, 'z')
```

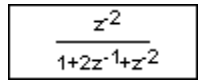
For example, for `num = [0 0 1];` and `den = [1 2 1];` the icon looks like this:



- To display a discrete transfer function in ascending powers of *1/z*, enter

  ```
  dpoly(num, den, 'z-')
  ```

  For example, for `num` and `den` as defined previously, the icon looks like this:

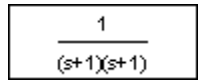
  

- To display a zero-pole gain transfer function, enter

  ```
  droots(z, p, k)
  ```

  For example, the preceding command creates this icon for these values:

  ```
  z = []; p = [-1 -1]; k = 1;
  ```

If the parameters are not defined or have no values when you create the icon, Simulink software displays three question marks (`? ? ?`) in the icon. When the parameter values are entered in the mask dialog box, Simulink software evaluates the transfer function and displays the resulting equation in the icon.

See Also disp, port_label, text

fprintf

Purpose

Display variable text centered on icon of masked subsystem

Syntax

```
fprintf(text)
fprintf(format, var)
```

Description

The `fprintf` command displays formatted text centered on the icon and can display *`format`* along with the contents of *`var`*.

Note While this command is identical in name to its corresponding MATLAB function, it provides only the functionality described above.

Examples

This command

```
fprintf('Hello');
```

displays the string `'Hello'` on the icon.

This command

```
fprintf('Juhi = %d',17);
```

uses the decimal notation format (`%d`) to display the variable `17`.

See Also

`disp`, `port_label`, `text`

Purpose

Display image on icon of masked subsystem

Syntax

```
image(a)
image(a, position)
image(a, position, rotation)
```

Description

`image(a)` displays the image `a`, where `a` is an M-by-N-by-3 array of RGB values. You can use the MATLAB commands `imread` and `ind2rgb` to read and convert bitmap files (such as GIF) to the necessary matrix format.

`image(a, position)` creates the image at the specified position:

Position	Description
[*x*, *y*, *w*, *h*]	Position (*x*, *y*) and size (*w*, *h*) of the image where the position is relative to the lower-left corner of the mask. The image is scaled to fit the specified size.
`'center'`	Center of the mask
`'top-left'`	Top left corner of the mask, unscaled
`'bottom-left'`	Bottom left corner of the mask, unscaled
`'top-right'`	Top right corner of the mask, unscaled
`'bottom-right'`	Bottom right corner of the mask, unscaled

`image(a, position, rotation)` allows you to specify whether the image rotates (`'on'`) or remains stationary (`'off'`) as the icon rotates. The default is `'off'`.

Examples

This command

```
image(imread('icon.tif'))
```

reads the icon image from a TIFF file named `icon.tif` in the MATLAB path.

The following commands read and convert a GIF file, `label.gif`, to the appropriate matrix format. You can type these commands in the **Initialization** pane of the Mask Editor.

```
[data, map]=imread('label.gif');
pic=ind2rgb(data,map);
```

Then type the command

```
image(pic)
```

in the **Icon** pane of the Mask Editor to read the converted label image.

See Also

`patch`, `plot`

Purpose

Draw color patch of specified shape on icon of masked subsystem

Syntax

```
patch(x, y)
patch(x, y, [r g b])
```

Description

`patch(x, y)` creates a solid patch having the shape specified by the coordinate vectors `x` and `y`. The patch's color is the current foreground color.

`patch(x, y, [r g b])` creates a solid patch of the color specified by the vector `[r g b]`, where `r` is the red component, `g` the green, and `b` the blue. For example,

```
patch([0 .5 1], [0 1 0], [1 0 0])
```

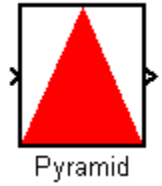
creates a red triangle on the mask's icon.

Examples

This command

```
patch([0 .5 1], [0 1 0], [1 0 0])
```

creates a red triangle on the mask's icon.



See Also

`image`, `plot`

Purpose

Draw graph connecting 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 software 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.

Simulink software 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

port_label

Purpose

Draw port label on icon of 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. Valid values for *`port_type`* include the following:

Value	Description
`input`	Simulink input port
`output`	Simulink output port
`lconn`	Physical Modeling connection port on the left side of a masked subsystem
`rconn`	Physical Modeling connection port on the right side of a masked subsystem

The input argument *`port_number`* is an integer, and *`label`* is a string specifying the port's label.

Note Physical Modeling port labels are assigned based on the nominal port location. If the masked subsystem has been rotated or flipped, for example, a port labeled using `'lconn'` as the `port_type` may not appear on the left side of the block.

`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 software.

Examples

The command

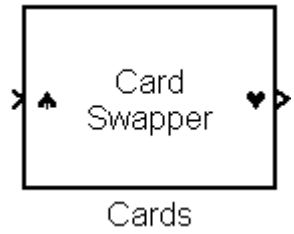
```
port_label('input', 1, 'a')
```

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 specific location on icon of 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.

`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 "Mathematical Symbols, Greek Letters, and TEX Characters" in the MATLAB documentation for information on the TeX formatting commands supported by Simulink software.

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

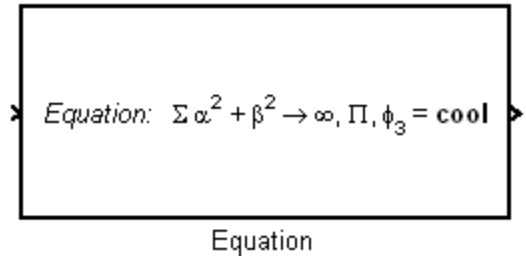
```
text(0.5, 0.5, 'foobar', 'horizontalAlignment', 'center')
```

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`

6

Simulink Debugger Commands

animate	Enable or disable animation mode
ashow	Show algebraic loop
atrace	Set algebraic loop trace level
bafter	Insert breakpoint after specified method
break	Insert breakpoint before specified method
bshow	Show specified block
clear	Clear breakpoints from model
continue	Continue simulation
disp	Display block's I/O when simulation stops
ebreak	Enable (or disable) breakpoint on solver errors
elist	List simulation methods in order in which they are executed during simulation
emode	Toggle model execution between accelerated and normal mode
etrace	Enable or disable method tracing
help	Display help for debugger commands

nanbreak	Set or clear nonfinite value break mode
next	Advance simulation to start of next method at current level in model's execution list
probe	I/O and state data for blocks
quit	Stop simulation debugger
rbreak	Break simulation before solver reset
run	Run simulation to completion
slist	Sorted list of model blocks
states	Current state values
status	Debugging options in effect
step	Advance simulation by one or more methods
stop	Stop simulation
strace	Set solver trace level
systems	List nonvirtual systems of model
tbreak	Set or clear time breakpoint
trace	Display block's I/O each time block executes
undisp	Remove block from debugger's list of display points
untrace	Remove block from debugger's list of trace points
where	Display current location of simulation in simulation loop
xbreak	Break when debugger encounters step-size-limiting state

zcbreak	Toggle breaking at nonsampled zero-crossing events
zclist	List blocks containing nonsampled zero crossings

animate

Purpose

Enable or disable animation mode

Syntax

```
animate [delay | stop]
```

Short Form

```
ani
```

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`

Purpose

Show algebraic loop

Syntax

```
ashow <gcb | s:b | s#n | clear>
```

Short Form

```
as
```

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
```

Short Form

```
at
```

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`

Purpose

Insert breakpoint after 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]
```

Short Form

```
ba
```

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.

`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`

Purpose

Insert breakpoint before 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]
```

Short Form

```
b
```

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.

`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`

Purpose

Show specified block

Syntax

```
bshow s:b
```

Short Form

```
bs
```

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`

clear

Purpose

Clear breakpoints from model

Syntax

```
clear
clear m:mid
clear id
clear <sid:bid | gcb>
```

Short Form

```
cl
```

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`

Purpose

Continue simulation

Syntax

```
continue
```

Short Form

```
c
```

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`

disp

Purpose

Display block's I/O when simulation stops

Syntax

```
disp
disp gcb
disp s:b
```

Short Form

```
d
```

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`

Purpose

Enable (or disable) breakpoint on solver errors

Syntax

```
ebreak
```

Short Form

```
eb
```

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 order in which they are executed during simulation

Syntax

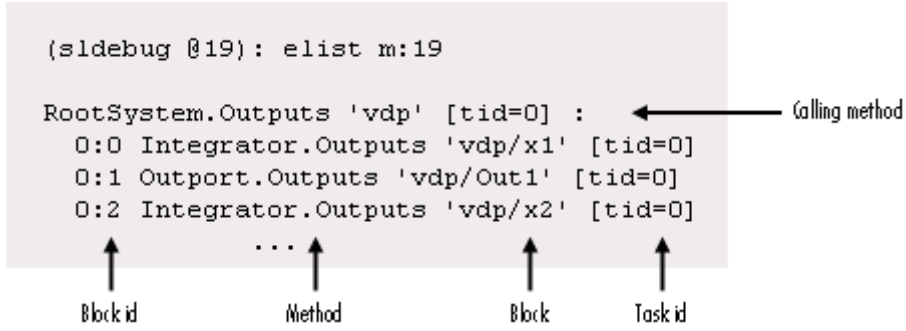
```
elist m:mid [tid:TID]
elist <gcs | s:sid> [mth] [tid:TID]
elist <gcb | sid:bid> [mth] [tid:TID]
```

Short Form

```
el
```

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 that it calls in the order in which they are invoked. The entry for the calling method includes

- The name of the method

  The name of the method is prefixed by the type of system that defines the method, e.g., RootSystem.

- The name of the model or subsystem instance on which the method is invoked
- The ID of the task that invokes the method

The entry for each called method includes

- The ID (*sid*:*bid*) of the block instance on which the method is invoked

  The block ID is prefixed by a number specifying the system that contains the block (the *sid*). This allows Simulink software to assign the same block ID to blocks residing in different subsystems.

- The name of the method

  The method name is prefixed with the type of block that defines the method, e.g., Integrator.

- The name of the block instance on which the method is invoked

- The task that invokes the method

The optional task ID parameter (**tid:***TID*) allows you to restrict the displayed lists to methods invoked for a specified task. You can specify this option only for system or atomic subsystem methods that invoke Outputs or Update methods.

`elist <`**`gcs`** `|` **`s:`**`sid>` lists the methods executed for the currently selected system (specified by the `gcs` command) or the system or nonvirtual subsystem specified by the system ID *sid*, e.g.,

```
(sldebug @19): elist gcs

RootSystem.Start 'vdp':
  0:0 Integrator.Start 'vdp/x1'
  0:2 Integrator.Start 'vdp/x2'
  0:4 Scope.Start 'vdp/Scope'
  0:5 Fcn.Start 'vdp/Fcn'
  0:6 Product.Start 'vdp/Product'
  0:7 Gain.Start 'vdp/Mu'
  0:8 Sum.Start 'vdp/Sum'

RootSystem.Initialize 'vdp':
  0:0 Integrator.Initialize 'vdp/x1'
            ...
```

The system ID of a model's root system is 0. You can use the debugger's `systems` command to determine the system IDs of a model's subsystems.

Note The `elist` and `where` commands use block IDs to identify subsystems in their output. The block ID for a subsystem is not the same as the system ID displayed by the `systems` command. Use the `elist` *`sid`*`:`*`bid`* form of the `elist` command to display the methods of a subsystem whose block ID appears in the output of a previous invocation of the `elist` or `where` command.

`elist <`**`gcs`** | **`s:`**`sid> `*`mth`* lists methods of type *`mth`* to be executed for the system specified by the `gcs` command or the system ID *`sid`*, e.g.,

```
(sldebug @19): elist gcs Start

RootSystem.Start 'vdp':
  0:0 Integrator.Start 'vdp/x1'
  0:2 Integrator.Start 'vdp/x2'
  0:4 Scope.Start 'vdp/Scope'
  0:5 Fcn.Start 'vdp/Fcn'
  0:6 Product.Start 'vdp/Product'
  0:7 Gain.Start 'vdp/Mu'
  0:8 Sum.Start 'vdp/Sum'
            ...
```

Use `elist` **`gcb`** to list the methods invoked by the nonvirtual subsystem currently selected in the model.

See Also

`where`, `slist`, `systems`

emode

Purpose

Toggle model execution between accelerated and normal mode

Syntax

```
emode
```

Short Form

```
em
```

Description

Toggles the simulation between accelerated and normal mode when using the Accelerator mode in Simulink software. See “Using the Accelerator Mode with the Simulink Debugger” in “Using Simulink” for more information.

Purpose

Enable or disable method tracing

Syntax

```
etrace level level-number
```

Short Form

```
et
```

Description

This command enables or disables method tracing, depending on the value of `level`:

Level	Description
0	Turn tracing off.
1	Trace model methods.
2	Trace model and system methods.
3	Trace model, system, and block methods.

When method tracing is on, the debugger prints a message at the command line every time a method of the specified level is entered or exited. The message specifies the current simulation time, whether the simulation is entering or exiting the method, the method id and name, and the name of the model, system, or block to which the method belongs.

See Also

`elist`, `where`, `trace`

help

Purpose

Display help for debugger commands

Syntax

```
help
```

Short Form

```
? or h
```

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

Set or clear nonfinite value break mode

Syntax

```
nanbreak
```

Short Form

```
na
```

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`

next

Purpose

Advance simulation to start of next method at current level in model's execution list

Syntax

```
next
```

Short Form

```
n
```

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`

Purpose

I/O and state data for blocks

Syntax

```
probe
probe s:b
probe gcb
probe level level-type
p
```

Description

`probe` sets the Simulink debugger to interactive probe mode. In this mode, the debugger displays the I/O of a selected block. To exit interactive probe mode, enter a debugger command or press the **Enter** key.

`probe s:b` displays the I/O of the block whose system index is *s* and block index is *b*.

`probe gcb` displays the I/O of the currently selected block.

`probe level level-type` sets the verbosity level for `probe`, `trace`, and `dis`. If *level-type* is `io`, the debugger displays block I/O. If *level-type* is `all` (default), the debugger displays all information for the current state of a block, including inputs and outputs, states, and zero crossings.

`p` is the short form of the command.

Examples

Display I/O for the currently selected block `Out2` in the model `vdp` using the Simulink debugger.

1. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2. Enter:

   ```
   probe gcb
   ```

The MATLAB Command Window displays:

```
probe: Data of 0:3 Outport block 'vdp/Out2':
U1     = [0]
```

See Also

`disp` | `trace`

Purpose

Stop simulation debugger

Syntax

```
quit
q
```

Description

`quit` stops the Simulink debugger and returns to the MATLAB command prompt.

`q` is the short form of the command.

Examples

Start the Simulink debugger for the model `vdp` and then stop it.

1. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2. Enter:

   ```
   quit
   ```

See Also

`stop`

rbreak

Purpose

Break simulation before solver reset

Syntax

```
rbreak
rb
```

Description

`rbreak` 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 requires a solver reset. The halt occurs before the solver resets.

`rb` is the short form of the command.

Examples

Start Simulink debugger for the model `vdp` and a set breakpoint before a solver reset.

1. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> is replaced with the Simulink debugger prompt (sldebug @0): >>.

2. Enter:

   ```
   rbreak
   ```

   The MATLAB Command Window displays:

   ```
   Break on solver reset request              : enabled
   ```

See Also

`break` | `bafter` | `nanbreak` | `ebreak` | `tbreak` | `xbreak` | `zcbreak`

Purpose

Run simulation to completion

Syntax

```
run
r
```

Description

`run` starts the simulation from the current breakpoint to its final time step. It ignores breakpoints and display points.

`r` is the short form of the command

Examples

Continue the simulation for the model `vdp` using the Simulink debugger.

1. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2. Enter:

   ```
   run
   ```

See Also

`continue` | `stop` | `quit`

slist

Purpose

Sorted list of model blocks

Syntax

```
slist
sli
```

Description

`slist` displays a list of blocks for the root system and each nonvirtual subsystem sorted according to data dependencies and other criteria.

For each system (root or nonvirtual), `slist` displays:

- Title line specifying 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.
- Entry for each block in the order in which the blocks appear in the sorted list.

For each block entry, `slist` displays the block 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`).

- Block index is the position of the block in the sorted list.
- System index is the order in which the Simulink software generated the system 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.

Simulink software uses sorted lists to create block method execution lists (see `elist`) for root system and nonvirtual subsystem methods. In general, root system and nonvirtual subsystem methods invoke the block methods in the same order as the blocks appear in the sorted list.

Exceptions occur in the execution order of block methods. For example, execution lists for multicast models group all blocks operating at the same rate and in the same task together. Slower groups appeare later than faster groups. The grouping of methods by task can result in a block method execution order that is different from the block sorted

order. However, within groups, methods execute in the same order as the corresponding blocks appear in the sorted list.

`sli` is the short form of the command.

Examples

Display a sorted list of the root system in the `vdp` model using the Simulink debugger.

1. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2. Enter:

   ```
   slist
   ```

   The MATLAB Command Window displays:

   ```
   ---- 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)
   ```

See Also

`systems | elist`

Purpose

Current state values

Syntax

```
states
```

Description

`states` displays a list of the current states of the model. The list includes the index, current value, system:block:element ID, state vector name, and block name for each state.

Examples

Display information about the states for the `vdp` model.

1. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2. Enter:

   ```
   states
   ```

   The MATLAB Command Window displays:

   ```
   Continuous States:
   Idx  Value                    (system:block:element  Name   'BlockNa
     0  0                        (0:0:0  CSTATE  'vdp/x1')
     1  0                        (0:2:0  CSTATE  'vdp/x2')
   ```

Purpose

Debugging options in effect

Syntax

```
status
```

Description

`status` displays a list of the debugging options in effect.

Examples

Display status for the model `vdp` using the Simulink debugger.

1. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2. Enter:

   ```
   status
   ```

   The MATLAB Command Window displays:

   ```
   %------------------------------------------------------------------
   Current simulation time                    : 0 (MajorTimeStep)
   Solver needs reset                         : no
   Solver derivatives cache needs reset       : no
   Zero crossing signals cache needs reset    : no
   Default command to execute on return/enter : ""
   Break at zero crossing events              : disabled
   Break on solver error                      : disabled
   Break on failed integration step           : disabled
   Time break point                           : disabled
   Break on non-finite (NaN,Inf) values       : disabled
   Break on solver reset request              : disabled
   Display level for disp, trace, probe       : 1 (i/o, states)
   Solver trace level                         : 0
   Algebraic loop tracing level               : 0
   Animation Mode                             : off
   Window reuse                               : not supported
   ```

```
Execution Mode                          : Normal
Display level for etrace                : 0 (disabled)
Break points                            : none installed
Display points                          : none installed
```

Purpose

Advance simulation by one or more methods

Syntax

```
step
step in
step over
step out
step top
step blockmth
s
```

Description

`step` or `step in` advances the simulation to the next method in the current time step.

`step over` advances the simulation over the next method.

`step out` advances the simulation the end of the current simulation point hierarchy.

`step top` advances the simulation to the first method executed in the next time step.

`step blockmth` advances the simulation to the next method that operates on a block.

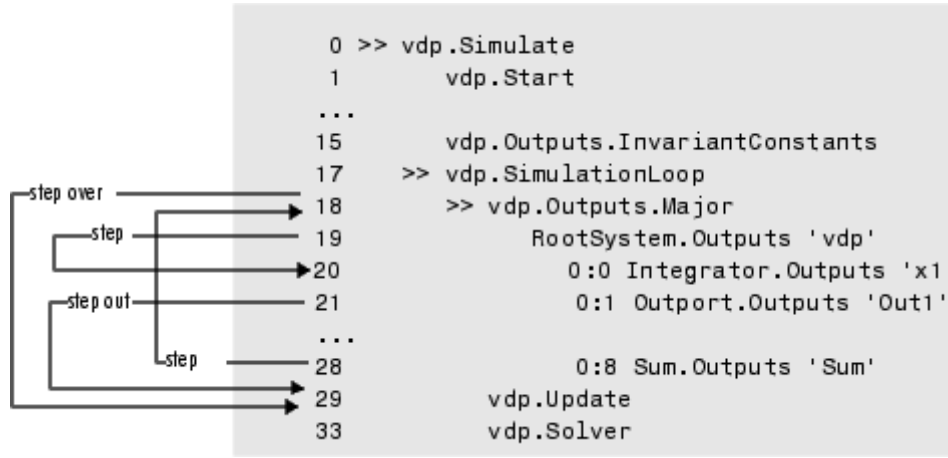
`s` is the short form of the command.

If `step` advances the simulation to the start of a block method, the debugger points at the block on which the method operates.

.

Examples

The following diagram illustrates the effect of various forms of the `step` command for the model `vdp`.



See Also

`next` | `where` | `elist`

Purpose

Model sample times

Syntax

```
stimes
sti
```

Description

`stimes` displays information about the model sample times, including the sample time period, offset, and task ID.

`sti` is the short form of the command.

Examples

Display sample times for the model `vdp` using the Simulink debugger.

1. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2. Enter:

   ```
   stimes
   ```

   The MATLAB Command Window displays:

   ```
   --- Sample times  for 'vdp' [Number of sample times = 1]
     1. [0     , 0     ]  tid=0 (continuous sample time)
   ```

stop

Purpose

Stop simulation

Syntax

```
stop
```

Description

`stop` stops the simulation of the model you are debugging.

Examples

Start and stop a simulation for the model `vdp` using the Simulink debugger.

1. Start a debugger session. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2. Start a simulation of the model. Enter:

   ```
   run
   ```

3. Stop the simulation. Enter:

   ```
   stop
   ```

See Also

`continue` | `run` | `quit`

Purpose

Set solver trace level

Syntax

```
strace level
i
```

Description

`strace level` causes the solver to display diagnostic information in the MATLAB Command Window, depending on the value of `level`. Values are `0` (no information) or `1` (maximum information about time steps, integration steps, zero crossings, and solver resets).

`i` is the short form of the command.

Examples

Display maximum information about a simulation for the model `vdp` using the Simulink debugger.

1. In the MATLAB Command Window, enter:

   ```
   sldebug 'vdp'
   ```

   The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2. Get information about the notation . Enter:

   ```
   help time
   ```

   The MATLAB Command Window displays:

   ```
   Time is displayed as:
    TM = <time while in MajorTimeStep>
    Tm = <time while in MinorTImeStep>
    Ts = <time of successful integration step>
    Tf = <time of failed integration step>
    Tr = <time of solver reset>
    Tj = <time of Jacobian evaluation> (when using implicit solvers)
    Tz = <time at left post of interval bracketing zero crossing eve
   Step size is displayed as:
    H = <step size>
   ```

```
Hs = <failed integration step size>
Hf = failed integration step size>
Hr = <step size at sovler reset>
Hz = <step size at zero crossing event>
Izc= <length of time interval bracketing zero crossing eent(s)>
```

3 Set trace to display all information. Enter:

```
strace 1
```

When diagnostic tracing is on, the debugger displays the sizes of major and minor time steps.

```
[TM = 13.21072088374186 ] Start of Major Time Step
[Tm = 13.21072088374186 ] Start of Minor Time Step
```

The debugger displays integration information. This information includes the time step of the integration method, step size of the integration method, outcome of the integration step, normalized error, and index of the state.

```
[Tm = 13.21072088374186 ] [H  = 0.2751116230148764 ] Begin Integration Step
[Tf = 13.48583250675674 ] [Hf = 0.2751116230148764 ] Fail  [Er = 1.0404e+000]
 [Ix = 1]
[Tm = 13.21072088374186 ] [H  = 0.2183536061326544 ] Retry
[Ts = 13.42907448987452 ] [Hs = 0.2183536061326539 ] Pass  [Er = 2.8856e-001]
 [Ix = 1]
```

For zero crossings, the debugger displays information about the iterative search algorithm when the zero crossing occurred. This information includes the time step of the zero crossing, step size of the zero crossing detection algorithm, length of the time interval bracketing the zero crossing, and a flag denoting the rising or falling direction of the zero crossing.

```
[Tz = 3.615333333333301 ] Detected 1 Zero Crossing Event 0[F]
                          Begin iterative search to bracket zero crossing event
[Tz = 3.621111157580072 ] [Hz = 0.005777824246771424 ] [Iz = 4.2222e-003] 0[F]
[Tz = 3.621116982080098 ] [Hz = 0.005783648746797265 ] [Iz = 4.2164e-003] 0[F]
```

```
[Tz = 3.621116987943544 ] [Hz = 0.005783654610242994 ] [Iz = 4.2163e-003] 0[F]
[Tz = 3.621116987943544 ] [Hz = 0.005783654610242994 ] [Iz = 1.1804e-011] 0[F]
[Tz = 3.621116987949452 ] [Hz = 0.005783654616151157 ] [Iz = 5.8962e-012] 0[F]
[Tz = 3.621116987949452 ] [Hz = 0.005783654616151157 ] [Iz = 5.1514e-014] 0[F]
                          End iterative search to bracket zero crossing event
```

When a solver resets occur, the debugger displays the time at which the solver was reset.

```
[Tr = 6.246905153573676 ] Process Solver Reset
[Tr = 6.246905153573676 ] Reset Zero Crossing Cache
[Tr = 6.246905153573676 ] Reset Derivative Cache
```

See Also

`atrace` | `etrace` | `states` | `trace` | `zclist`

systems

Purpose

List nonvirtual systems of model

Syntax

```
systems
sys
```

Description

`systems` displays the nonvirtual subsystems for a model in the MATLAB Command Window.

`sys` is the short form of the command.

Examples

Display the nonvirtual systems for the model `sldemo_enginewc` using the Simulink debugger.

1 In the MATLAB Command Window, enter:

```
sldebug 'sldemo_enginewc'
```

The MATLAB command prompt >> changes to the Simulink debugger prompt (sldebug @0): >>.

2 Enter:

```
systems
```

The MATLAB Command Window displays the nonvirtual subsystems.

```
  0   'sldemo_enginewc'
  1   'sldemo_enginewc/Compression'
  2   'sldemo_enginewc/Controller/TmpAtomicSubsysAtSwitchInport3'
  3   'sldemo_enginewc/Controller/TmpAtomicSubsysAtSwitchInport1'
  4   'sldemo_enginewc/Controller'
  5   'sldemo_enginewc/Throttle & Manifold/Throttle/TmpAtomicSubsysAt
  6   'sldemo_enginewc/valve timing/positive edge to dual edge conver
```

See Also

`slist`

Purpose

Set or clear time breakpoint

Syntax

```
tbreak
tbreak t
```

Short Form

```
tb
```

Description

The `tbreak` command sets a breakpoint at the specified time step. If a breakpoint already exists at the specified time, `tbreak` clears the breakpoint. If you do not specify a time, `tbreak` toggles a breakpoint at the current time step.

See Also

`break`, `bafter`, `ebreak`, `xbreak`, `nanbreak`, `zcbreak`, `rbreak`

trace

Purpose

Display block's I/O each time block executes

Syntax

```
trace gcb
trace s:b
```

Short Form

```
tr
```

Arguments

`s:b`	The block whose system index is `s` and block index is `b`.
`gcb`	Current block.

Description

The `trace` command registers a block as a trace point. The debugger displays the I/O of each registered block each time the block executes.

See Also

`disp`, `probe`, `untrace`, `slist`, `strace`

Purpose

Remove block from debugger's list of display points

Syntax

```
undisp gcb
undisp s:b
```

Short Form

```
und
```

Arguments

`s:b`	The block whose system index is `s` and block index is `b`.
`gcb`	Current block.

Description

The `undisp` command removes the specified block from the debugger's list of display points.

See Also

`disp`, `slist`

untrace

Purpose

Remove block from debugger's list of trace points

Syntax

```
untrace gcb
untrace s:b
```

Short Form

```
unt
```

Arguments

`s:b`	The block whose system index is `s` and block index is `b`.
`gcb`	Current block.

Description

The `untrace` command removes the specified block from the debugger's list of trace points.

See Also

`trace`, `slist`

Purpose

Display current location of simulation in simulation loop

Syntax

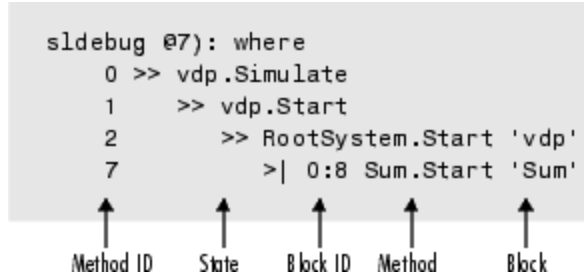
```
where [detail]
```

Short Form

```
w
```

Description

The `where` command displays the current location of the simulation in the simulation loop, for example,



The display consists of a list of simulation nodes with the last entry being the node that is about to be entered or exited. Each entry contains the following information:

- Method ID

  The method ID identifies a specific invocation of a method.

- A symbol specifying its state:
  - `>>` (active)
  - `>|` (about to be entered)
  - `<|` (about to be exited)
- Name of the method invoked (e.g., RootSystem.Start)
- Name of the block or system on which the method is invoked (e.g., Sum)

- System and block ID (`sid:bid`) of the block on which the method is invoked

  For example, `0:8` indicates that the specified method operates on block `8` of system `0`.

`where detail`, where `detail` is any nonnegative integer, includes inactive nodes in the display.

```
0 >> vdp.Simulate
    1     >> vdp.Start
    2        >> RootSystem.Start 'vdp'
    3           0:4 Scope.Start 'Scope'
    4              0:5 Fcn.Start 'Fcn'
    5              0:6 Product.Start
'Product'
    6              0:7 Gain.Start 'Mu'
    7           >| 0:8 Sum.Start 'Sum'
```

See Also

`step`

Purpose

Break when debugger encounters step-size-limiting state

Syntax

```
xbreak
```

Short Form

```
x
```

Description

The `xbreak` command pauses execution of the model when the debugger encounters a state that limits the size of the steps that the solver takes. If `xbreak` mode is already on, `xbreak` turns the mode off.

See Also

`break`, `bafter`, `ebreak`, `zcbreak`, `tbreak`, `nanbreak`, `rbreak`

zcbreak

Purpose

Toggle breaking at nonsampled zero-crossing events

Syntax

```
zcbreak
```

Short Form

```
zcb
```

Description

The `zcbreak` command causes the debugger to break when a nonsampled zero-crossing event occurs. If zero-crossing break mode is already on, `zcbreak` turns the mode off.

See Also

`break`, `bafter`, `xbreak`, `tbreak`, `nanbreak`, `zclist`

Purpose

List blocks containing nonsampled zero crossings

Syntax

```
zclist
```

Short Form

```
zcl
```

Description

The `zclist` command displays a list of blocks in which nonsampled zero crossings can occur. The command displays the list in the MATLAB Command Window.

See Also

`zcbreak`

7

Simulink Classes

eventData	Provide information about block method execution events
getDescription	Extract MDL file description without loading the block diagram into memory
getMetadata	Extract MDL file metadata without loading the block diagram into memory
Simulink.AliasType	Create alias for signal and/or parameter data type
Simulink.Annotation	Specify properties of model annotation
Simulink.BlockCompDworkData	Provide postcompilation information about block's DWork vector
Simulink.BlockCompInputPortData	Provide postcompilation information about block input port
Simulink.BlockCompOutput-PortData	Provide postcompilation information about block output port
Simulink.BlockData	Provide run-time information about block-related data, such as block parameters
Simulink.BlockPortData	Describe block input or output port
Simulink.BlockPreComp-InputPortData	Provide precompilation information about block input port

Simulink.BlockPreComp-OutputPortData	Provide precompilation information about block output port
Simulink.Bus	Specify properties of signal bus
Simulink.Buselement	Describe element of signal bus
Simulink.ConfigSet	Access model configuration set
Simulink.ConfigSetRef	Link model to configuration set stored independently of any model
Simulink.MDLInfo	Extract MDL file information without loading block diagram into memory
Simulink.ModelAdvisor	Run Model Advisor from M-file
Simulink.ModelDataLogs	Container for model's signal data logs
Simulink.ModelWorkspace	Describe model workspace
Simulink.MSFcnRunTimeBlock	Get run-time information about Level-2 M-file S-function block
Simulink.NumericType	Specify data type
Simulink.Parameter	Specify value, value range, data type, and other properties of block parameter
Simulink.ParamRTWInfo	Specify information needed to generate code for parameter
Simulink.RunTimeBlock	Allow Level-2 M-file S-function and other M-file programs to get information about block while simulation is running
Simulink.ScopeDataLogs	Store data logged by Scope signal viewer
Simulink.Signal	Specify attributes of signal
Simulink.SignalRTWInfo	Specify information needed to generate code for signal
Simulink.Structelement	Describe element of data structure

Simulink.StructType	Describe data structure used as value of signal or parameter
Simulink.SubsysDataLogs	Container for subsystem's signal data logs
Simulink.TimeInfo	Provide information about time data in `Simulink.Timeseries` object
Simulink.Timeseries	Store data for any signal except mux or bus signal
Simulink.TsArray	Store data for mux or bus signal
Simulink.Variant	Specifies a model reference variant and its execution environment

eventData

Purpose

Provide information about block method execution events

Description

Simulink software 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 *Simulink User's Guide* 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 alias for 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 7-6). 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 software 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 software 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.

Note Suppose you specify an instance of the `Simulink.AliasType` class as the value of a `Simulink.Parameter` object's **Data type** property. If you enter the parameter object in a subsystem's mask, the subsystem displays the data type's base type instead of its alias name.

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 7-7) to create a data type alias.

To use the Model explorer to create an alias:

1. Select `Base Workspace` (i.e., 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 software displays an error.

2. Select **Simulink.AliasType** from the Model explorer's **Add** menu.

   Simulink software 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. Use the MATLAB `save` command to save the newly created alias in a MAT-file that can be 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.

```
old = MyFloat.BaseType;
MyFloat.BaseType = 'single';
```

See "Properties" on page 7-10 for information on the names, permitted values, and usage of the properties of data type alias objects.

Data Type Aliases in the Generated Code

You can cause data type aliases to appear in the code generated for a model using any of the following methods.

- Specifying the signal data type of a block in the model as a `Simulink.AliasType` via the **Block Parameters** dialog box.
- Creating a `Simulink.Signal` object that uses the `Simulink.AliasType` as its data type. Use this signal object as the name of a signal in the model and specify that the signal name must resolve to an object in the MATLAB workspace. See "Signal Objects" in the Real-Time Workshop User's Guide for more information.
- Creating a `Simulink.Parameter` object that uses the `Simulink.AliasType` as its data type. Use this parameter object as a block parameter in the model. See "Generated Code for Parameter Data Types" in the Real-Time Workshop User's Guide for more information.

Notes

- If you assign a data type in a block's **Block Parameters** dialog box and by using a `Simulink.Signal` object on the signal feeding into the block, the code is always generated using the data type in the dialog box.
- The Real-Time Workshop code generator tries to preserve the names of alias types in the generated code. However, in some cases, an alias type name might revert to its underlying equivalent built-in data type. If you have a Real-Time Workshop Embedded Coder license, you can guarantee that the code generator uses the alias type name in the generated code, by using replacement types (see in the Real-Time Workshop Embedded Coder documentation).

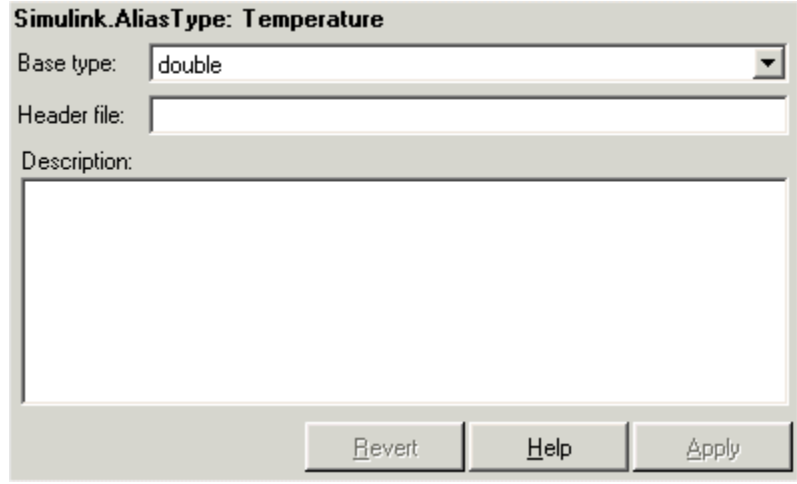
Parent

None

Children

None

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`.

Note `Fixed-point:  unspecified scaling` is a partially specified type whose definition is completed by the block that uses the `Simulink.NumericType`. Forbidding its use in alias types avoids creating aliases that have different base types depending on where they are used.

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**)

Name	Description
`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 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';
```

Children

None.

Property Summary

Property	Description	Values
`Text`	String specifying text of annotation. Same as `Name`.	string
`ClickFcn`	Specifies MATLAB code to be executed when a user single-clicks this annotation. Simulink software stores the code entered in this field with the model. See "Associating Click Functions with Annotations" for more information.	string
`Description`	String that describes this annotation.	string

Property	Description	Values
`FontAngle`	String specifying the angle of the annotation's font. The default value, `'auto'`, specifies use of the model's preferred font angle.	`'normal' | 'italic' | 'oblique' | {'auto'}`
`FontName`	String specifying name of annotation's font. The default value, `'auto'`, specifies use of the model's preferred font.	string
`FontSize`	Integer specifying size of annotation's font in points. The default value, `-1`, specifies use of the model's preferred font size.	real `{'-1'}`
`FontWeight`	String specifying the weight of the annotation's font. The default value, `'auto'`, specifies use of the model's preferred font weight.	`'light' | 'normal' | 'demi' | 'bold' | {'auto'}`
`Handle`	Annotation handle.	real
`HiliteAncestors`	For internal use.	
`Name`	String specifying text of annotation. Same as `Text`.	string
`Selected`	String specifying whether this annotation is currently selected (`'on'`) or not selected (`'off'`).	`{'on'} | 'off'`
`Parent`	String specifying parent name of annotation object.	string
`Path`	Path to the annotation.	string

Property	Description	Values
`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]`.	vector `[left bottom]` not enclosed in quotation marks. The maximum value for a coordinate is 32767.
`Horizontal-Alignment`	String specifying the horizontal alignment of this annotation, e.g., `'center'`.	`{'center'} | 'left'|'right'`
`VerticalAlignment`	String specifying the vertical alignment of this annotation, e.g., `'middle'`.	`{'middle'} | 'top'|'cap'|'baseline'|'bottom'`
`ForegroundColor`	String specifying foreground color of this annotation.	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, delineated with commas. The alpha value is optional and ignored. Block background color can also be `'black'`, `'white'`, `'red'`, `'green'`, `'blue'`, `'cyan'`, `'magenta'`, `'yellow'`, `'gray'`, `'lightBlue'`, `'orange'`, `'darkGreen'`.

Property	Description	Values
`BackgroundColor`	String specifying background color of this annotation.	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, delineated with commas. The alpha value is optional and ignored. Block background color can also be `'black'`, `'white'`, `'red'`, `'green'`, `'blue'`, `'cyan'`, `'magenta'`, `'yellow'`, `'gray'`, `'lightBlue'`, `'orange'`, `'darkGreen'`.
`DropShadow`	String specifying whether to display a drop shadow. Options are `'on'` or `'off'`.	`'on' | {'off'}`
`TeXMode`	String specifying whether to render TeX markup. Options are `'on'` or `'off'`.	`'on' | {'off'}`
`Type`	Annotation type. This is always `'annotation'`	string
`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.	string

Property	Description	Values
`DeleteFcn`	String specifying M-code to be executed before deleting this annotation. See "Annotation Callback Functions" in the online Simulink documentation.	string
`RequirementInfo`	For internal use.	string
`Tag`	User-specified text that is assigned to the annotation's Tag parameter and saved with the annotation.	string
`UseDisplayText-AsClickCallback`	String specifying whether to use the contents of the `Text` property as this annotation's click function. Options are `'on'` or `'off'`. If set to `'on'`, the text of the annotation is interpreted as a valid MATLAB expression and run. If set to `'off'`, clicking on the annotation runs the click function, if there is one. If there is no click function, clicking the annotation has no effect. See "Associating Click Functions with Annotations" in the Simulink documentation for more information.	`'on' | {'off'}`
`UserData`	Any data that you want to associate with this annotation.	vector

Purpose

Provide postcompilation information about block's DWork vector

Description

Simulink software 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 7-141 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
"Usage" on page 7-17	Usage type of this DWork vector.
"UsedAsDiscState"	True if this DWork vector is being used to store the values of a block's discrete states.

Properties

`Usage`

Description

Returns a string indicating how this DWork vector is used. Permissible values are:

- DWork
- DState
- Scratch
- Mode

Data Type

string

Access

`RW` for M-file S-function blocks, `RO` for other blocks.

`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

Provide postcompilation information about block input port

Description

Simulink software 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 7-142 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

Provide postcompilation information about block output port

Description

Simulink software 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 7-143 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"	Specifies whether an output port's memory is reusable.

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
"AliasedThroughDataType" on page 7-22	Fundamental base data type.
"AliasedThroughDataType-ID" on page 7-23	Fundamental base data type ID.
"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

`AliasedThroughDataType`

Description

Data type aliases allow a data type (B) to be recursively aliased to another alias type or `BaseType` (A). If alias type A is aliased to another alias type that is aliased to another alias type and so forth, this property allows the alias type to be iteratively searched (aliased through) until the type is no longer an alias type and that final result is the value of the property returned. For example, assume that you have created the Simulink Alias types A and B as follows:

```
A=Simulink.AliasType('double')

A =
Simulink.AliasType
    Description: ''
     HeaderFile: ''
       BaseType: 'double'
B=Simulink.AliasType('A')

B =
Simulink.AliasType
    Description: ''
     HeaderFile: ''
       BaseType: 'A'
```

If the data type of an item of block data is B, this property returns the base type A instead of B.

Data Type

string

Access

`RO`

`AliasedThroughDataTypeID`

Description

Index of the data type alias returned by the `AliasedThroughDataType` property.

Data Type

integer

Access

`RO`

`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. enter the numeric value for the desired data type, as follows:

Data Type	Value
`'inherited'`	`-1`
`'double'`	`0`
`'single'`	`1`
`'int8'`	`2`
`'uint8'`	`3`
`'int16'`	`4`
`'uint16'`	`5`

Data Type	Value
'int32'	6
'uint32'	7
'boolean' or fixed-point data types	8

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.
'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

Simulink.BlockPortData

Purpose

Describe 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.BlockPreComp-OutputPortData`, `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 "How to Specify the 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

Purpose

Provide precompilation information about block input port

Description

Simulink software 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 7-142 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

Simulink.BlockPreCompOutputPortData

Purpose

Provide precompilation information about block output port

Description

Simulink software 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 7-143 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"	Specifies whether an output port's memory is reusable.

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

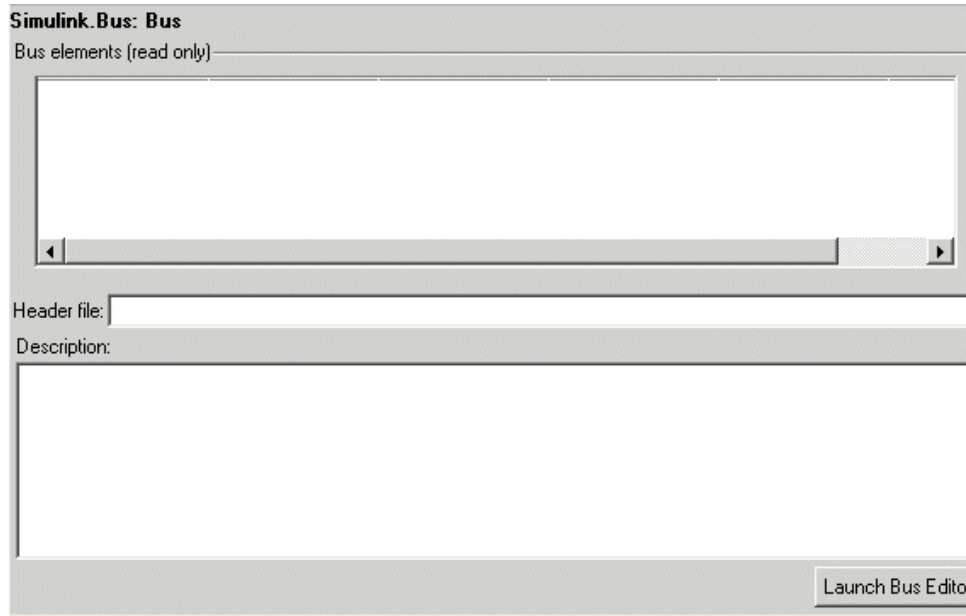
Specify properties of signal bus

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 software to validate the properties 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 software checks whether the buses connected to the blocks have the properties specified by the bus objects. If not, Simulink software halts and displays an error message.

You can use the Model explorer's **Add > Simulink Bus** command (see "Using the Model Explorer to Create Data Objects"), the Simulink Bus editor (see Using the Bus editor), or MATLAB commands (see "Working with Data Objects") to create bus objects in the base MATLAB workspace. If you attempt to create an alias in a model workspace, Simulink software displays an error. You must use the Bus editor or the MATLAB command line to set the properties of a bus object. Simulink software also provides a set of utility functions for creating and saving bus objects.

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 "Using Bus Objects") 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 software (see "Code Generation with User-Defined Data Types" *Real-Time Workshop Embedded Coder User's Guide*). Simulink software ignores this field.

Description

Description of this structure. This field is intended for you to use to document this bus. Simulink software does not use this field.

Properties

Name	Access	Description
`Description`	`RW`	String that describes this bus. This property is intended for user use. Simulink software 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 software. Simulink does not use it. (Header file)

See Also

"Using Composite Signals", Bus Assignment, Bus Creator, Bus Selector, Bus to Vector, `Simulink.Bus.cellToObject`, `Simulink.Bus.createObject`, `Simulink.Buselement`, `Simulink.Bus.objectToCell`, `Simulink.Bus.save`

Simulink.Buselement

Purpose

Describe element of signal bus

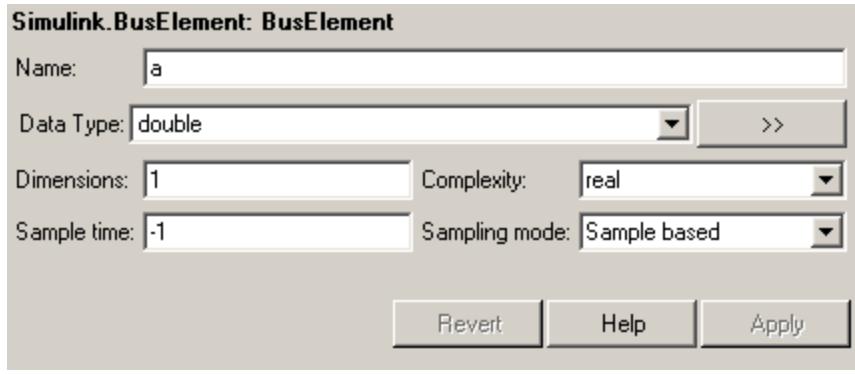
Description

Objects of this class define elements of buses defined by objects of the `Simulink.Bus` class.

Property Summary

Name	Description
"Complexity"	Numeric type of this bus element.
"DataType"	Data type of this bus element.
"Dimensions"	Dimensions of this bus element.
"Name"	Name of this bus element.
"SampleTime"	Sample time of this bus element.
"SamplingMode"	Sampling mode of this bus element.

Property Dialog Box



Properties

`Complexity`

Numeric type (`'real'` or `'complex'`) of this element. Must be `'real'` if this bus element is itself a bus.

`Data Type:` string

`Access:` RW

`DataType`

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`.

> **Note** `Fixed-point: unspecified scaling` is a partially specified type whose definition is completed by the block that uses the `Simulink.NumericType`. Forbidding its use for bus elements avoids creating bus elements that have different data types depending on where they are used.

- `Simulink.Bus` object. This allows you to create bus objects that specify hierarchical buses, i.e., buses that contain other buses.

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Data type** parameter. (See "Using the Data Type Assistant".)

`Data Type:` string

`Access:` RW

`Dimensions`

A vector specifying the dimensions of this element. Must be `1` if this element is itself a bus.

`Data Type:` array.

Access: RW

Name

Name of this element.

Data Type: string

Access: RW

SampleTime

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 "How to Specify the Sample Time" for more information.

Data Type: double

Access: RW

SamplingMode

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 models.

Data Type: string

Access: RW

See Also

"Using Composite Signals", Bus Assignment, Bus Creator, Bus Selector, Bus to Vector, `Simulink.Bus`, `Simulink.Bus.cellToObject`, `Simulink.Bus.createObject`, `Simulink.Bus.objectToCell`, `Simulink.Bus.save`

Purpose

Access model configuration set

Description

Instances of this handle class allow you to write programs to create, modify, and attach configuration sets to models. See "Setting Up Configuration Sets" and "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 for simulation with this configuration.

Note You can use the **Model Configuration** dialog box to set the `Name` and `Description` properties of a configuration set. See "Model Configuration Dialog Box" 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.
"isActive"	Determine whether a configuration set is the active set of the model that owns it.

Name	Description
“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. This field can remain blank.

Data Type

`string`

Access

`RW`

`Name`

Description

Configuration set’s name. This name represents the configuration set in the Model explorer.

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 *Simulink User's Guide* 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 pathname 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. Specifying *`paramName`* as `'ObjectParameters'` returns the names of the valid parameters in the configuration set.

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.

Simulink.ConfigSetRef

Purpose

Link model to configuration set stored independently of any model

Description

Instances of this handle class allow a model to reference configuration sets that exist outside any model. See “Setting Up Configuration Sets”, “Configuration Set API”, and “Referencing Configuration Sets” for more information.

Property Summary

Name	Description
“Description”	Description of the configuration reference.
“Name”	Name of the configuration reference.
“WSVarName”	Name of the workspace variable that contains the referenced configuration set.

Note You can use the **Configuration Reference** dialog box to set the `Name`, `Description`, and `WSVarName` properties of a configuration reference. See “Creating and Attaching a Configuration Reference” for details.

Method Summary

Name	Description
“copy”	Create a copy of a configuration reference.
“getFullName”	Get the full pathname of a configuration reference.
“getModel”	Get the handle of the model that owns a configuration reference.
“get_param”	Get the value of a configuration set parameter indirectly through a configuration reference.
“getRefConfigSet”	Get the configuration set specified by a configuration reference.

Name	Description
“isActive”	Determine whether a configuration reference is the active configuration object of the model.
“refresh”	Update configuration reference after any change to properties or configuration set availability.

Properties

`Description`

Description

Description of the configuration reference. You can use this property to provide additional information about a configuration reference, such as its purpose. This field can remain blank.

Data Type

`string`

Access

`RW`

`Name`

Description

Name of the configuration reference. This name represents the configuration reference in the GUI.

Data Type

`string`

Access

`RW`

`WSVarName`

Description

Name of the workspace variable that contains the referenced configuration set.

Data Type
`string`

Access
`RW`

Methods

`copy`

Purpose
Create a copy of this configuration reference.

Syntax
`copy`

Description
This method creates a copy of this configuration set.

Note You must use this method to create copies of configuration references. This is because `Simulink.ConfigSetRef` is a handle class. See "Handle Versus Value Classes" for more information.

`getFullName`

Purpose
Get the full pathname of a configuration reference.

Syntax
`getFullName`

Description
This method returns a string specifying the full pathname of a configuration reference, e.g., `'vdp/Configuration'`.

`getModel`

Purpose
Get the model that owns this configuration reference.

Syntax

```
getModel
```

Description

Returns a handle to the model that owns this configuration reference.

example

The following command opens the block diagram of the model that owns the configuration set referenced by the MATLAB workspace variable `hCr`.

```
open_system(hCr.getModel);
```

`get_param`

Purpose

Get the value of a configuration set parameter indirectly through a configuration reference.

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 from the configuration set to which the configuration reference points. To obtain this value, the method uses the value of `WSVarName` to retrieve the configuration set, then retrieves the value of *`paramName`* from that configuration set. Specifying *`paramName`* as `'ObjectParameters'` returns the names of all valid parameters in the configuration set. If a valid configuration set is not attached to the configuration reference, the method returns unreliable values.

The inverse method, `set_param`, is not defined for configuration references. To obtain a parameter value through a configuration reference, you must first use the `getRefConfigSet` method to retrieve

the configuration set from the reference, then use `set_param` directly on the configuration set itself.

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.

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');
```

`getRefConfigSet`

Purpose

Get the configuration set specified by a configuration reference

Syntax

`getRefConfigSet`

Description

Returns a handle to the configuration set specified by the `WSVarName` property of a configuration reference.

`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.

`refresh`

Purpose

Update configuration reference after any change to properties or configuration set availability

Syntax

```
refresh
```

Description

Updates a configuration reference after using the API to change any property of the reference, or after providing a configuration set that did not exist at the time the set was originally specified in `WSVarName`. If you omit executing `refresh` after any such change, the configuration reference handle will be stale, and using it will give incorrect results.

Simulink.MDLInfo

Purpose

Extract MDL file information without loading block diagram into memory

Description

The class Simulink.MDLInfo extracts information from an MDL file without loading the block diagram into memory.

You can create an `MdlInfo` object containing all the model information properties, or you can use the static methods for convenient access to individual properties without creating the class first. For example, to get the description only:

```
description = Simulink.MDLInfo.getDescription('mymodel')
```

To get the metadata only:

```
metadata = Simulink.MDLInfo.getMetadata('mymodel')
```

All model information properties are read only.

Construction

`info = MDLInfo(*'mymodel'*)` creates an instance of the `MdlInfo` class `info` and populates the properties with the information from the MDL file *`'mymodel'`*.

`mymodel` can be:

- A block diagram name (for example, `vdp`)
- The file name for a file on the MATLAB path (for example, `vdp.mdl`)
- A file name relative to the current folder (for example, `mydir/mymodel.mdl`)
- A fully qualified file name (for example, `C:\mydir\mymodel.mdl`)

Properties

`BlockDiagramName`

Name of block diagram.

`Description`

Description of model.

`FileName`

Name of MDL file.

`IsLibrary`

Whether the block diagram is a library.

`Metadata`

Names and attributes of arbitrary data associated with the model.

Structure. The structure fields can be strings, numeric matrices of type "double", or more structures. Use the method `getMetadata` to extract this metadata structure without loading the model.

`ModelVersion`

Model version number.

`SimulinkVersion`

Version number of Simulink software that was used to save the MDL file.

Methods

getDescription	Extract MDL file description without loading the block diagram into memory
getMetadata	Extract MDL file metadata without loading the block diagram into memory

Copy Semantics

Handle. To learn how this affects your use of the class, see Copying Objects in the MATLAB Programming Fundamentals documentation.

Examples

Construct and view a model information object:

```
info = Simulink.MDLInfo('mymodel')
% Get the Version when the model was saved
simulink_version = info.SimulinkVersion;
```

```
% Get model metadata
metadata = info.metadata
```

To add metadata to a model, create a metadata structure containing the information you require and use `set_param` to attach it to the model. For example:

```
metadata.TestStatus = 'untested';
metadata.ExpectedCompletionDate = '01/01/2011';
load_system(mymodelname);
set_param(mymodelname,'Metadata',...
metadata)  % must be a struct
save_system(mymodelname);
close_system(mymodelname);
```

See Also

`Simulink.MDLInfo.getDescription`; `Simulink.MDLInfo.getMetadata`

Purpose

Extract MDL file description without loading the block diagram into memory

Syntax

```
description = Simulink.MDLInfo.getDescription('mymodel')
description = info.getDescription
```

Description

`description = Simulink.MDLInfo.getDescription('mymodel')` returns the `description` associated with the file *`mymodel`*, without loading the model.

`mymodel` can be:

- A block diagram name (for example, `vdp`)
- The file name for a file on the MATLAB path (for example, `vdp.mdl`)
- A file name relative to the current folder (for example, `mydir/mymodel.mdl`)
- A fully qualified file name (for example, `C:\mydir\mymodel.mdl`)

`description = info.getDescription` returns the `description` property of the `Simulink.MDLInfo` object `info`.

Examples

Get the description without loading the model or creating a `Simulink.MDLInfo` object:

```
description = Simulink.MDLInfo.getDescription('mymodel')
```

Create a `Simulink.MDLInfo` object containing all the model information properties, and get the description property:

```
info = Simulink.MDLInfo('mymodel')
description = info.getDescription
```

See Also

`Simulink.MDLInfo`; `Simulink.MDLInfo.getMetadata`

Simulink.MDLInfo.getMetadata

Purpose

Extract MDL file metadata without loading the block diagram into memory

Syntax

```
metadata = Simulink.MDLInfo.getMetadata('mymodel')
metadata = info.getMetadata
```

Description

`metadata = Simulink.MDLInfo.getMetadata('mymodel')` extracts the structure `metadata` associated with the file *`mymodel`*, without loading the model.

`mymodel` can be:

- A block diagram name (for example, `vdp`)
- The file name for a file on the MATLAB path (for example, `vdp.mdl`)
- A file name relative to the current folder (for example, `mydir/mymodel.mdl`)
- A fully qualified file name (for example, `C:\mydir\mymodel.mdl`)

`metadata = info.getMetadata` returns the `metadata` property of the `Simulink.MDLInfo` object `info`.

`metadata` is a structure containing the names and attributes of arbitrary data associated with the model. The structure fields can be strings, numeric matrices of type "double", or more structures.

To add metadata to a model, create a metadata structure containing the information you require and use `set_param` to attach it to the model. If it is important to extract the information without loading the model, use metadata instead of adding custom user data with `add_param`.

Examples

Create a metadata structure and use `set_param` to attach it to the model:

```
metadata.TestStatus = 'untested';
metadata.ExpectedCompletionDate = '01/01/2011';
load_system('mymodel');
set_param('mymodel','Metadata',metadata) % must be a struct
```

```
save_system('mymodel');
close_system('mymodel');
```

Get the metadata without loading the model or creating a `Simulink.MDLInfo` object:

```
metadata = Simulink.MDLInfo.getMetadata('mymodel')
```

Create a `Simulink.MDLInfo` object containing all the model information properties, and get the metadata property:

```
info = Simulink.MDLInfo('mymodel')
metadata = info.getMetadata
```

See Also

`Simulink.MDLInfo`; `Simulink.MDLInfo.getDescription`

Simulink.ModelAdvisor

Purpose

Run Model Advisor from M-file

Description

Use instances of this class in M-file programs to run the Model Advisor, for example, to perform a standard set of checks. MATLAB software creates an instance of this object for each model that you open in the current MATLAB session. To get a handle to a model's Model Advisor object, execute the following command

```
ma = Simulink.ModelAdvisor.getModelAdvisor(model);
```

where `model` is the name of the model or subsystem that you want to check. Your program can then use the Model Advisor object's methods to initialize and run the Model Advisor's checks.

About IDs

Many `Simulink.ModelAdvisor` object methods require or return IDs. An *ID* is a string that identifies a Model Advisor check, task, or group. ID must remain constant. A `Simulink.ModelAdvisor` object includes methods that enable you to retrieve the ID or IDs for all checks, tasks, and groups, checks belonging to groups and tasks, the active check, and selected checks, tasks and groups. See the `Simulink.ModelAdvisor` "Method Summary" for more information.

You can also determine check IDs as follows:

1. In the Model Advisor, select **View > Source Tab**.
2. Navigate to the folder that contains the check.
3. In the right pane, click **Source**. The Model Advisor displays the **Title**, **TitleID**, and **Source** information for each check in the folder.

Syntax

```
ma = Simulink.ModelAdvisor
```

Arguments

ma

A variable representing the `Simulink.ModelAdvisor` object you create.

Method Summary

Name	Description
“closeReport”	Close Model Advisor report.
“deselectCheck”	Deselect checks.
“deselectCheckAll”	Deselect all checks.
“deselectCheckForGroup”	Deselect a group of checks.
“deselectCheckForTask”	Deselect checks that belong to a specified task or set of tasks.
“deselectTask”	Deselect tasks.
“deselectTaskAll”	Deselect all tasks.
“displayReport”	Display Model Advisor report.
“exportReport”	Copy report to a specified location.
“getBaselineMode”	Get baseline mode setting for the Model Advisor.
“getCheckAll”	Get the IDs of the checks performed by the Model Advisor.
“getCheckForGroup”	Get checks belonging to a check group.
“getCheckForTask”	Get checks belonging to a task.

Name	Description
"getCheckResult"	Get check results.
"getCheckResultData"	Get check result data.
"getCheckResultStatus"	Get pass/fail status of a check or set of checks.
"getGroupAll"	Get the IDs of the groups of tasks performed by the Model Advisor.
"getInputParameters" on page 7-73	Get input parameters of a check.
"getListViewParameters" on page 7-74	Get list view parameters of a check.
"getModelAdvisor"	Get the Model Advisor for a model or subsystem.
"getSelectedCheck"	Get selected checks.
"getSelectedSystem" on page 7-76	Get path of system currently targeted by the Model Advisor.
"getSelectedTask"	Get selected tasks.
"getTaskAll"	Get the IDs of the tasks performed by the Model Advisor.
"Simulink.ModelAdvisor.openConfigUI" on page 7-77	Start the Model Advisor Configuration editor.
"Simulink.ModelAdvisor.reportexists"	Determine whether a report exists for a system or subsystem.
"runCheck"	Run selected checks.
"runTask"	Run checks for selected tasks.

Name	Description
"selectCheck"	Select checks.
"selectCheckAll"	Select all checks.
"selectCheckForGroup"	Select a group of checks.
"selectCheckForTask"	Select checks that belong to a specified task.
"selectTask"	Select tasks.
"selectTaskAll"	Select all tasks.
"setActionenable" on page 7-82	Set enable/disable status for a check action.
"setBaselineMode"	Set baseline mode for the Model Advisor.
"setCheckerrorSeverity" on page 7-84	Set severity of a check failure.
"setCheckResult"	Set result for the currently running check.
"setCheckResultData"	Set result data for the currently running check.
"setCheckResultStatus"	Set pass/fail status for the currently running check.
"setListViewParameters" on page 7-87	Set list view parameters for a check.
"verifyCheckRan"	Verify that checks have run.
"verifyCheckResult"	Generate a baseline set of check results or compare the current set of results to the baseline results.

Name	Description
“verifyCheckResultStatus”	Verify that a model has passed or failed a set of checks.
“verifyHTML”	Generate a baseline report or compare the current report to a baseline report.

Methods

closeReport

Purpose

Close Model Advisor report.

Syntax

```
closeReport
```

Description

Closes the report associated with this Model Advisor object, which closes the Model Advisor window.

See Also

“displayReport”

deselectCheck

Purpose

Deselect a check.

Syntax

```
success = deselectCheck(ID)
```

Arguments

`ID`

String or cell array that specifies the IDs of the checks to be deselected

`success`

True (1) if the check is deselected.

Description

This method deselects the checks specified by `ID`.

Note This method cannot deselect disabled checks.

See Also

"getCheckAll", "deselectCheckForGroup", "selectCheck"

deselectCheckAll

Purpose

Deselect all checks.

Syntax

```
success = deselectCheckAll
```

Arguments

`success`

True (1) if all checks are deselected.

Description

Deselects all checks that are not disabled.

See Also

"selectCheckAll"

deselectCheckForGroup

Purpose

Deselect a group of checks.

Syntax

```
success = deselectCheckForGroup(groupName)
```

Arguments

`groupName`

String or cell array that specifies the names of the groups to be deselected

`success`

True (1) if the method succeeds in deselecting the specified group.

Description

Deselects a specified group of checks.

See Also

"selectCheckForGroup"

deselectCheckForTask

Purpose

Deselect checks that belong to a specified task or set of tasks.

Syntax

```
success = deselectCheckForTask(ID)
```

Arguments

`ID`

String or cell array of strings that specify the IDs of tasks whose checks are to be deselected.

`success`

True (1) if the specified tasks are deselected.

Description

Deselects checks belonging to the tasks specified by the `ID` argument.

See Also

"getTaskAll", "selectCheckForTask"

deselectTask

Purpose

Deselect a task.

Syntax

```
success = deselectTask(ID)
```

Arguments

`ID`

String or cell array that specifies the ID of tasks to be deselected

`success`

True (1) if the method succeeded in deselecting the specified tasks

Description

Deselects the tasks specified by `ID`.

See Also

"selectTask", "getTaskAll"

deselectTaskAll

Purpose

Deselect all tasks.

Syntax

```
success = deselectTaskAll
```

Arguments

`success`

True (1) if this method succeeds in deselecting all tasks

Description

Deselects all tasks.

See Also

"selectTaskAll"

displayReport

Purpose

Display report in Model Advisor.

Syntax

```
displayReport
```

Description

Displays the report associated with this Model Advisor object in the Model Advisor window. The report includes the most recent results of running checks on the system associated with this Model Advisor object and the current selection status of checks, groups, and tasks for the system.

See Also

"Simulink.ModelAdvisor.reportexists"

exportReport

Purpose

Create a copy of a report generated by Model Advisor.

Syntax

```
[success message] = exportReport(destination)
```

Arguments

`destination`
: Path name of copy to be made of the report file.

`success`
: True (1) if this method succeeded in creating a copy of the report at the specified location.

`message`
: empty if the copy was successful; otherwise, the reason the copy did not succeed.

Description

This method creates a copy of the last report generated by the Model Advisor and stores the copy at the specified location.

See Also

"Simulink.ModelAdvisor.reportexists"

getBaselineMode

Purpose

Determine whether the Model Advisor is in baseline data generation mode.

Syntax

```
mode = getBaselineMode
```

Arguments

`mode`
: Boolean value indicating baseline mode

Description

The `mode` output variable returns true if the Model Advisor is in baseline data mode. Baseline mode causes the Model Advisor's verification methods, e.g., "verifyHTML", to generate baseline data.

See Also

"setBaselineMode", "verifyHTML", "verifyCheckResult", "verifyCheckResultStatus"

getCheckAll

Purpose

Get the IDs of all checks.

Syntax

```
IDs = getCheckAll
```

Arguments

`IDs`

Cell array of strings specifying the IDs of all checks performed by the Model Advisor

Description

Returns a cell array of strings specifying the IDs of all checks performed by the Model Advisor.

See Also

"getTaskAll", "getGroupAll"

getCheckForGroup

Purpose

Get checks that belong to a check group.

Syntax

```
IDs = getCheckForTask(groupName)
```

Arguments

`groupName`

String specifying the name of a group

`IDs`

Cell array of IDs

Description

Returns a cell array of IDs of the tasks belonging to the group specified by `groupName`.

See Also

"getCheckForTask"

getCheckForTask

Purpose

Get the checks that belong to a task.

Syntax

```
checkIDs = getCheckForTask(taskID)
```

Arguments

`taskID`
: ID of a task

`checkIDs`
: Cell array of IDs of checks belonging to the specified task

Description

Returns a cell array of IDs of the checks belonging to the task specified by `taskID`.

See Also

"getCheckForGroup"

getCheckResult

Purpose

Get the results of running a check or set of checks.

Syntax

```
result = getCheckResult(ID)
```

Arguments

`ID`
: ID of a check or cell array of check IDs

`result`
: A check result or cell array of check results

Description

Gets results for the specified checks. The format of the results depends on the checks that generated the data.

Note This method is intended for accessing results generated by custom checks created with the Model Advisor's customization API, an optional feature available with Simulink® Verification and Validation™ software (see the online Simulink Verification and Validation documentation for more information).

See Also

"getCheckResultData", "getCheckResultStatus"

getCheckResultData

Purpose

Get the data resulting from running a check or set of checks.

Syntax

```
result = getCheckResultData(ID)
```

Arguments

`ID`

Check ID or cell array of check IDs

`result`

Data from a check result or cell array of data from check results

Description

Gets the check result data for the specified checks. The format of the data depends on the checks that generated the data.

Note This method is intended for accessing check result data generated by custom checks created with the Model Advisor's customization API, an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

See Also

"getCheckResult", "getCheckResultStatus"

getCheckResultStatus

Purpose

Get the pass/fail status of a check or set of checks.

Syntax

```
result = getCheckResultStatus(ID)
```

Arguments

`ID`

Check ID or cell array of check IDs

`result`

Boolean or a cell array of Boolean values indication the pass/fail status of a check or set of checks

Description

Invoke this method after running a set of checks to determine whether the checks passed or failed.

See Also

"getCheckResult", "getCheckResultData"

getGroupAll

Purpose

Get all groups of checks performed by the Model Advisor.

Syntax

```
IDs = getGroupAll
```

Arguments

`IDs`

Cell array of IDs of all groups of checks performed by the Model Advisor.

Description

Returns a cell array of IDs of all groups of checks performed by the Model Advisor.

See Also

"getCheckAll", "getTaskAll"

getInputParameters

Purpose

Get input parameters of a check.

Syntax

```
params = obj.getInputParameters(check_ID)
```

Arguments

`params`

A cell array of `ModelAdvisor.InputParameter` objects.

`obj`

A variable representing the `Simulink.ModelAdvisor` object.

`check_ID`

A string that uniquely identifies the check.

You can omit the *`check_ID`* if you use the method inside a check callback function.

Description

Returns the input parameters associated with a check.

Note This method is intended for accessing custom checks created with the Model Advisor's customization API, an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

See Also

`ModelAdvisor.InputParameter`

getListViewParameters

Purpose

Get list view parameters of a check.

Syntax

```
params = obj.getListViewParameters(check_ID)
```

Arguments

params
: A cell array of `ModelAdvisor.ListViewParameter` objects.

obj
: A variable representing the `Simulink.ModelAdvisor` object.

check_ID
: A string that uniquely identifies the check.

  You can omit the *check_ID* if you use the method inside a check callback function.

Description

Returns the list view parameters associated with a check.

Note This method is intended for accessing custom checks created with the Model Advisor's customization API, an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

See Also

"setListViewParameters" on page 7-87,
`ModelAdvisor.ListViewParameter`

getModelAdvisor

Purpose

Get a Model Advisor object for a system or subsystem.

Syntax

```
obj = Simulink.ModelAdvisor.getModelAdvisor(system)
obj = Simulink.ModelAdvisor.getModelAdvisor(system, 'new')
```

Arguments

`system`
: Name of model for which the Model Advisor is to be gotten

`'new'`
: Required when changing Model Advisor working scope from one system to another without closing the previous session. Alternatively, you can close the previous session before invoking `getModelAdvisor`, in which case `'new'` can be omitted.

`obj`
: Model Advisor object

Description

This static method (see "Static Methods") creates and returns an instance of `Simulink.ModelAdvisor` class for the model or subsystem specified by `system`.

getSelectedCheck

Purpose

Get the currently selected checks.

Syntax

```
IDs = getSelectedCheck
```

Arguments

`IDs`

Cell array of IDs of currently selected checks

Description

Returns the checks currently selected in the Model Advisor.

See Also

"getSelectedTask"

getSelectedSystem

Purpose

Get system currently targeted by the Model Advisor.

Syntax

```
path = getSelectedSystem
```

Arguments

`path`

Path of the system selected system

Description

Gets the path of the system currently targeted by the Model Advisor, i.e., the system or subsystem most recently selected for checking either interactively by the user or programmatically via `Simulink.ModelAdisor.getModelAdvisor`.

See Also

"getModelAdvisor"

getSelectedTask

Purpose

Get selected tasks.

Syntax

```
IDs = getSelectedTask
```

Arguments

`IDs`

Cell array of IDs of currently selected tasks.

Description

Returns the tasks currently selected in the Model Advisor.

See Also

"getSelectedCheck"

getTaskAll

Purpose

Get the tasks performed by the Model Advisor.

Syntax

```
IDs = getTaskAll
```

Arguments

`IDs`

Cell array of IDs of tasks performed by the Model Advisor.

Description

Returns a cell array of IDs of tasks performed by the Model Advisor.

See Also

"getCheckAll", "getGroupAll"

Simulink.ModelAdvisor.openConfigUI

Purpose

Starts the Model Advisor Configuration editor.

Syntax

```
Simulink.ModelAdvisor.openConfigUI
```

Description

This static method starts the Model Advisor Configuration editor. Use the Model Advisor Configuration editor to create customized configurations for the Model Advisor.

Note The Model Advisor Configuration editor is an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

- Before starting the Model Advisor Configuration editor, ensure that the current directory is writable. If the directory is not writable, you see an error message when you start the Model Advisor Configuration editor.
- The Model Advisor Configuration editor uses the Simulink project (`slprj`) directory (for more information, see "Model Reference Simulation Targets") in the current directory to store reports and other information. If this directory does not exist in the current directory, the Model Advisor Configuration editor creates it.

Simulink.ModelAdvisor.reportexists

Purpose

Determine whether a report exists for a model or subsystem.

Syntax

```
exists = reportexists('system')
```

Arguments

`'system'`
: String specifying path name of a system or subsystem

`exists`
: True (1) if a report exists for `system`

Description

This method returns true (1) if a report file exists for the model (system) or subsystem specified by `system` in the `slprj/modeladvisor` subdirectory of the MATLAB working directory.

See Also

"exportReport"

runCheck

Purpose

Run the currently selected checks.

Syntax

```
success = runCheck
```

Arguments

`success`

True (1) if the checks were run.

Description

Runs the checks currently selected in the Model Advisor. Invoking this method is equivalent to selecting the **Run Advisor** button on the Model Advisor window.

See Also

"selectCheck"

runTask

Purpose

Run the currently selected tasks.

Syntax

```
success = runTask
```

Arguments

`success`

True (1) if the tasks were run.

Description

Runs the tasks currently selected in the Model Advisor. Invoking this method is equivalent to selecting the **Run Selected Checks** button on the Model Advisor window.

See Also

"selectTask"

selectCheck

Purpose

Select a check.

Syntax

```
success = selectCheck(ID)
```

Arguments

`ID`
: ID or cell array of IDs of checks to be selected

`success`
: True (1) if this method succeeded in selecting the specified checks

Description

This method cannot select a check that is disabled.

See Also

“selectCheckAll”, “selectCheckForGroup”, “deselectCheck”

selectCheckAll

Purpose

Select all checks.

Syntax

```
success = selectCheckAll
```

Arguments

`success`
: True (1) if this method succeeded in selecting all checks.

Description

Selects all checks that are not disabled.

See Also

“selectCheck”, “selectCheckForGroup”, “deselectCheck”

selectCheckForGroup

Purpose

Select a group of checks.

Syntax

```
success = selectCheckForGroup(ID)
```

Arguments

`ID`
: ID or cell array of group IDs

`success`
: True (1) if this method succeeded in selecting the specified groups

Description

Selects the groups specified by `ID`.

See Also

"deselectCheckForGroup"

selectCheckForTask

Purpose

Select checks that belong to a specified task or set of tasks.

Syntax

```
success = selectCheckForTask(ID)
```

Arguments

`ID`
: ID or cell array of IDs of tasks whose checks are to be selected

`success`
: True (1) if this method succeeded in selecting the checks for the specified tasks

Description

Selects checks belonging to the tasks specified by the `ID` argument.

See Also

"deselectCheckForTask"

selectTask

Purpose

Select a task.

Syntax

```
success = selectTask(ID)
```

Arguments

`ID`

ID or cell array of IDs of the task to be selected

`success`

True (1) if this method succeeds in selecting the specified tasks

Description

Selects a task.

See Also

"deselectTask"

selectTaskAll

Purpose

Select all tasks.

Syntax

```
success = selectTaskAll
```

Arguments

`success`

True (1) if this method succeeds in selecting all tasks

Description

Selects all tasks.

See Also

"deselectTaskAll"

setActionenable

Purpose

Set enable/disable status for check action

Syntax

`obj``.setActionenable(`*`value`*`)`

Arguments

obj

A variable representing the `Simulink.ModelAdvisor` object.

value

Boolean value indicating whether the Action box is enabled or disabled.

- `true` — enable the Action box.
- `false` — Disable the Action box.

Description

The `setActionenable` method specifies the enables or disables the Action box. Only a check callback function can invoke this method.

Note This method is intended for accessing custom checks created with the Model Advisor's customization API, an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

See Also

`ModelAdvisor.Action`

setBaselineMode

Purpose

Set the baseline data generation mode for the Model Advisor.

Syntax

```
setBaselineMode(mode)
```

Arguments

`mode`

Boolean value indicating setting of Model Advisor's baseline mode, either on (`true`) or off (`false`)

Description

Sets the Model Advisor's baseline mode to `mode`. Baseline mode causes the Model Advisor's verify methods to generate baseline comparison data for verifying the results of a Model Advisor run.

See Also

"getBaselineMode", "verifyCheckResult", "verifyHTML"

setCheckerrorSeverity

Purpose

Set severity of a check failure.

Syntax

```
obj.setCheckerrorSeverity(value)
```

Arguments

obj

A variable representing the `Simulink.ModelAdvisor` object.

value

Integer indicating severity of failure.

- `0` — Check Result = Warning
- `1` — Check Result = Failed

Description

Sets result status for a currently running check that fails to *`value`*. Only a check callback function can invoke this method.

Note This method is intended for accessing custom checks created with the Model Advisor's customization API, an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

See Also

"setCheckResultStatus"

setCheckResult

Purpose

Set the result for the currently running check.

Syntax

```
success = setCheckResult(result)
```

Arguments

`result`

String or cell array that specifies the result of the currently running task

`success`

True (1) if this method succeeds in setting the check result

Description

Sets the check result for the currently running check. Only the check's callback function can invoke this method.

Note This method is intended for use with custom checks created with the Model Advisor's customization API, an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

See Also

"getCheckResult", "setCheckResultData", "setCheckResultStatus"

setCheckResultData

Purpose

Set the result data for the currently running check.

Syntax

```
success = setCheckResultData(data)
```

Arguments

`data`

Result data to be set

`success`
: True (1) if this method succeeds in setting the result data for the current check

Description

Sets the check result data for the currently running check. Only the check's callback function can invoke this method.

Note This method is intended for use with custom checks created with the Model Advisor's customization API, an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

See Also

"getCheckResultData", "setCheckResult", "setCheckResultStatus"

setCheckResultStatus

Purpose

Set the pass/fail status for the currently running check.

Syntax

```
success = setCheckResultStatus(status)
```

Arguments

`status`
: Boolean value that indicates the status of the check that just ran, either pass (`true`) or fail (`false`)

`success`
: True (1) if the status was set.

Description

Sets the pass/fail status for the currently running check to `status`. Only the check's callback function can invoke this method.

Note This method is intended for use with custom checks created with the Model Advisor's customization API, an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

See Also

"getCheckResultStatus", "setCheckResult", "setCheckResultData", "setCheckerrorSeverity" on page 7-84

setListViewParameters

Purpose

Specify list view parameters for a check.

Syntax

```
obj.setListViewParameters(check_ID, params)
```

Arguments

obj

A variable representing the `Simulink.ModelAdvisor` object.

check_ID

A string that uniquely identifies the check.

You can omit the *check_ID* if you use the method inside a check callback function.

params

A cell array of `ModelAdvisor.ListViewParameter` objects.

Description

Set the list view parameters for the check.

Note This method is intended for accessing custom checks created with the Model Advisor's customization API, an optional feature available with Simulink Verification and Validation software (see the online Simulink Verification and Validation documentation for more information).

See Also

"getListViewParameters" on page 7-74, `ModelAdvisor.ListViewParameter`

verifyCheckRan

Purpose

Verify that the Model Advisor has run a set of checks.

Syntax

```
[success, missingChecks, additionalChecks] =
verifyCheckRan(IDs)
```

Arguments

`IDs`
: Cell array of IDs of checks to verify

`success`
: Boolean value specifying whether the checks ran

`missingChecks`
: Cell array of IDs for specified checks that ran

`additionalChecks`
: Cell array of IDs for unspecified checks that ran

Description

The output variable success returns `true` if all the checks specified by `IDs` have run. If not, `success` returns `false`, `missingChecks` lists specified checks that did not run. The `additionalChecks` argument lists unspecified checks that ran.

See Also

"verifyCheckResultStatus"

verifyCheckResult

Purpose

Generate a baseline Model Advisor check results file or compare the current check results to the baseline check results.

Syntax

```
[success message] = verifyCheckResult(baseline, checkIDs)
```

Arguments

`baseline`
: Pathname of the baseline check results MAT-file

`checkIDs`
: Cell array of check IDs.

`success`
: Boolean value specifying whether the method succeeded

`message`
: String specifying an error message

Description

If the Model Advisor is in baseline mode (see "setBaselineMode"), this method stores the most recent results of running the checks specified by `checkIDs` in a MAT-file at the location specified by `baseline`. If the method is unable to store the check results at the specified location, it returns `false` in the output variable `success` and the reason for the failure in the output variable `message`. If the Model Advisor is not in baseline mode, this method compares the most recent results of running the checks specified by `checkIDs` with the report specified by `baseline`. If the current results match the baseline results, this method returns`true` as the value of the `success` output variable.

Note You must run the checks specified by `checkIDs` (see "runCheck") before invoking `verifyCheckResult`.

This method enables you to compare the most recent check results generated by the Model Advisor with a baseline set of check results. You can use the method to generate the baseline report as well as perform current-to-baseline result comparisons. To generate a baseline report, put the Model Advisor in baseline mode, using "setBaselineMode". Then invoke this method with the baseline argument set to the location where you want to store the baseline results. To perform a current-to-baseline report comparison, first ensure that the Model Advisor is not in baseline mode (see "getBaselineMode"). Then invoke this method with the path of the baseline report as the value of the `baseline` input argument.

See Also

"setBaselineMode", "getBaselineMode", "runCheck", "verifyCheckResultStatus"

verifyCheckResultStatus

Purpose

Verify that a model has passed or failed a set of checks.

Syntax

```
[success message] = verifyCheckResultStatus(baseline,
checkIDs)
```

Arguments

`baseline`
: Array of Boolean variables

`checkIDs`
: Cell array of check IDs.

`success`
: Boolean value specifying whether the method succeeded

`message`
: String specifying an error message

Description

This method compares the pass/fail (`true`/`false`) statuses from the most recent running of the checks specified by `checkIDs` with the Boolean

values specified by `status`. If the statuses match the baseline, this method returns `true` as the value of the `success` output variable.

Note You must run the checks specified by `checkIDs` (see "runCheck") before invoking `verifyCheckResultStatus`.

See Also

"runCheck"

verifyHTML

Purpose

Generate a baseline Model Advisor report or compare the current report to a baseline report.

Syntax

```
[success message] = verifyHTML(baseline)
```

Arguments

`baseline`
: Pathname of a Model Advisor report

`success`
: Boolean value specifying whether the method succeeded

`message`
: String specifying an error message

Description

If the Model Advisor is in baseline mode (see "setBaselineMode"), this method stores the report most recently generated by the Model Advisor at the location specified by `baseline`. If the method is unable to store a copy of the report at the specified location, it returns `false` in the output variable `success` and the reason for the failure in the output variable `message`. If the Model Advisor is not in baseline mode, this method compares the report most recently generated by the Model Advisor with the report specified by `baseline`. If the current report has

exactly the same content as the baseline report, this method returns `true` as the value of the `success` output variable.

This method enables you to compare a report generated by the Model Advisor with a baseline report to determine if they differ. You can use the method to generate the baseline report as well as perform current-to-baseline report comparisons. To generate a baseline report, put the Model Advisor in baseline mode. Then invoke this method with the baseline argument set to the location where you want to store the baseline report. To perform a current-to-baseline report comparison, first ensure that the Model Advisor is not in baseline mode (see "getBaselineMode"). The invoke this method with the path of the baseline report as the value of the `baseline` input argument.

See Also

"setBaselineMode", "getBaselineMode", "verifyCheckResult"

Purpose

Container for model's signal data logs

Description

Simulink software creates instances of this class to contain signal logs that it creates while simulating a model (see "Logging Signals"). In particular, Simulink software creates an instance of this class for a top model and for each model referenced by the top model that contains signals to be logged. Simulink software assigns the `ModelDataLogs` object for the top 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.ModelDataLogs`

  Container for the data logs of a model

- `Simulink.SubsysDataLogs`

  Container for the data logs of a subsystem

- `Simulink.ScopeDataLogs`

  Container for the data logs of Scope signal viewers

- `Simulink.Timeseries`

  Data log for any signal except a mux or bus signal

- `Simulink.TsArray`

  Data log for a mux or bus signal

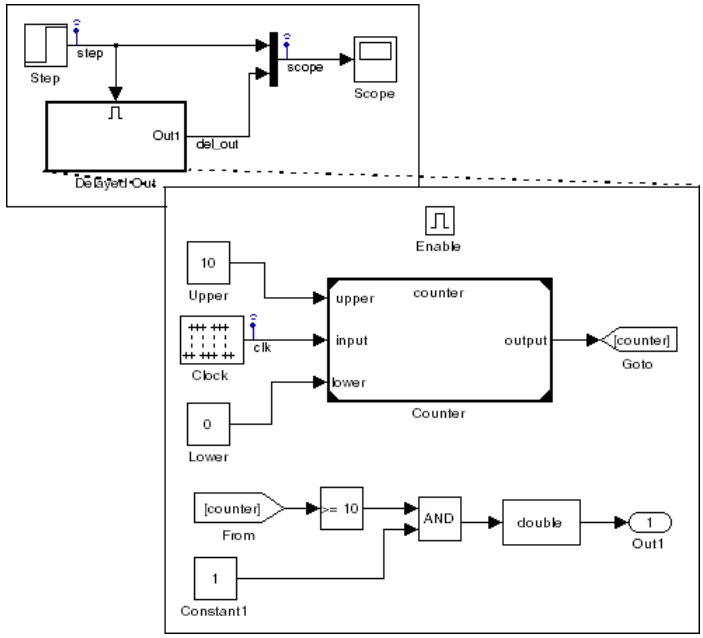
The names of the properties identify the data being logged as follows:

- 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')`. See "Handling Spaces and Newlines in Logged Names" for more information.

Consider, for example, the following model.



As indicated by the testpoint icons, this model specifies that Simulink software 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
```

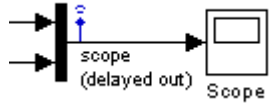
The `logsout` variable contains the signal data logged during the simulation. You can use fully qualified object names or the Simulink `unpack` command to access the signal data stored in `logsout`. For example, to access the amplitudes of the `clk` signal in the Delayed Out subsystem, enter

```
>> data = logsout.('Delayed Out').clk.Data;
```

or

```
>> logsout.unpack('all');
>> data = clk.Data;
```

You can use a custom logging name or signal name when logging a signal. If you use the signal name, and that name occupies more than one line, include an `sprintf('\n')` between the lines of the signal name when accessing the logged data. For example, to access the signal in the following model:



Use the following syntax:

```
logsout.(['scope' sprintf('\n') '(delayed out)'])
```

See "Handling Spaces and Newlines in Logged Names" for more information.

See Also

"Importing and Exporting Data", `Simulink.SubsysDataLogs`, `Simulink.ScopeDataLogs`, `Simulink.Timeseries`, `Simulink.TsArray`, `who`, `whos`, `unpack`

Purpose

Describe model workspace

Description

Instances of this class describe model workspaces. Simulink software creates an instance of this class for each model that you open during a Simulink session. See "Using Model Workspaces" in *Simulink User's Guide* 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 software ignores this property if `DataSource` is not `'MAT-File'`.
`MCode`	`RW`	A string specifying M code used to initialize this workspace. Simulink software 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 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 software and provide information to Simulink software about a Level-2 M-file S-function block. Simulink software creates an instance of this class for each Level-2 M-file S-function block in a model. Simulink software 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 software. 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 *Simulink User's Guide* for more information.

The Level-2 M-file S-Function template `matlabroot/toolbox/simulink/blocks/msfuntmpl.m` shows how to use a number of the following methods.

Parent Class

`Simulink.RunTimeBlock`

Derived Classes

None

Property Summary

Name	Description
"AllowSignalsWithMoreThan2D"	enable Level-2 M-file S-function to use multidimensional signals.

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.

Name	Description
"RegBlockMethod"	Register a callback method for this block.
"RegisterDataTypeFxpBinaryPoint"	Register fixed-point data type with binary point-only scaling.
"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.
"SetAccelRunOnTLC"	Specify whether to use this block's TLC file to generate the simulation target for the model that uses it.
"SetPreCompInpPortInfoToDynamic"	Set precompiled attributes of this block's input ports to be inherited.
"SetPreCompOutPortInfoToDynamic"	Set precompiled attributes of this block's output ports to be inherited.
"SetPreCompPortInfoToDefaults"	Set precompiled attributes of this block's ports to the default values.

Name	Description
"SetSimViewingDevice"	Specify whether block is a viewer.
"WriteRTWParam"	Write custom parameter information to Real-Time Workshop file.

Properties

`AllowSignalsWithMoreThan2D`

Description

Allow Level-2 M-file S-functions to use multidimensional signals. You must set the `AllowSignalsWithMoreThan2D` property in the `setup` method.

Data Type

Boolean

Access

`RW`

`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. For example, the following lines initializes three dialog parameters where the first is tunable, the second in not tunable, and the third is tunable only during simulation.

```
block.NumDialogPrms     = 3;
block.DialogPrmsTunable = {'Tunable','Nontunable','SimOnlyTunable'};
```

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
Use in the `PostPropagationSetup` method to 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 `ProcessParameters`.

See the S-function *matlabroot*/toolbox/simulink/simdemos/adapt_lms.m in the Simulink model sldemo_msfcn_lms.mdl for an example.

`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 software computes the values of block outputs that cannot change during the simulation (see "Constant Sample Time" in the *Simulink User's Guide*). Use this method in the `Outputs` 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 the `Outputs` or `Update` methods.

`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`

Global index of the 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`. The sample time index `stIdx` is a global index for the Simulink model. For example, consider a model that contains three sample rates of 0.1, 0.2, and 0.5, and an M-file S-function block that contains two rates of 0.2 and 0.5. In the M-file S-function, `block.IsSampleHit(0)` returns true for the rate 0.1, not the rate 0.2.

This block method is similar to `ssIsSampleHit` for C-MeX S-functions, however `ssIsSampleHit` returns values based on only the sample times contained in the S-function. For example, if the model described

above contained a C-MeX S-function with sample rates of 0.2 and 0.5, `ssIsSampleHit(S,0,tid)` returns `true` for the rate of 0.2.

Use port-based sample times to avoid using the global sample time index for multi-rate systems (see `Simulink.BlockPortData`).

`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 the `setup` function of 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,
fractionalLength, 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 software 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 software 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 software 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 7-140.

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 Accelerator mode of Simulink software.

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. See the S-function *`matlabroot`*`/toolbox/simulink/blocks/msfcn_times_two.m` in the Simulink model `msfcndemo_timestwo.mdl` for an example.

`SetPreCompInpPortInfoToDynamic`

Purpose

Set precompiled attributes of this block's input ports to be inherited.

Syntax

```
SetPreCompInpPortInfoToDynamic;
```

Description

Initialize the compiled information (dimensions, data type, complexity, and sampling mode) of this block's input ports to be inherited. See the S-function *`matlabroot`*`/toolbox/simulink/simdemos/adapt_lms.m` in the Simulink model `sldemo_msfcn_lms.mdl` for an example.

`SetPreCompOutPortInfoToDynamic`

Purpose

Set precompiled attributes of this block's output ports to be inherited.

Syntax

```
SetPreCompOutPortInfoToDynamic;
```

Description

Initialize the compiled information (dimensions, data type, complexity, and sampling mode) of the block's output ports to be inherited. See the S-function *`matlabroot`*`/toolbox/simulink/simdemos/adapt_lms.m` in the Simulink model `sldemo_msfcn_lms.mdl` for an example.

`SetPreCompPortInfoToDefaults`

Purpose

Set precompiled attributes of this block's ports to the default values.

Syntax

```
SetPreCompPortInfoToDefaults;
```

Description

Initialize the compiled information (dimensions, data type, complexity, and sampling mode) of the block's ports to the default values. By default, a port accepts a real scalar sampled signal with a data type of `double`.

`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 `WriteRTW` 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. See the S-function *matlabroot*/toolbox/simulink/simdemos/adapt_lms.m in the Simulink model sldemo_msfcn_lms.mdl for an example.

Purpose

Specify data type

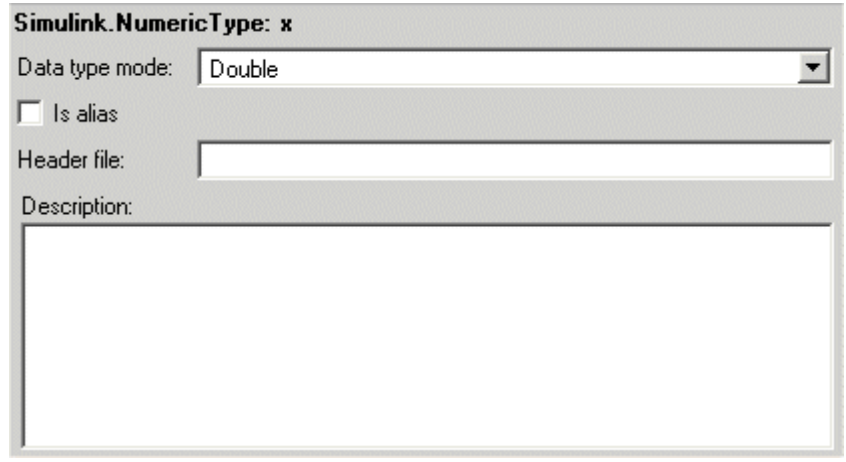
Description

This class lets you specify a data type. To do this,

1. Create an instance of this class in the MATLAB base workspace or a model workspace. To create a numeric type in a model workspace, you must disable the **Is alias** option.
2. Set object properties to the properties of the custom data type
3. Assign the data type to all signals and parameters of your model that you want to conform to the data type.

Assigning a 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.
`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 software to enable controls on the dialog box (see below) that apply to the category and to disable other 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 for a workspace object of this type, Simulink software uses the name of the object as the data type for all objects that specify the object as its data type. Otherwise, Simulink software uses the category of the data type as its name, or, if the category is a fixed-point category, Simulink software generates a name that encodes the type's properties, using the encoding specified by the Simulink Fixed Point product.

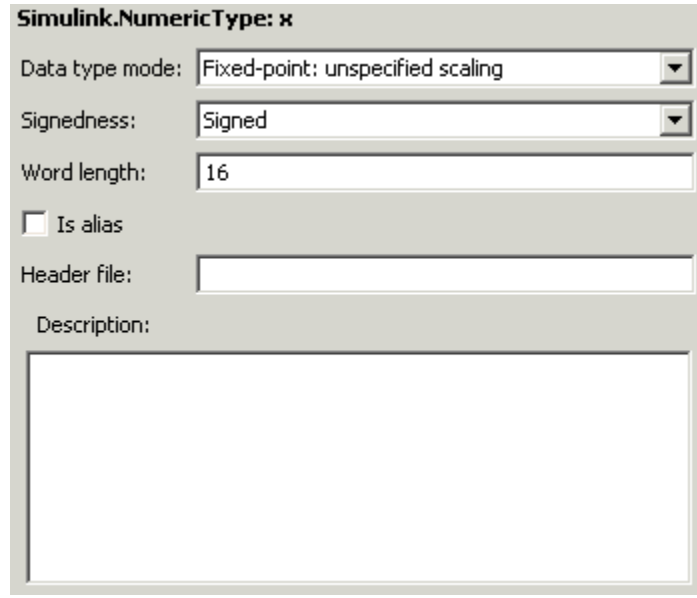
Header file

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 software ignores it.



Signedness

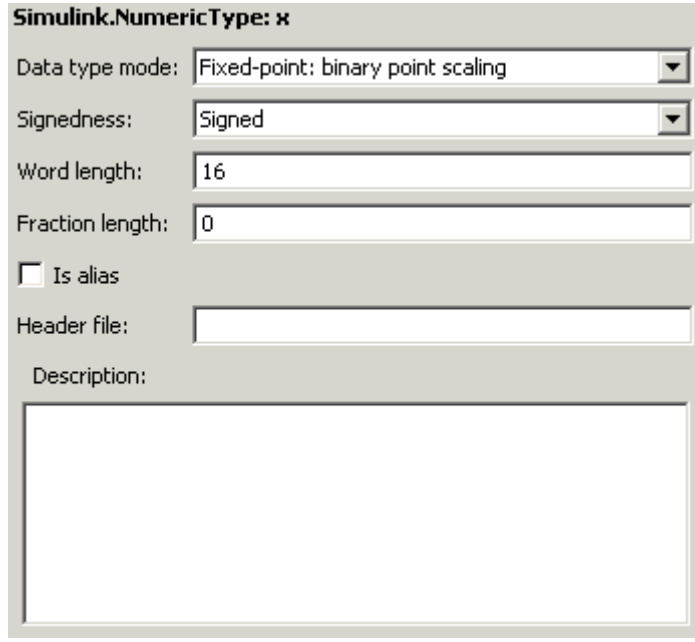
Specifies whether the data type is signed or unsigned, or inherits its signedness. Set the option to `Signed`, `Unsigned`, or `Auto`. This option is enabled only for fixed-point data type categories.

Word-Length

Word length in bits of the fixed-point data type. This option is enabled only for fixed-point data type categories.

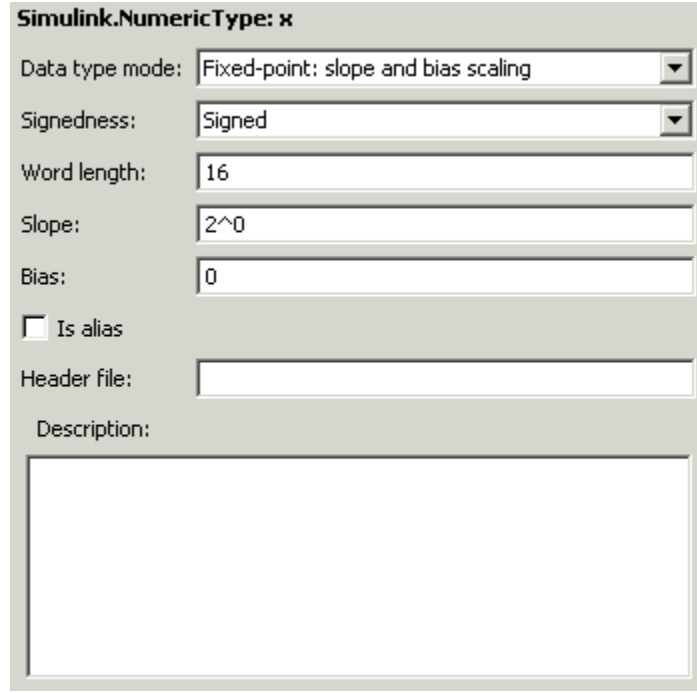
Fraction length

Number of bits to the right of the binary point. This option is enabled only if the data type category is `Fixed-point: binary point scaling`.



Slope

Slope for slope and bias scaling. This option is enabled only if the data type category is `Fixed-point: slope and bias scaling`.



Bias

Bias for slope and bias scaling. This option is enabled only if the data type category is `Fixed-point:  slope and bias scaling`. See the preceding figure.

Properties

Name	Access	Description
`Bias`	`RW`	Bias used for slope and bias scaling of a fixed-point data type. This field is intended for use by the Simulink Fixed Point product. (Bias)

Name	Access	Description
`DataTypeMode`	`RW`	String that specifies the data type of this numeric type. Valid values are `'Double'`, `'Boolean'`, `'Single'`, `'Fixed-point: unspecified scaling'`, `'Fixed-point: binary point scaling'`, and `'Fixed-point: slope and bias scaling'`. (Data type mode)
`Description`	`RW`	Description of this data type. (Description)
`Fixedexponent`	`RW`	exponent used for binary point scaling. This property equals `-FractionLength`. Setting this property causes Simulink software to set the `FractionLength` and `Slope` properties accordingly, and vice versa. This property applies only if the data type category is `Fixed-point: binary point scaling` or `Fixed-point: slope and bias scaling`. It does not appear on the object's Property dialog box, but can be accessed from the MATLAB command prompt.
`FractionLength`	`RW`	Integer that specifies the size in bits of the fractional portion of the fixed-point number. This property equals `-Fixedexponent`. Setting this property causes Simulink software to set the `Fixedexponent` property accordingly, and vice versa. This field is intended for use by the Simulink Fixed Point product. (Fraction length)

Name	Access	Description
`IsAlias`	`RW`	Integer that specifies whether to use the name of this object as the name of the data type that it specifies. Valid values are `1` (yes) or `0` (no). (Is alias)
`Signedness`	`RW`	Boolean that specifies whether this data type is signed, unsigned, or inherits its signedness. Valid values are `1` (signed), `0` (unsigned), or `Auto` (inherit signedness). (Signedness)
`Slope`	`RW`	Slope for slope and bias scaling of fixed-point numbers. This property equals `SlopeAdjustmentFactor * 2^Fixedexponent`. If `SlopeAdjustmentFactor` is 1.0, Simulink software displays the value of this field as `2^SlopeAdjustmentFactor`. Otherwise, it displays it as a numeric value. Setting this property causes Simulink software to set the `Fixedexponent` and `SlopeAdjustmentFactor` properties accordingly, and vice versa. This property appears only if `Category` is `Fixed-point: slope and bias scaling`. (Slope)

Name	Access	Description
`SlopeAdjustmentFactor`	`RW`	Slope for slope and bias scaling of fixed-point numbers. Setting this property causes Simulink software to adjust the Slope property accordingly, and vice versa. This property applies only if `Category` is `Fixed-point:  slope and bias scaling`. It does not appear on the object's Property dialog box, but can be accessed from the MATLAB command prompt.
`WordLength`	`RW`	Integer that specifies the word size of this data type. This field is intended for use by the Simulink Fixed Point product. This property appears only if `Category` is `Fixed-point`. (Word Length)

Purpose

Specify value, value range, data type, and other properties of block parameter

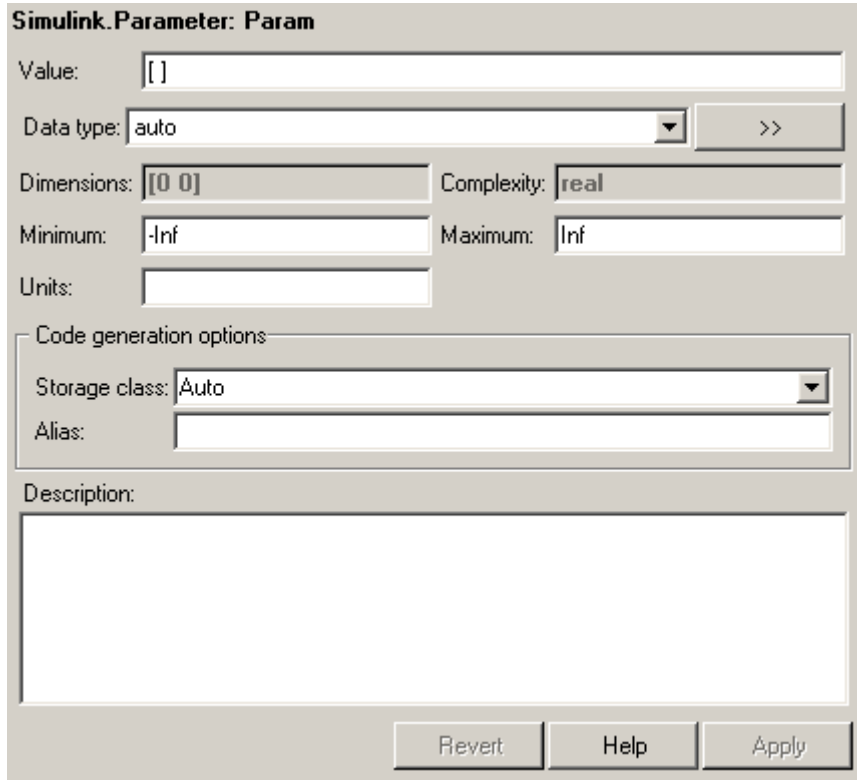
Description

This class enables you to create workspace objects that you can then use as the values of block parameters, e.g., the value of a Gain block's `Gain` parameter. You can create a `Simulink.Parameter` object in the base MATLAB workspace or a model workspace. However, to create the object in a model workspace, you must set the object's storage class to `Auto`.

Parameter objects let you specify not only the value of a parameter but also other information about the parameter, such as the parameter's purpose, its dimensions, its minimum and maximum values, etc. Some Simulink products use this information. For example, Simulink and Real-Time Workshop products use information specified by `Simulink.Parameter` objects to determine whether the parameter is tunable (see "Changing the Values of Block Parameters During Simulation" in *Simulink User's Guide*).

The Simulink software performs range checking of parameter values. The software alerts you when the parameter object's value lies outside a range that corresponds to its specified minimum and maximum values and data type.

Property Dialog Box



Value

Value of the parameter. You can use MATLAB expressions to specify the numeric type, dimensions, and data type of the parameter (see "Data Types Supported by Simulink"). You can also specify fixed-point values for block parameters (see "Specifying Fixed-Point Values Directly" in the Simulink Fixed Point documentation). The following examples illustrate this syntax.

expression	Description
`single(1.0)`	Specifies a single-precision value of 1.0

expression	Description
`int8(2)`	Specifies an 8-bit integer of value 2
`int32(3+2i)`	Specifies a complex value whose real and imaginary parts are 32-bit integers
`fi(2.3,true,16,3)`	Specifies a signed fixed-point numeric object having a value of 2.3, a word length of 16 bits, and a fraction length of 3.

Note If you specify a typed expression as the parameter object's **Value** property, it overrides the current setting of the **Data type** property.

Data type

Data type of the parameter. You can either select a data type from the adjacent pulldown menu or enter a string. If you select `auto` (the default), the block that references the parameter object determines the data type of the variable used to represent this parameter in code generated from the model. If you enter a string, it must evaluate to one of the following:

- A built-in data type that Simulink software supports (see "Data Types Supported by Simulink").
- A `Simulink.NumericType` object
- A `Simulink.AliasType` object

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Data type** parameter. (See "Using the Data Type Assistant" in *Simulink User's Guide*.)

Note If you specify a parameter object's data type using the **Data type** property, it overrides any typed expression in the **Value** property and changes the value to be untyped.

Units

Measurement units in which this value is expressed, e.g., inches. This field is intended for use in documenting this parameter. Simulink software ignores it.

Dimensions

Dimensions of the parameter. Simulink software 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 software 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. Specify a value that evaluates to a scalar, real number with `double` data type. The Simulink software generates a warning if the parameter value is less than the minimum value or if the minimum value is outside the range of the parameter data type. When updating the diagram or starting a simulation, Simulink generates an error in these cases.

Maximum

Maximum value that the parameter can have. Specify a value that evaluates to a scalar, real number with `double` data type. The Simulink software generates a warning if the parameter value is greater than the maximum value or if the maximum value is outside the range of the parameter data type. When updating the diagram or starting a simulation, Simulink generates an error in these cases.

Storage class
Storage class of this parameter. Simulink code generation products use this property to allocate memory for this parameter in generate code. See "Tunable Parameter Storage Classes" in *Real-Time Workshop User's Guide* for more information.

Alias
Alternate name for this parameter. Simulink software ignores this setting.

Description
Description of this parameter. This field is intended for use in documenting this parameter. Simulink software ignores it.

Properties

Name	Access	Description
`Value`	`RW`	Value of this parameter. (Value)
`RTWInfo`	`RW`	Information used by Real-Time Workshop software for generating code for this parameter. The value of this property is an object of `Simulink.ParamRTWInfo` class.
`Description`	`RW`	String that describes this parameter. This property is intended for user use. Simulink software itself does not use it. (Description)
`DataType`	`RW`	String specifying the data type of this parameter. (Data type)
`Min`	`RW`	Minimum value that this parameter can have. (Minimum)
`Max`	`RW`	Maximum value that this parameter can have. (Maximum)
`DocUnits`	`RW`	Measurement units in which this parameter's value is expressed. (Units)

Name	Access	Description
`Complexity`	`RO`	String specifying the numeric type of this parameter. Valid values are `'real'` or `'complex'`. (Complexity)
`Dimensions`	`RO`	Vector specifying the dimensions of this parameter. (Dimensions)

Purpose

Specify information needed to generate code for parameter

Description

Simulink software creates an instance of this class for each instance of a `Simulink.Parameter` object that it creates. Simulink software 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 `myparam`.

```
myparam.RTWInfo.StorageClass = 'exportedGlobal';
```

Property Dialog Box

Use the Code Generation Options section of the `Simulink.Parameter` property dialog box to set the `StorageClass` and `Alias` properties of objects of this class.

Properties

Name	Description
`Alias`	Alternate name for this parameter.
`CustomAttributes`	Custom storage class attributes of this parameter. See “Creating and Using 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 “Parameter Considerations” 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 block while simulation is running

Description

This class allows a Level-2 M-file S-function or other M program to obtain information about a block. Simulink software creates an instance of this class or a derived class for each block in a model. Simulink software 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 software. 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 the Simulink documentation for more information.

Note `Simulink.RunTimeBlock` objects do not support MATLAB sparse matrices. For example, the following line of code attempts to assign a sparse identity matrix to the run-time object's output port data. This line of code in a Level-2 M-file S-function produces an error:

```
block.Outport(1).Data = speye(10);
```

Parent Class

None

Derived Classes

`Simulink.MSFcnRunTimeBlock`

Property Summary

Name	Description
"BlockHandle"	Block's handle.
"CurrentTime"	Current simulation time.

Name	Description
“NumDworks”	Number of discrete work vectors used by the block.
“NumOutputPorts”	Number of block output ports.
“NumContStates”	Number of block’s continuous states.
“NumDworkDiscStates”	Number of block’s discrete states
“NumDialogPrms”	Number of parameters that can be entered on S-function block’s dialog box.
“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”	Determine whether a data type is fixed point.
“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.

Name	Description
"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`

`NumDworkDiscStates`

Description

Number of discrete states. In an M-file S-function, you need to use DWorks to set up discrete states.

Access

`RW`

See Also

`ssGetNumDiscStates`

`NumDialogPrms`

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 run-time parameters used by this block. See "Run-Time Parameters" for more information.

Access

RW

See Also

ssGetNumSFcnParams

SampleTimes

Description

Block'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

```
derivs = Derivatives();
```

Description

Get vector of state derivatives.

See Also

`ssGetdX`

`DialogPrm`

Purpose

Get an S-function's dialog parameters.

Syntax

```
param = DialogPrm(pIdx);
```

Arguments

`pIdx`

Integer value specifying the index of the parameter to be returned.

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(dwIdx);
```

Arguments

`dwIdx`

Integer value specifying the index of a work vector.

Description

Get information about the DWork vector specified by `dwIdx` where `dwIdx` 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 Fixed-Point Toolbox software. 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);
```

Arguments

`pIdx`

Integer value specifying the index of an input port.

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.BlockPreComp-InputPortData` 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);
```

Arguments

`pIdx`

Integer value specifying the index of an output port.

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.BlockPreComp-OutputPortData` or `Simulink.BlockCompOutputPortData`, depending on whether the model that contains the port is uncompiled or compiled, respectively. You can use this object to get and set the output port's uncompiled or compiled properties, respectively.

See Also

`ssGetInputPortSignalPtrs`, `Simulink.BlockPreComp-OutputPortData`, `Simulink.BlockCompOutputPortData`

`RuntimePrm`

Purpose

Get an S-function's run-time parameters.

Syntax

```
param = RuntimePrm(pIdx);
```

Arguments

`pIdx`

Integer value specifying the index of a run-time parameter.

Description

Get the run-time parameter whose index is `pIdx`.

See Also

`ssGetRunTimeParamInfo`

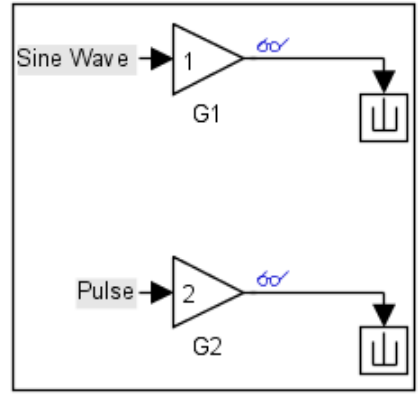
Purpose

Store data logged by Scope signal viewer

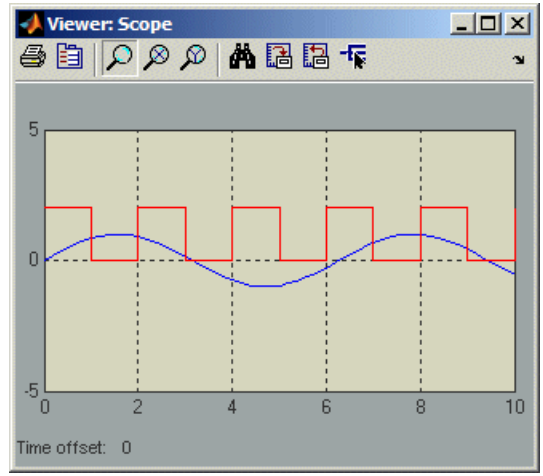
Description

Simulink software creates instances of this class to log data displayed on Scope viewers (see "Visualizing Simulation Results"). In particular, if you have enabled data logging for a model, Simulink software 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.

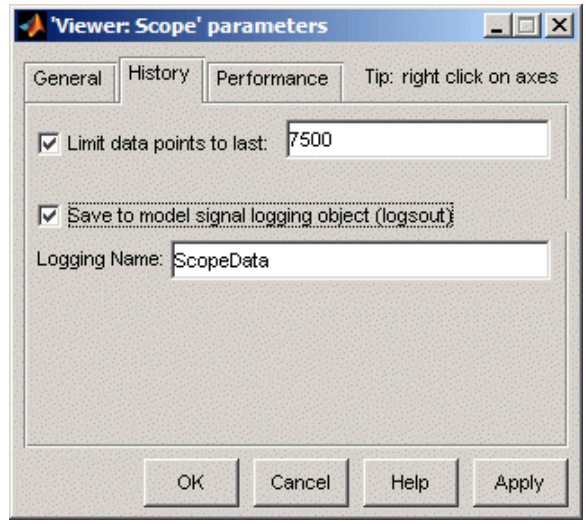
Consider, for example, the following model:



This model displays signals `out1` and `out2` on a 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`.

You can use Simulink data object dot notation to access the data, e.g.,

```
>> logsout.ScopeData.axes1

ans =

Simulink.ScopeDataLogs (axes1):
  Name                    elements  Simulink Class

  out1                       1      Timeseries
  out2                       1      Timeseries
```

See Also

"Importing and Exporting Data", `Simulink.ModelDataLogs`, `Simulink.SubsysDataLogs`, `Simulink.Timeseries`, `Simulink.TsArray`, `who`, `whos`, `unpack`

Simulink.Signal

Purpose

Specify attributes of signal

Description

This class enables you to create workspace objects that you can use to assign or validate the attributes of a signal or discrete state, such as its data type, numeric type, dimensions, and so on. You can create a `Simulink.Signal` object in the MATLAB workspace or in a model workspace. If you create the object in a model workspace, you must set the object's storage class to `Auto`.

Objects of this class allow you to assign or validate signal or discrete state attributes by giving the signal or discrete state the same name as the workspace variable that references the `Simulink.Signal` object. For brevity, the rest of section refers only to specifying and validating signal attributes. The same techniques work with discrete states also.

You can use a variety of techniques to associate a signal object with a signal. For examples, see "Using Signal Objects to Initialize Signals and Discrete States", "Using Signal Objects to Tune Initial Values", and "Applying CSCs to Parameters and Signals".

Multiple Signal Objects

A given signal object can be associated with more than one signal only if the storage class of the signal object is `Auto`. If the object's storage class is other than `Auto`, it can be associated with at most one signal.

A given signal can be associated with at most one signal object under any circumstances. The signal can refer to the object more that once, but every reference must resolve to exactly the same object. A different signal object that has exactly the same properties will not meet the requirement for uniqueness.

A compile-time error occurs if a model associates more than one signal object with any signal. To prevent the error, decide which object the signal will use, then delete or reconfigure all references to any other signal objects, so that all remaining references resolve to the chosen signal object. See "Displaying Signal Sources and Destinations" for a description of techniques that you can use to trace the full extent of a signal.

Using Signal Objects to Assign Signal Attributes

You can use a signal object to assign values to signal attributes that are left unassigned (have a value of `-1` or `auto`) by the signal source. To use a signal object to assign signal attribute values:

- Create a `Simulink.Signal` object that has the same name as the signal.
- Set the properties of the object that correspond to the attributes left unspecified by the signal source.
- enable explicit or implicit signal resolution:
  - **explicit resolution:** In the signal's Signal Properties dialog box, enable **Signal name must resolve to Simulink signal object**. This is the preferred technique. See "Explicit and Implicit Symbol Resolution" for more information.
  - **Implicit resolution:** Set the model's **Configuration Parameters > Diagnostics > Data Validity > Signal resolution** option to `explicit and implicit` or `explicit and warn implicit`. The MathWorks recommends using explicit resolution only.

The signal object then provides the value of each signal attribute to which the signal source assigns a value of `-1` or `auto`. (The same object could also validate any attribute to which the signal source assigns a specific value, rather than `-1` or `auto`, as described in "Using Signal Objects to Validate Signal Attributes" on page 7-152.)

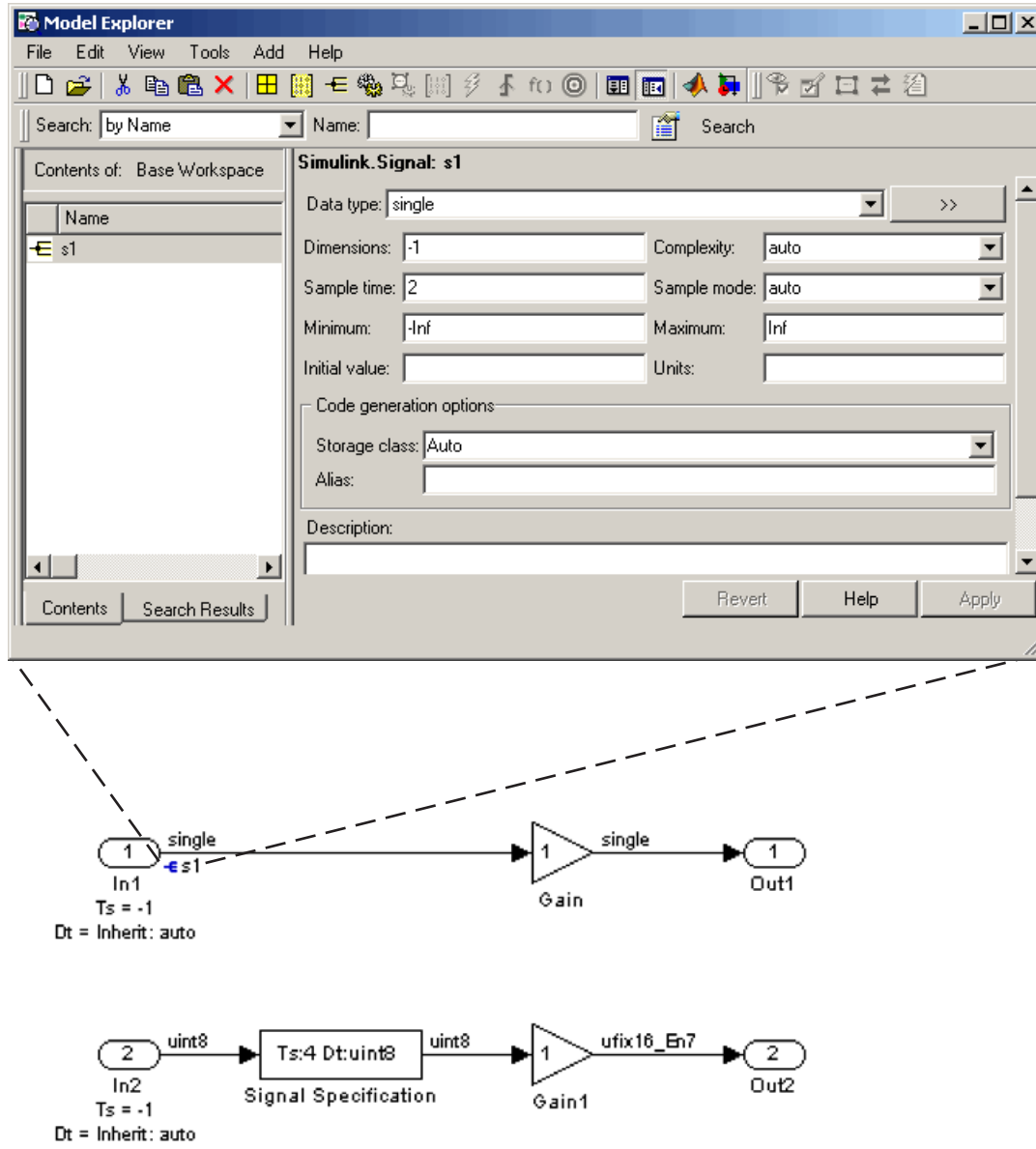
Using a Signal Specification Block

You can use a `Signal Specification` block rather than a `Simulink.Signal` object to assign properties left unspecified by a signal source. each technique has advantages and disadvantages:

- Using a signal object simplifies the model and allows you to change signal property values without editing the model, but does not show signal property values directly in the block diagram.

- Using a Signal Specification block displays signal property values directly in the block diagram, but complicates the model and requires editing it to change signal property values.

The following model illustrates the respective advantages of the two ways of assigning attributes to a signal.


Model Explorer
File Edit View Tools Add Help
Search: by Name
Name:
Search
Contents of: Base Workspace
Name
s1
Simulink.Signal: s1
Data type: single
Dimensions: -1
Complexity: auto
Sample time: 2
Sample mode: auto
Minimum: -Inf
Maximum: Inf
Initial value:
Units:
Code generation options
Storage class: Auto
Alias:
Description:
Revert
Help
Apply
Contents
Search Results
1
In1
Ts = -1
Dt = Inherit: auto
single
s1
1
Gain
single
1
Out1
2
In2
Ts = -1
Dt = Inherit: auto
uint8
Ts:4 Dt:uint8
Signal Specification
uint8
1
Gain1
ufix16_En7
2
Out2

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, using 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 Validate Signal Attributes

You can use a signal object to validate signal attributes whose values are explicitly assigned by the signal source. Such attributes have values other than `-1` or `auto`. Successful validation guarantees that the signal has the attributes that you intended it to have. To use a signal object to validate signal attributes:

- Create a `Simulink.Signal` object that has the same name as the signal.
- Set properties of the object to the values that corresponding signal attributes must have.
- enable explicit or implicit signal resolution:
  - **explicit resolution:** In the signal's Signal Properties dialog box, enable **Signal name must resolve to Simulink signal object**. This is the preferred technique. See "Explicit and Implicit Symbol Resolution" for more information.
  - **Implicit resolution:** Set the model's **Configuration Parameters > Diagnostics > Data Validity > Signal resolution** option to `explicit and implicit` or `explicit and warn implicit`. The MathWorks recommends using explicit resolution only.

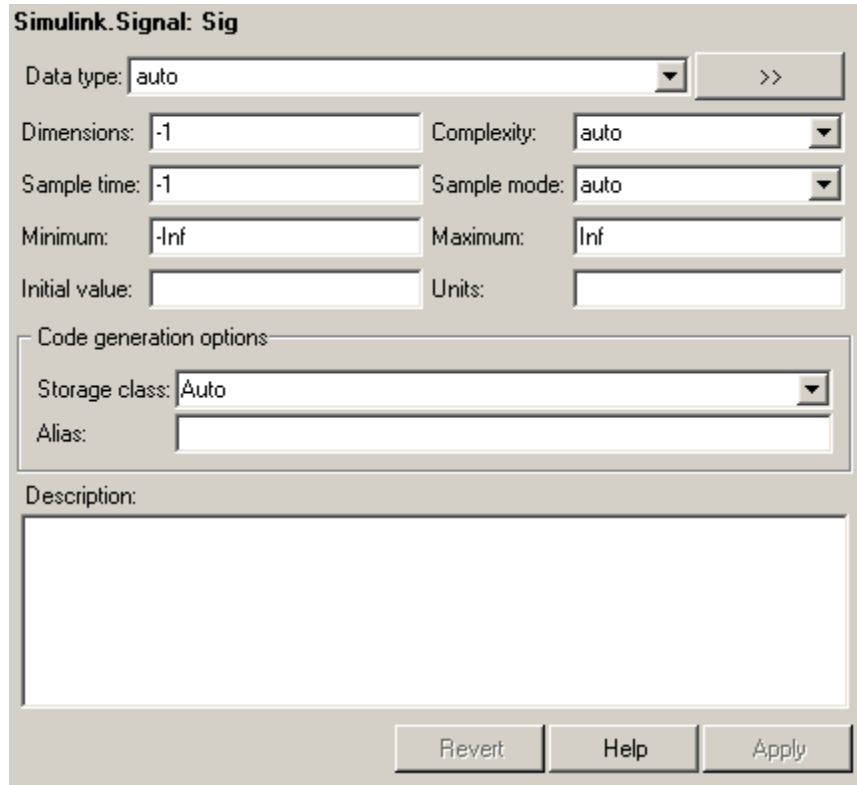
If the signal source assigns any value other than `-1` or `auto` to a signal property, and the assigned value differs from the corresponding signal object value, the signal does not match the signal object and is therefore invalid. (The same object could also provide values for attributes that the signal source defines as `-1` or `auto`, as described in "Using Signal Objects to Assign Signal Attributes" on page 7-149.)

The result when a signal does not match a signal object can depend on several factors. The Simulink engine can validate a signal property when you update the diagram, while you run a simulation, or both. When and how validation occurs can depend on internal rules that are subject to change, and sometimes on configuration parameter settings.

Not all signal validation compares signal source attributes with signal object properties. For example, if you specify Minimum and Maximum signal values using a signal object, the signal source must specify the same values as the signal object (or inherit the values from the object) but such validation relates only to agreement between the source and the object, not to enforcement of the Minimum and Maximum values during simulation.

If the value of **Configuration Parameters > Diagnostics > Data Validity > Simulation range checking** is `none` (the default) the Simulink engine does not enforce any signal's Minimum and Maximum values during simulation, even though a signal object provided or validated them. To enforce Minimum and Maximum signal values during simulation, you must set **Simulation range checking** to `warning` or `error`. See "Checking Signal Ranges" and "Diagnostics Pane: Data Validity" for more information.

Property Dialog Box



Data type

Data type of the signal. The default entry, `auto`, specifies that Simulink software should determine the data type. Use the adjacent pulldown 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.

Click the **Show data type assistant** button  to display the **Data Type Assistant**, which helps you set the

Data type parameter. (See "Using the Data Type Assistant" in *Simulink User's Guide*.)

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 software), `real`, or `complex`.

Sample time

Rate at which the value of this signal should be computed. See "How to Specify the Sample Time" in the Simulink documentation for information on how to specify the sample time.

Sample mode

Sample mode of this signal. Simulink software ignores the setting of this field.

Minimum

Minimum value that the signal should have. Specify a value that evaluates to a scalar, real number with `double` data type. Simulink software uses this value in the following ways:

- When updating the diagram or starting a simulation, Simulink generates an error if the signal's initial value is less than the minimum value or if the minimum value is outside the range of the signal's data type.
- When the **Simulation range checking** diagnostic is enabled, Simulink alerts you during simulation if the signal's value is less than the minimum value (see "Simulation range checking").

Maximum

Maximum value that the signal should have. Specify a value that evaluates to a scalar, real number with `double` data type. Simulink software uses this value in the following ways:

- When updating the diagram or starting a simulation, Simulink generates an error if the signal's initial value is greater than the maximum value or if the maximum value is outside the range of the signal's data type.
- When the **Simulation range checking** diagnostic is enabled, Simulink alerts you during simulation if the signal's value is greater than the maximum value (see "Simulation range checking").

Initial value

Signal or state value before a simulation takes its first time step. You can specify any MATLAB string expression that evaluates to a `double` numeric scalar value or array.

Valid:

```
1.5
[1 2 3]
1+0.5

foo = 1.5;
s1.InitialValue = 'foo';
```

Invalid:

```
uint(1)
foo = '1.5';
s1.InitialValue = 'foo';
```

If necessary, Simulink software converts the initial value to ensure type, complexity, and dimension consistency with the corresponding block parameter value. If you specify an invalid value or expression, an error message appears when you update the model. Also, Simulink performs range checking of the initial value. The software alerts you when the signal's initial value lies outside a range that corresponds to its specified minimum and maximum values and data type.

Initial value settings for signal objects that represent the following signals and states override the corresponding block parameter initial values if undefined (specified as `[]`):

- Output signals of conditionally executed subsystems and Merge blocks
- Block states

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 software ignores it.

Storage class

Storage class of this signal. See "Tunable Parameter Storage Classes" in the Real-Time Workshop User's Guide for more information.

Alias

Alternate name for this signal. Simulink software ignores this setting. This property is used for code generation.

Description

Description of this signal. This field is intended for use in documenting this signal. This property is used by the Simulink Report Generator and for code generation.

Properties

Name	Access	Description
`RTWInfo`	`RW`	Information used by Real-Time Workshop software for generating code for this signal. The value of this property is an object of `Simulink.SignalRTWInfo` class.
`Description`	`RW`	Description of this signal. This field is intended for use in documenting this signal. (Description)

Name	Access	Description
`DataType`	`RW`	String specifying the data type of this signal. (Data type)
`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)
`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)
`SampleTime`	`RW`	Rate at which this signal should be updated. (Sample time)
`Sampling Mode`	`RW`	Sampling mode of this signal. (Sample mode)
`InitialValue`	`RW`	Signal or state value before a simulation takes its first time step. (Initial Value)

Purpose

Specify information needed to generate code for signal

Description

Simulink software creates an instance of this class for each instance of a `Simulink.Signal` object that it creates. Simulink software uses the `Simulink.SignalRTWInfo` object to store information needed to generate code for the signal specified by the `Simulink.Signal` object.

You can set the properties of an instance of this class via the `RTWInfo` property or the property dialog box of the `Simulink.Signal` object that uses it. For example, the following MATLAB expression sets the `StorageClass` property of a `Simulink.SignalRTWInfo` object used by a signal object name `mysignal`.

```
mysignal.RTWInfo.StorageClass = 'exportedGlobal';
```

Property Dialog Box

Use the Code Generation Options section of the `Simulink.Signal` property dialog box to set the `StorageClass` and `Alias` properties of objects of this class.

Properties

Name	Description
`Alias`	Alternate name for this signal.
`CustomAttributes`	Custom storage class attributes of this signal. See “Creating and Using Custom Storage Classes” in the Real-Time Workshop Embedded Coder documentation for more information.
`CustomStorageClass`	Custom storage class of this signal.
`StorageClass`	Storage class of this signal. See “Signal Considerations”in the Real-Time Workshop documentation for more information.

Simulink.Structelement

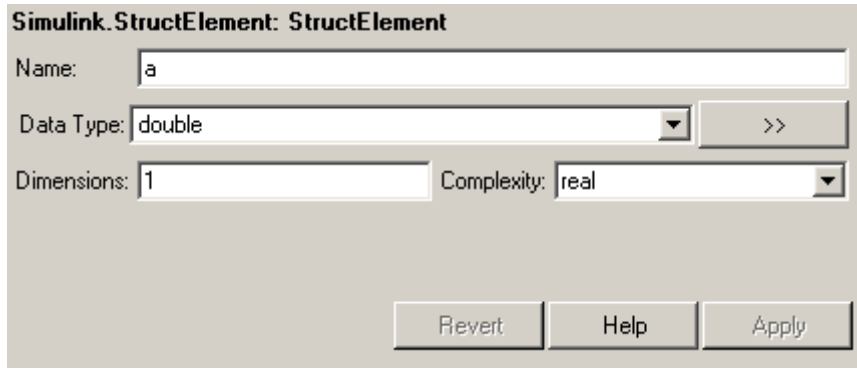
Purpose

Describe element of data structure

Description

Objects of this class describe elements of structures described by objects of the `Simulink.StructType` class.

Property Dialog Box



Name

Specify a name for the element.

Data type

Specify a data type for this element. You can either select a data type from the adjacent pulldown list or enter a string. If you enter a string, it must evaluate to one of the following:

- A built-in data type supported by Simulink software (see "Data Types Supported by Simulink")
- A `Simulink.NumericType` object
- A `Simulink.AliasType` object

Click the **Show data type assistant** button 

to display the **Data Type Assistant**, which helps you set the **Data type** parameter. (See "Using the Data Type Assistant" in *Simulink User's Guide*.)

Dimensions
Specify a vector that represents the dimensions of the element.

Complexity
Specify the numeric type (i.e., real or complex) of this element.

Properties

Name	Access	Description
`Name`	`RW`	String specifying the name of this element. (Name)
`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)
`Dimensions`	`RW`	A vector specifying the dimensions of this element. (Dimensions)

See Also

`Simulink.StructType`

Simulink.StructType

Purpose

Describe data structure used as value of 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. You must create structure types in the MATLAB workspace. If you attempt to create a structure type in a model workspace, Simulink software displays an error.

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.

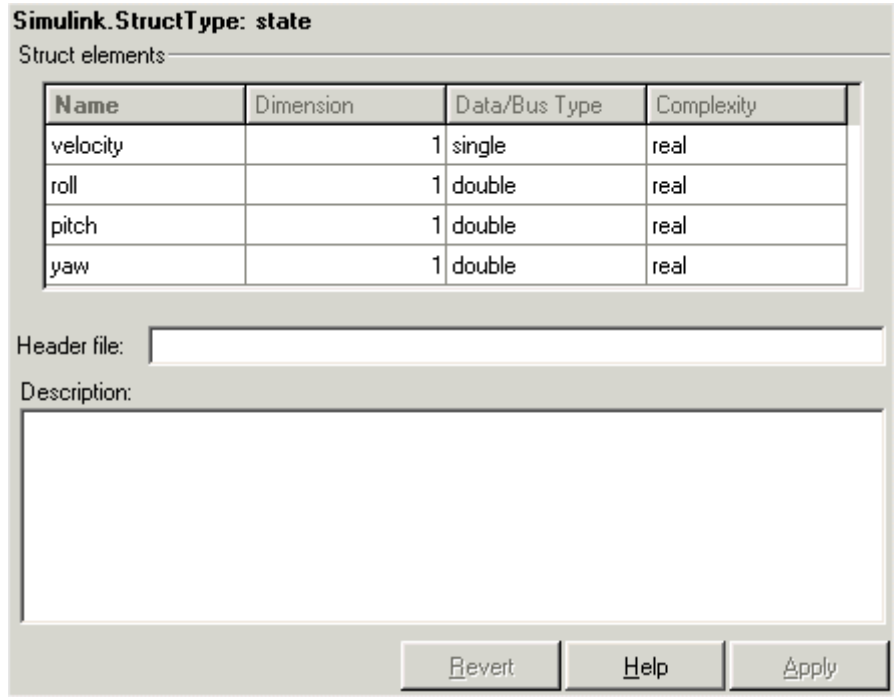
```
v = Simulink.Structelement;
v.Name = 'v';
v.DataType = 'single';
n = Simulink.Structelement;
n.Name = 'n';
n.DataType = 'uint8';

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 `simstruc.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 software. Simulink software ignores it.

Description

Description of this structure. This field is intended for you to use to document this structure. Simulink software 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 software 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`

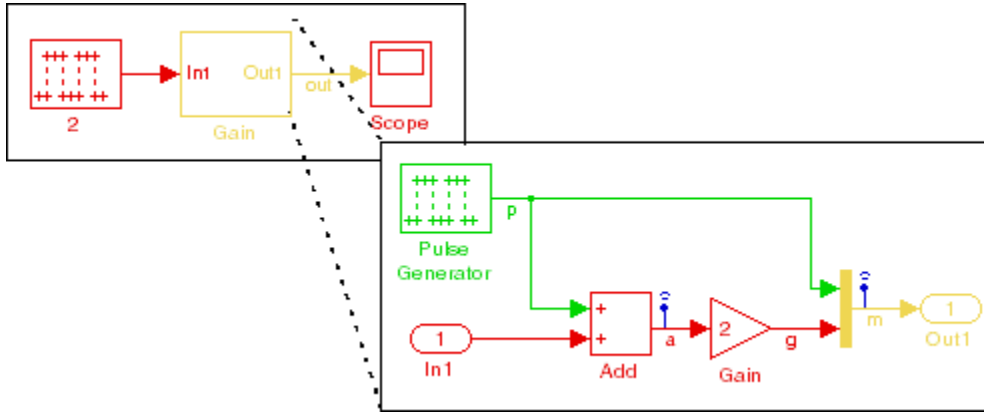
Purpose

Container for subsystem's signal data logs

Description

Simulink software creates instances of this class to contain logs for signals belonging to a subsystem (see "Logging Signals" in the Simulink documentation). 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 `m` 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

"Importing and Exporting Data", `Simulink.ModelDataLogs`, `Simulink.ScopeDataLogs`, `Simulink.Timeseries`, `Simulink.TsArray`, `who`, `whos`, `unpack`

Purpose

Provide information about time data in `Simulink.Timeseries` object

Description

Simulink software 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., `'seconds'`, 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. If the signal is aperiodic (continuous signal with variable-step solver), this property has a value of `NaN`. A signal is periodic if it has a discrete sample time (not continuous or constant) or is continuous with a fixed-step solver.
`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`

Purpose

Store data for any signal except mux or bus signal

Description

Simulink software creates instances of this class to store signal data that it logs for any signal except a mux or bus signal, which are stored in a `Simulink.TsArray`. See "Logging Signals" for more information. The MATLAB Time Series Tools can import and manipulate instances of this class, as described in Using Time Series Tools in the MATLAB Data Analysis documentation.

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 mux or a virtual bus; 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

"Importing and Exporting Data",
`Simulink.TimeInfo`, `Simulink.ModelDataLogs`,
`Simulink.SubsysDataLogs`, `Simulink.ScopeDataLogs`,
`Simulink.TsArray`, `who`, `whos`, `unpack`

Simulink.TsArray

Purpose

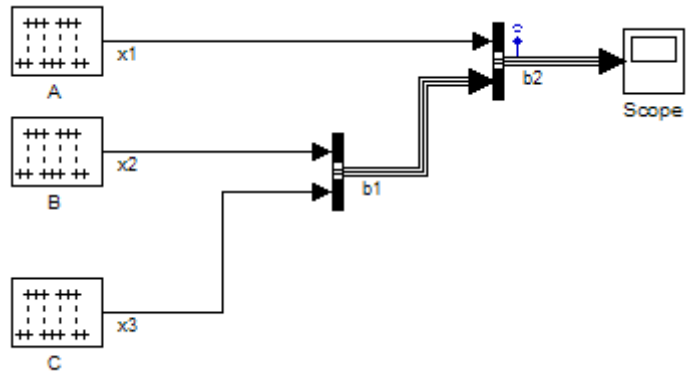
Store data for mux or bus signal

Description

Simulink software creates instances of this class to contain the data that it logs for a mux or bus signal. Other types of signals are stored in a `Simulink.Timeseries`. See "Logging Signals" for more information.

Objects of the `Simulink.TsArray` class have a variable number of properties. The first property, called `Name`, specifies the log name of the logged signal. The remaining properties reference logs for the elements of the logged signal: `Simulink.Timeseries` objects for elementary signals and `Simulink.TSArray` objects for mux or bus signals. The name of each property is the log name of the corresponding signal.

For example, consider the following model.



This model specifies that Simulink software should log the values of the signal `b2` during simulation. After simulation of this model, the MATLAB workspace contains a `Simulink.ModelDataLogs` object, named `logsout`, that contains a `Simulink.TsArray object`, named `b2`, that contains the logs for the elements of `b2`, i.e., for the elementary signal `x1` and the bus signal `b1`. entering the fully qualified name of the `Simulink.TsArray` object, i.e., `logsout.b2`, at the MATLAB command line reveals the structure of the signal log for this model.

```
>> logsout.b2
```

```
Simulink.TsArray (untitled/Bus Creator1):
  Name                    elements   Simulink Class

  x1                         1       Timeseries
  b1                         2       TsArray
```

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.b2.x1.Data;
```

or

```
>> logsout.unpack('all');
data = x1.Data;
```

See Also

"Importing and Exporting Data",`Simulink.ModelDataLogs`, `Simulink.SubsysDataLogs`, `Simulink.ScopeDataLogs`, `Simulink.Timeseries`, `who`, `whos`, `unpack`

Simulink.Variant

Purpose

Specifies a model reference variant and its execution environment

Description

A `Simulink.Variant` object specifies a Boolean expression called a *variant condition*. Each Model block that uses the object associates it with an execution environment consisting of parameter names and values (if the model is parameterized) and an execution mode: `Accelerator`, `Normal`, or `PIL`.

A variant condition is not an object, but an attribute whose value exists separately in each variant object. The expression references MATLAB variables and/or Simulink parameter objects in the base workspace, and must evaluate to `true` or `false` when the model is compiled.

When a Model block that uses model reference variants lists a given variant object among its candidate variant objects, the block references the associated model and simulates it in the associated execution environment if:

- That variant object's variant condition is `true`.
- The variant conditions of all other variant objects associated with the block are `false`.

The variant object whose variant condition is `true` is the *active variant* for that Model block. Any number of Model blocks can reference the same variant object, and each can associate a different model and execution environment with the object. Furthermore, different variant objects that specify identical variant conditions can specify:

- The same model with different execution environments
- Different models, each with its own execution environment.

When the variant condition that is common to all the variant objects is `true`, each variant object that specifies that condition becomes the active variant in its Model block. The block will then use that object's model and execution environment in simulation.

You can use this capability to globally customize a model hierarchy to represent any number of contexts. The customization requires only changing the values of the base workspace variables and parameters that appear in variant conditions, thereby changing the active variant in any or all Model blocks.

Construction

variant=Simulink.Variant(*VariantCondition*) creates a variant object in the base workspace. The object's name is *variant*, and the associated variant condition is *VariantCondition*. The object can have any unique legal MATLAB name.

The *VariantCondition* must be a Boolean expression that references at least one base workspace variable or parameter. A variant condition can include scalar variables, enumerated values, the operators ==, !=, &&, ||, and ~, and parentheses if needed for grouping. If you specify the condition literally, surround it with single quotes.

You can also create a variant object using the Model explorer. Select the Base Workspace, choose **Add > Simulink.Variant**, and specify the desired name and variant condition in the Contents and Dialog panes. If you specify the condition literally, do *not* surround it with single quotes, which are required only by the MATLAB API.

Properties

Condition

Description

The Boolean expression associated with the variant object.

Access

RW

Examples

Create a variant object with a variant condition:

```
GU=Simulink.Variant('Fuel==1 && Emis==1')
```

See Also

- "Referencing a Model"
- "Using Model Reference Variants"

- “Generating Code Variants for Variant Models”

8

Model and Block Parameters

Model Parameters

In this section...

About Model Parameters

The following sections list parameters that you can set for Simulink models and blocks, using the `set_param` command.

This table lists and describes, in alphabetical order, parameters that describe a model. The table also includes model callback parameters (see "Using Callback Functions"). The **Description** column indicates where you can set the value on a dialog box. For examples, see "Examples of Setting Model Parameters" on page 8-86.

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 in Alphabetical Order

Parameter	Description	Values
`AbsTol`	Specify the largest acceptable solver error, as the value of the measured state approaches zero. Set by **Absolute tolerance** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'auto'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`AccelVerboseBuild`	Controls the verbosity level during code generation for Simulink Accelerator mode, model reference Accelerator mode, and Rapid Accelerator mode. Set by **Verbose accelerator builds** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `{'off'} | 'on'`
`AlgebraicLoopMsg`	Specifies diagnostic action to take when there is an algebraic loop. Set by **Algebraic loop** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`ArrayBoundsChecking`	Select the diagnostic action to take when blocks write data to locations outside the memory allocated to them. Set by **Array bounds exceeded** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ArtificialAlgebraic-LoopMsg`	Specifies diagnostic action to take if algebraic loop minimization cannot be performed for a subsystem because an input port of that subsystem has direct feedthrough. Set by **Minimize algebraic loop** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`AssertControl`	Enable model verification blocks in the current model either globally or locally. Set by **Model Verification block enabling** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'UseLocalSettings'} | 'EnableAll' | 'DisableAll'`
`AutoInsertRateTranBlk`	Specify whether Simulink software inserts hidden Rate Transition blocks between blocks that have different sample rates. Set by **Automatically handle rate transition for data transfer** on the **Solver** pane of the Configuration Parameters dialog box.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`BlockDescription-StringDataTip`	Specifies whether to display the user description string for a block as a data tip. Set by **User Description String** on the **View > Block Data Tips Options** menu of the Model Editor.	string — `'on' | {'off'}`
`BlockNameDataTip`	Specifies whether to display the block name as a data tip. Set by **Block Name** on the **View > Block Data Tips Options** menu of the Model Editor.	string — `'on' | {'off'}`
`BlockParametersDataTip`	Specifies whether to display a block parameter in a data tip. Set by **Parameter Names and Values** on the **View > Block Data Tips Options** menu of the Model Editor.	string — `'on' | {'off'}`
`BlockPriorityViolationMsg`	Select the diagnostic action to take if Simulink software detects a block priority specification error. Set by **Block priority violation** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'warning'} | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`BlockReduction`	Enables block reduction optimization. Set by **Block reduction** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`BlockReductionOpt`	See `BlockReduction` parameter for more information.	
`BooleanDataType`	Enable Boolean mode. Set by **Implement logic signals as Boolean data (vs. double)** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`BrowserLookUnderMasks`	Show masked subsystems in the Model Browser. Set by **Show Masked Subsystems** on the **View > Model Browser Options** menu of the Model Editor.	string — `'on' | {'off'}`
`BrowserShowLibraryLinks`	Show library links in the Model Browser. Set by **Show Library Links** on the **View > Model Browser Options** menu of the Model Editor.	string — `'on' | {'off'}`
`BufferReusableBoundary`	For internal use.	

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`BufferReuse`	Enable reuse of block I/O buffers. Set by **Reuse block outputs** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`BusObjectLabelMismatch`	Select the diagnostic action to take if the name of a bus element does not match the name specified by the corresponding bus object. Set by **Element name mismatch** on the **Connectivity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`CheckExecutionContext-PreStartOutputMsg`	Specify whether to display a warning if Simulink software detects potential initial output differences from previous releases. Set by **Check preactivation output of execution context** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`CheckExecutionContext-RuntimeOutputMsg`	Specify whether to display a warning if Simulink software detects potential output differences from previous releases. Set by **Check runtime output of execution context** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'on' | {'off'}`
`CheckForMatrixSingularity`	See `CheckMatrixSingularityMsg` parameter for more information.	
`CheckMatrixSingularityMsg`	Select the diagnostic action to take if the Product block detects a singular matrix while inverting one of its inputs in matrix multiplication mode. Set by **Division by singular matrix** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`CheckModelReference-TargetMessage`	Select the diagnostic action to take if Simulink software detects a target that needs to be rebuilt. Set by **Never rebuild targets diagnostic** on the **Model Referencing** pane of the Configuration Parameters dialog box.	string — `'none' | 'warning' | {'error'}`
`CheckSSInitialOutputMsg`	Enable checking for undefined initial subsystem output. Set by **Check undefined subsystem initial output** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`CloseFcn`	Set the close callback function, which can be a command or a variable. Set by **Model close function** on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" for more information.	string — `{''}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ConditionallyExecute-Inputs`	Enable conditional input branch execution optimization. Set by **Conditional input branch execution** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`ConfigurationManager`	Configuration manager for this model.	string — `{'None'}`
`ConsecutiveZCsStepRelTol`	Relative tolerance associated with the time difference between zero-crossing events. Set by **Time tolerance** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'10*128*eps'}`
`ConsistencyChecking`	Select the diagnostic action to take if S-functions have continuous sample times, but do not produce consistent results when executed multiple times. Set by **Solver data inconsistency** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`CovCompData`	If `CovHTMLOptions` is set to `off` and `CovCumulativeReport` is set to `on`, this parameter specifies `cvdata` objects containing additional model coverage data to include in the model coverage report. Set by **Additional data to include in report (cvdata objects)** on the **Report** pane of the Coverage Settings dialog box.	string — `{''}`
`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`, the Simulink Verification and Validation software displays the coverage results from successive simulations in the report. If set to `off`, the Simulink Verification and Validation software displays the coverage results for the last simulation in the report. Set by the radial buttons **Cumulative runs** (on) / **Last run** (off) on the **Report** pane	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
	of the Coverage Settings dialog box.	
`CovCumulativeVarName`	If `CovSaveCumulativeToWorkspaceVar` is set to `on`, the Simulink Verification and Validation software saves the results of successive simulations in the workspace variable specified by this property. Set by **cvdata object name** below the selected **Save cumulative results in workspace variable** check box on the **Results** pane of the Coverage Settings dialog box.	string — `{'covCumulativeData'}`
`CovExternalEMLEnable`	Enables coverage for any external M-file functions that Embedded MATLAB functions call in your model. The functions may be defined in an Embedded MATLAB Function block or in a Stateflow chart. Enable this feature by checking **Coverage for External Embedded MATLAB Files** on the Coverage Settings dialog box.	`'on' | {'off'}`

Model Parameters in Alphabetical Order (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. On the **Report** pane of the Coverage Settings dialog box, select **Settings** to open 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'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
		Include cyclomatic complexity numbers in block details
`CovHTMLReporting`	Set to `on` to tell the Simulink Verification and Validation software to create an HTML report containing the coverage data in the MATLAB Help browser at the end of simulation. Set by **Generate HTML report** on the **Report** pane of the Coverage Settings dialog box.	string — `{'on'} | 'off'`
`CovMetricSettings`	Selects coverage metrics for a 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 **Treat Simulink Logic blocks as short-circuited** and **Warn when unsupported blocks exist in model**, respectively, on the **Options** pane of the Coverage Settings dialog box. Option `'e'` is disabled by selecting **Display coverage results using model**	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'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
	coloring on the **Results** pane of the Coverage Settings dialog box.	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
`CovModelRefEnable`	If `CovModelRefEnable` is set to `on` or `all`, the Simulink Verification and Validation software generates coverage data for the entire model. Set by **Coverage for referenced models** on the **Coverage** pane of the Coverage Settings dialog box.	string — `'on' | {'off'} | 'all' | 'filtered'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`CovModelRefExcluded`	If `CovModelRefEnable` is set to `filtered`, this parameter stores a comma-separated list of referenced models for which coverage is disabled. Set by selecting **Coverage for referenced models** on the **Coverage** pane of the Coverage Settings dialog box and then clicking **Select Models**.	string — `{''}`
`CovNameIncrementing`	If `CovSaveSingleToWorkspaceVar` is set to `on`, setting `CovNameIncrementing` to `on` causes the Simulink Verification and Validation software to append numerals to the workspace variable names for results so that earlier results are not overwritten (for example, `covdata1`, `covdata2`, etc.) Set by **Increment variable name with each simulation** below the selected **Save last run in workspace variable** check box on the **Results** pane of the Coverage Settings dialog box.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`CovPath`	Model path of the subsystem for which the Simulink Verification and Validation software gathers and reports coverage data. Set by selecting **Coverage for this model: <model name>** on the **Coverage** pane of the Coverage Settings dialog box and then clicking **Select Subsystem**.	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 **Update results on pause** on the **Results** pane of the Coverage Settings dialog box.	string — `{'on'} | 'off'`
`CovSaveCumulativeTo-WorkspaceVar`	If set to `on`, the Simulink Verification and Validation software accumulates and saves the results of successive simulations in the workspace variable specified by `CovCumulativeVarName`. Set by **Save cumulative results in workspace variable** on the **Results** pane of the Coverage Settings dialog box.	string — `{'on'} | 'off'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`CovSaveName`	If `CovSaveSingleToWorkspace-Var` is set to `on`, the Simulink Verification and Validation software saves the results of the last simulation run in the workspace variable specified by this property. Set by **cvdata object name** below the selected **Save last run in workspace variable** check box on the **Results** pane of the Coverage Settings dialog box.	string — `{'covdata'}`
`CovSaveSingleTo-WorkspaceVar`	If set to `on`, the Simulink Verification and Validation software saves the results of the last simulation run in the workspace variable specified by `CovSaveName`. Set by **Save last run in workspace variable** on the **Results** pane of the Coverage Settings dialog box.	string — `{'on'} | 'off'`
`Created`	Date and time model was created. Set by **Created on** on the **History** pane of the Model Properties dialog box. See “Model History Controls” for more information.	string

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`Creator`	Name of model creator. Set by **Created by** on the **History** pane of the Model Properties dialog box. See "Model History Controls" for more information.	string
`CurrentBlock`	For internal use.	
`CurrentOutputPort`	For internal use.	
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`Decimation`	Specify that Simulink software output only every `N` points, where `N` is the specified decimation factor. Set by **Decimation** on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `{'1'}`
`DeleteChildFcn`	Delete child callback function. Created on the **Callbacks** pane of the Block Properties dialog box. See "Creating Block Callback Functions" for more information.	string — `{''}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`Description`	Description of this model. Set by **Model description** on the **Description** pane of the Model Properties dialog box.	string — `{''}`
`Dirty`	If the parameter is `on`, the model has unsaved changes.	string — `'on' | {'off'}`
`DiscreteInherit-ContinuousMsg`	Specifies diagnostic action to take when a Unit Delay block inherits a continuous sample time. Set by **Discrete used as continuous** on the **Sample Time Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`DisplayBdSearchResults`	For internal use.	
`DisplayBlockIO`	For internal use.	
`DisplayCallgraph-Dominators`	For internal use	
`DisplayCompileStats`	For internal use.	
`DisplayCondInputTree`	For internal use.	
`DisplayCondStIdTree`	For internal use.	
`DisplayErrorDirections`	For internal use.	
`DisplayInvisibleSources`	For internal use.	
`DisplaySortedLists`	For internal use.	
`DisplayVectorAnd-FunctionCounts`	For internal use.	
`DisplayVect-PropagationResults`	For internal use.	

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`Echo`	For internal use.	
`EnableOverflowDetection`	For internal use.	
`ExecutionContextIcon`	Toggles display of execution context icons on this model's block diagram. Set by **Execution Context Indicator** on the **Format > Block Displays** menu of the Model Editor.	string — `'on' | {'off'}`
`ExpressionFolding`	Enables expression folding. Set by **Eliminate superfluous local variables (Expression folding)** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`ExternalInput`	Names of MATLAB workspace variables used to designate data and times to be loaded from the workspace. Set by the **Input** field on the **Data Import/Export** pane of the Configuration Parameters dialog box.	scalar or vector — `{'[t, u]'}`
`ExtMode...`	Parameters whose names start with `ExtMode` apply to Simulink External Mode. See External Mode for more information.	

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ExtrapolationOrder`	Extrapolation order of the `ode14x` implicit fixed-step solver. Set by **Extrapolation order** on the **Solver** pane of the Configuration Parameters dialog box.	integer — `1 | 2 | 3 | {4}`
`FcnCallInpInside-ContextMsg`	Specifies diagnostic action to take when Simulink software must compute any function-call subsystem inputs directly or indirectly during execution of a call to a function-call subsystem. Set by **Context-dependent inputs** on the **Connectivity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'Use local settings'} | 'Enable All' | 'Disable All'`
`FileName`	For internal use.	
`FinalStateName`	Names of final states to save to the workspace after a simulation ends. Set by the **Final states** field on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `{'xFinal'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`FixedStep`	Fixed-step size. Set by **Fixed step size (fundamental sample time)** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'auto'}`
`FixptConstOverflowMsg`	Specifies diagnostic action to take when a fixed-point constant underflow occurs during simulation. Set by **Detect overflow** on the **Type Conversion Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`FixptConstPrecisionLossMsg`	Specifies diagnostic action to take when a fixed-point constant precision loss occurs during simulation. Set by **Detect precision loss** on the **Type Conversion Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`FixptConstUnderflowMsg`	Specifies diagnostic action to take when a fixed-point constant underflow occurs during simulation. Set by **Detect underflow** on the **Type Conversion Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`FixPtInfo`	For internal use.	
`FollowLinksWhen-OpeningFromGotoBlocks`	Specifies whether to search for Goto tags in libraries referenced by the model when opening the From block dialog box.	string — `'on' | {'off'}`
`ForceArrayBoundsChecking`	For internal use.	
`ForceConsistencyChecking`	For internal use.	
`ForceModelCoverage`	For internal use.	
`ForwardingTable`	Specifies the forwarding table for this library. See “Forwarding Tables” for more information.	string — `{{'old_path_1', 'new_path_1'} ... {'old_path_n', 'new_path_n'}}`
`ForwardingTableString`	For internal use.	

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`GeneratePreprocessorConditionals`	Globally controls whether generated code contains preprocessor conditionals. This parameter is relevant only to code generation, and has no effect on the behavior of a model in Simulink. The parameter is available only for an ERT target when **Inline parameters** is selected. Set by Configuration Parameters > Real-Time Workshop > Interface > **"Generate preprocessor conditionals"**. See "Generating Code Variants for Variant Models" for more information.	string — `{'Use local settings'} | 'off' | 'on'`
`GridSpacing`	Spacing of the Model Editor grid in pixels.	integer — `{20}`
`Handle`	Handle of the block diagram for this model.	double
`HiliteAncestors`	For internal use.	
`HiliteFcnCallInp-InsideContext`	Enables highlighting of Function-Call Subsystem blocks when one or more inputs depend on source blocks that appear in their own calling context.	string — `'on' | {'off'}`
`IgnoreBidirectionalLines`	For internal use.	

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`InheritedTsInSrcMsg`	Message behavior when the sample time is inherited. Set by **Source block specifies -1 sample time** on the **Sample Time Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`InitFcn`	Function that is called when this model is first compiled for simulation. Set by **Model initialization function** on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" for more information.	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 **Initial step size** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'auto'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`InlineParams`	Enable inline of parameters in generated code. Set by **Inline parameters** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `'on' | {'off'}`
`InsertRTBMode`	Control whether the Rate Transition block parameter **Ensure deterministic data transfer (maximum delay)** is set for auto-inserted Rate Transition blocks. Set by **Deterministic data transfer** on the **Solver** pane of the Configuration Parameters dialog box.	string — `'Always' | {'Whenever possible'} | 'Never (minimum delay)'`
`InspectSignalLogs`	Enable Simulink software to display logged signals in the MATLAB **Time Series Tools** viewer at the end of a simulation or whenever you pause the simulation. Set by **Inspect signal logs when simulation is paused/stopped** on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`Int32ToFloatConvMsg`	Specify message behavior when a 32-bit integer is converted to a single-precision float. Set by **32-bit integer to single precision float conversion** on the **Type Conversion Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none'` | `{'warning'}`
`IntegerOverflowMsg`	Specify message behavior when an integer overflow occurs. Set by **Detect overflow** in the **Signals** section on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none'` | `{'warning'}` | `'error'`
`InvalidFcnCallConnMsg`	Specify message behavior when an invalid function-call connection exists. Set by **Invalid function-call connection** on the **Connectivity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none'` | `'warning'` | `{'error'}`
`Jacobian`	For internal use.	
`LastModifiedBy`	User name of the person who last modified this model.	string
`LastModifiedDate`	Date used for version control.	string

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`LibraryLinkDisplay`	Shows which blocks in the model are linked or have disabled or modified links. Set by **Library Link Display** on the **Format** menu of the Model Editor.	string — `{'none'} | 'user' | 'all'`
`LibraryType`	For internal use.	
`LifeSpan`	Specify how long (in days) an application that contains blocks depending on elapsed or absolute time should be able to execute before timer overflow. Set by **Application lifespan (days)** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `{'inf'} |` any positive, nonzero scalar value
`LimitDataPoints`	Specify that the number of data points exported to the MATLAB workspace be limited to the number specified. Set by the **Limit data points to last** check box on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`LinearizationMsg`	For internal use.	
`Lines`	For internal use.	

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`LoadExternalInput`	Load input from workspace. Set by the **Input** check box on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `'on' | {'off'}`
`LoadInitialState`	Load initial state from	string — `'on' | {'off'}`
`Location`	For internal use.	
`Lock`	Lock or unlock a block library. Setting this parameter to `on` prevents a user from inadvertently changing a library.	string — `'on' | {'off'}`
`MaxConsecutiveMinStep`	Maximum number of minimum step size violations allowed during simulation. This option appears when the solver type is `Variable-step` and the solver is an ode one. Set by **Number of consecutive min steps** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'1'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`MaxConsecutiveZCs`	Maximum number of consecutive zero crossings allowed during simulation. This option appears when the solver type is `Variable-step` and the solver is an ode one. Set by **Number of consecutive zero crossings** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'1000'}`
`MaxConsecutiveZCsMsg`	Specifies diagnostic action to take when Simulink software detects the maximum number of consecutive zero crossings allowed. This option appears when the solver type is `Variable-step` and the solver is an ode one. Set by **Consecutive zero crossings violation** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | 'warning' | {'error'}`
`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'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`MaxMDLFileLineLength`	Controls the line lengths in the `.mdl` file. Use this to avoid line-wrapping, which can be important for source control tools. Specifies the maximum length in bytes, which may different from the number of characters in Japanese, and is different from the number of columns when tabs are present.	integer — `-1` (unlimited) or `>= 80`. Default is 120.
`MaxNumMinSteps`	Maximum number of times the solver uses the minimum step size.	string — `{'-1'}`
`MaxOrder`	Maximum order for `ode15s`. Set by **Maximum order** on the **Solver** pane of the Configuration Parameters dialog box.	string — `'1' | '2' | '3' | '4' | {'5'}`
`MaxStep`	Maximum step size. Set by **Max step size** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'auto'}`
`MdlSubVersion`	For internal use	

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`MergeDetectMultiDriving-BlocksExec`	Select the diagnostic action to take when the software detects a Merge block with more than one driving block executing at the same time step. Set by **Detect multiple driving blocks executing at the same time step** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`Metadata`	Names and attributes of arbitrary data associated with the model. To extract this metadata structure without needing to load the model, use the method `Simulink.MDLInfo.getMetadata`.	Structure. Fields can be strings, numeric matrices of type "double", or more structures.
`MinMaxOverflowArchiveData`	For internal use	
`MinMaxOverflowArchiveMode`	Logging type for fixed-point logging. Set by **Overwrite or merge results** in the Fixed-Point Tool.	string — `{'Overwrite'} | 'Merge'`
`MinMaxOverflowLogging`	Setting for fixed-point logging. Set by **Fixed-point instrumentation mode** in the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`MinStep`	Minimum step size for the solver. Set by **Min step size** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'auto'}`
`MinStepSizeMsg`	Message shown when minimum step size is violated. Set by **Min step size violation** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'warning'} | 'error'`
`ModelBrowserVisibility`	Show the Model Browser. Set by **Model Browser** on the **View > Model Browser Options** menu of the Model Editor.	string — `'on' | {'off'}`
`ModelBrowserWidth`	Width of the Model Browser pane in the model window. To display the Model Browser pane, see the `ModelBrowserVisibility` parameter.	integer — `{200}`
`ModelDataFile`	For internal use.	string — `{''}`
`ModelDependencies`	List of model dependencies. Set by **Model dependencies** on the **Model Referencing** pane of the Configuration Parameters dialog box.	string — `{''}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ModelReferenceCS-MismatchMessage`	Message shown when there is a model configuration mismatch. Set by **Model configuration mismatch** on the **Model Referencing Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`ModelReferenceData-LoggingMessage`	Message shown when there is unsupported data logging. Set by **Unsupported data logging** on the **Model Referencing Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`ModelReferenceExtra-NoncontSigs`	Specifies diagnostic action to take when a discrete signal appears to pass through a Model block to the input of a block with continuous states. Set by **Extraneous discrete derivative signals** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | 'warning' | {'error'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ModelReferenceIO-MismatchMessage`	Message shown when there is a port and parameter mismatch. Set by **Port and parameter mismatch** on the **Model Referencing Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`ModelReferenceIOMsg`	Message shown when there is an invalid root Inport or Outport block connection. Set by **Invalid root Inport/Outport block connection** on the **Model Referencing Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`ModelReferenceMin-AlgLoopOccurrences`	Toggles the minimization of algebraic loop occurrences. Set by **Minimize algebraic loop occurrences** on the **Model Referencing** pane of the Configuration Parameters dialog box.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ModelReferenceNum-InstancesAllowed`	Total number of model reference instances allowed per top model. Set by **Total number of instances allowed per top model** on the **Model Referencing** pane of the Configuration Parameters dialog box.	string — `'Zero' | 'Single' | {'Multi'}`
`ModelReferencePass-RootInputsByReference`	Toggles the passing of scalar root inputs by value. Set by **Pass scalar root inputs by value** on the **Model Referencing** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`ModelReferenceSim-TargetVerbose`	This parameter is deprecated and has no effect. Use `AccelVerboseBuild` instead.	
`ModelReferenceSymbol-NameMessage`	For internal use.	
`ModelReferenceTargetType`	For internal use.	
`ModelReferenceVersion-MismatchMessage`	Message shown when there is a model block version mismatch. Set by **Model block version mismatch** on the **Model Referencing Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`ModelVersion`	Version number of model.	string — `{'1.1'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ModelVersionFormat`	Format of model's version number. Set by **Model version** on the **History** pane of the Model Properties dialog box. See "Model History Controls" for more information.	string — `{'1.%<AutoIncrement:0>'}`
`ModelWorkspace`	References this model's model workspace object.	an instance of the `Simulink.ModelWorkspace` class
`ModifiedBy`	Last person to modify this model.	string
`ModifiedByFormat`	Format for the display of last modifier. Set by **Last saved by** on the **History** pane of the Model Properties dialog box. See "Model History Controls" for more information. Can also be set by **Last saved by** on the **Model history** field on the **History** pane of the Model Explorer.	string — `{'%<Auto>'}`
`ModifiedComment`	Field for user comments.	string — `{''}`
`ModifiedDate`	Date of last model modification.	string

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ModifiedDateFormat`	Format of modified date. Set by **Last saved on** on the **History** pane of the Model Properties dialog box. See "Model History Controls" for more information.	string — `{'%<Auto>'}`
`ModifiedHistory`	Area for keeping notes about the history of the model. Set by the **Model history** field on the **History** pane of the Model Properties dialog box. See "Model History Controls" for more information. Can also be set by the **Model history** field on the **History** pane of the Model Explorer.	string — `{''}`
`MultiTaskCondExecSysMsg`	Select the diagnostic action to take if Simulink software detects a subsystem that might cause data corruption or nondeterministic behavior. Set by **Multitask conditionally executed subsystem** on the **Sample Time Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | 'warning' | {'error'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`MultiTaskDSMMsg`	Specifies diagnostic action to take when one task reads data from a Data Store Memory block to which another task writes data. Set by **Multitask data store** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | 'warning' | {'error'}`
`MultiTaskRateTransMsg`	Specifies diagnostic action to take when an invalid rate transition takes place between two blocks operating in multitasking mode. Set by **Multitask rate transition** on the **Sample Time Diagnostics** pane of the Configuration Parameters dialog box.	string — `'warning' | {'error'}`
`Name`	Model name.	string
`NumberNewtonIterations`	Number of Newton's method iterations performed by the `ode14x` implicit fixed-step solver. Set by **Number Newton's iterations** on the **Solver** pane of the Configuration Parameters dialog box.	integer — `{1}`
`ObjectParameters`	Names and attributes of model parameters.	structure
`Open`	For internal use.	

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`OptimizeBlockIOStorage`	Enables signal storage reuse optimization. Set by **Signal storage reuse** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`OutputOption`	Time step output options for variable-step solvers. Set by **Output options** on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `'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.	string — `{'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 **Output times** on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `{'[]'}`
`PaperOrientation`	Printing paper orientation.	string — `'portrait' | {'landscape'} | 'rotated'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`PaperPosition`	When `PaperPositionMode` is set to `manual`, this parameter determines the position and size of a diagram on paper and the size of the diagram exported as a graphic file in the units specified by `PaperUnits`.	vector — `[left, bottom, width, height]`
`PaperPositionMode`	Paper position mode. • `auto` When printing, Simulink software sizes the diagram to fit the printed page. When exporting a diagram as a graphic image, Simulink software sizes the exported image to be the same size as the diagram's normal size on screen. • `manual` When printing, Simulink software positions and sizes the diagram on the page as indicated by `PaperPosition`. When exporting a diagram as a graphic image, Simulink software sizes the exported graphic to have the height and width specified by `PaperPosition`. • `tiled` Enables tiled printing.	string — `{'auto'} | 'manual' | 'tiled'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
	See "Tiled Printing" for more information.	
`PaperSize`	Size of `PaperType` in `PaperUnits`.	vector — `[width height]` (read only)
`PaperType`	Printing paper type.	string — `'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.	string — `'normalized' | {'inches'} | 'centimeters' | 'points'`
`ParameterArgumentNames`	List of parameters used as arguments when this model is called as a reference. Set by **Model arguments (for referencing this model)** in the **Model Workspace** pane of the Model Explorer.	string — `{''}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ParameterDowncastMsg`	Specifies diagnostic action to take when a parameter downcast occurs during simulation. Set by **Detect downcast** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | 'warning' | {'error'}`
`ParameterOverflowMsg`	Specifies diagnostic action to take when a parameter overflow occurs during simulation. Set by **Detect overflow** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | 'warning' | {'error'}`
`ParameterPrecisionLossMsg`	Specifies diagnostic action to take when parameter precision loss occurs during simulation. Set by **Detect precision loss** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ParameterTunabilityLossMsg`	Specifies diagnostic action to take when a parameter cannot be tuned because it uses unsupported functions or operators. Set by **Detect loss of tunability** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`ParameterUnderflowMsg`	Specifies diagnostic action to take when a parameter underflow occurs during simulation. Set by **Detect underflow** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'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 — `{''}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`PositivePriorityOrder`	Choose the appropriate priority ordering for the real-time system targeted by this model. The Real-Time Workshop software uses this information to implement asynchronous data transfers. Set by **Higher priority value indicates higher task priority** on the **Solver** pane of the Configuration Parameters dialog box.	string — `'on' | {'off'}`
`PostLoadFcn`	Function invoked just after this model is loaded. Set by **Model post-load function** on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" for more information.	string — `{''}`
`PostSaveFcn`	Function invoked just after this model is saved to disk. Set by **Model post-save function** on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" for more information.	string — `{''}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`PreLoadFcn`	Preload callback. Set by **Model pre-load function** on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" for more information.	string — `{''}`
`PreSaveFcn`	Function invoked just before this model is saved to disk. Set by **Model pre-save function** on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" for more information.	string — `{''}`
`ProdBitPerChar`	Describes the length in bits of the C `char` data type supported by the production hardware to be used by this model. Set by **char** in the **Embedded hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{8}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ProdBitPerInt`	Describes the length in bits of the C `int` data type supported by the production hardware to be used by this model. Set by **int** in the **Embedded hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{32}`
`ProdBitPerLong`	Describes the length in bits of the C `long` data type supported by the production hardware to be used by this model. Set by **long** in the **Embedded hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{32}`
`ProdBitPerShort`	Describes the length in bits of the C `short` data type supported by the production hardware to be used by this model. Set by **short** in the **Embedded hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{16}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ProdEndianess`	Describes the significance of the first byte of a data word of the production hardware to be used by this model. Set by **Byte ordering** in the **Embedded hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	string — `{'Unspecified'} | 'LittleEndian' | 'BigEndian'`
`ProdEqTarget`	Specifies that the hardware used to test the code generated from this model is the same as the production hardware or has the same characteristics. Set by **None** in the **Emulation hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`ProdHWDeviceType`	Predefined hardware device to specify the C language constraints for your microprocessor. Set by **Device type** in the **Embedded hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	string — `{'32-bit Generic'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ProdIntDivRoundTo`	Describes how the C compiler that creates production code for this model rounds the result of dividing one signed integer by another to produce a signed integer quotient. Set by **Signed integer division rounds to** in the **Embedded hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	string — `'Floor' | 'Zero' | {'Undefined'}`
`ProdShiftRightIntArith`	Describes whether the C compiler that creates production code for this model implements a signed integer right shift as an arithmetic right shift. Set by **Shift right on a signed integer as arithmetic shift** in the **Embedded hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ProdWordSize`	Describes the word length in bits of the production hardware to be used by this model. Set by **"Number of bits: native word size"** in the **Embedded hardware** section on the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{32}`
`Profile`	Enables the simulation profiler for this model. Set by **Profiler** on the **Tools** menu of the Model Editor.	string — `'on' | {'off'}`
`ReadBeforeWriteMsg`	Specifies diagnostic action to take when the model attempts to read data from a data store before it has stored data at the current time step. Set by **Detect read before write** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'UseLocalSettings'} | 'DisableAll' | 'EnableAllAsWarning' | 'EnableAllAsError'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values

`RecordCoverage`	A value of `on` causes Simulink software 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`
	`CovModelRefEnable`
	`CovModelRefExcluded`
	`CovNameIncrementing`
	`CovPath`
	`CovReportOnPause`
	`CovSaveCumulativeToWork-SpaceVar`
	`CovSaveName`
	`CovSaveSingleToWorkspace-Var`
	If the value is `off`, no model coverage data is collected or reported and the preceding coverage report parameters have no effect.
		string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
	Set by **Coverage for this model: <model name>** on the **Coverage** pane of the Coverage Settings dialog box.	
`Refine`	Refine factor. Set by **Refine factor** on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `{'1'}`
`RelTol`	Relative error tolerance. Set by **Relative tolerance** 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'}`
`ReqHilite`	Highlights all the blocks in the Simulink diagram that have requirements associated with them. Set by **Highlight model** on the **Tools > Requirements** menu of the Model Editor.	string — `'on' | {'off'}`
`RequirementInfo`	For internal use.	

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`RootOutportRequire-BusObject`	Specifies diagnostic action to take when a bus enters a root model Outport block for which a bus object has not been specified. Set by **Unspecified bus object at root Outport block** on the **Connectivity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`RTPrefix`	Specifies diagnostic action to take when Simulink software encounters an object name that begins with `rt`. Set by **"rt" prefix for identifiers** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | 'warning' | {'error'}`
`RTW...`	See the Real-Time Workshop documentation for more information on parameters whose names begin with `RTW`.	
`SampleTimeAnnotations`	Set by **Annotations** on the **Format > Sample Time Display** menu of the Model Editor.	string — `'on' | {'off'}`
`SampleTimeColors`	Set by **Colors** on the **Format > Sample Time Display** menu of the Model Editor.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SampleTimeConstraint`	This option appears when the solver type is `Fixed-step`. Set by **Periodic sample time constraint** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'Unconstrained'} | 'STIndependent' | 'Specified'`
`SampleTimeProperty`	Specifies and assigns priorities to the sample times implemented by the model. This option appears when **Periodic sample time constraint** is set to `Specified`. Set by **Sample time properties** on the **Solver** pane of the Configuration Parameters dialog box.	Structure containing the fields `SampleTime`, `Offset`, and `Priority`
`SavedCharacterEncoding`	Specifies the character set used to encode this model. See the `slCharacterEncoding` command for more information.	string
`SaveDefaultBlockParams`	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.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SaveFormat`	Format used to save data to the MATLAB workspace. Set by **Format** on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `{'Array'} | 'Structure' | 'StructureWithTime'`
`SaveOutput`	Save simulation output to workspace. Set by the **Output** check box on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `{'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.	string — `'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.	string — `{'on'} | 'off'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SaveWithDisabledLinksMsg`	Specifies diagnostic action to take when saving a block diagram having disabled library links. Set by **Block diagram contains disabled library links** on the **Saving Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none'` | `{'warning'}` | `'error'`
`SaveWithParameterized-LinksMsg`	Specifies diagnostic action to take when saving a block diagram having parameterized library links. Set by **Block diagram contains parameterized library links** on the **Saving Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none'` | `{'warning'}` | `'error'`
`ScreenColor`	Background color of the model window. Set by **Screen Color** on the **Format** menu of the Model Editor.	string — `'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 in Alphabetical Order (Continued)

Parameter	Description	Values
`SFcnCompatibilityMsg`	See `SfunCompatibilityCheckMsg` parameter for more information.	
`SFSimEcho`	Enables output to appear in the MATLAB Command Window during simulation of a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Echo expressions without semicolons** on the **Simulation Target** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`SFSimEnableDebug`	Enables debugging and animation during simulation of a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Enable debugging/animation** on the **Simulation Target** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SFSimOverflowDetection`	Enables overflow detection of data during simulation of a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Overflow occurs for data when a value assigned to it exceeds the numeric capacity of the data type. **Note** To enable this option, you must also select the **Data Range** check box in the Stateflow Debugger window. Set by **Enable overflow detection (with debugging)** on the **Simulation Target** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`SfunCompatibilityCheckMsg`	Specifies diagnostic action to take when S-function upgrades are needed. Set by **S-function upgrades needed** on the **Compatibility Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ShapePreserveControl`	At each time step, use derivative information to improve integration accuracy. Set by **Shape preservation** on the **Solver** pane of the Configuration Parameters dialog box.	string — `'EnableAll' | {'DisableAll'}`
`ShowGrid`	Show the Model Editor grid.	string — `'on' | {'off'}`
`ShowLinearization-Annotations`	Toggles linearization icons in the model.	string — `{'on'} | 'off'`
`ShowLineDimensions`	Show signal dimensions on this model's block diagram. Set by **Signal Dimensions** on the **Format > Port/Signal Displays** menu of the Model Editor.	string — `'on' | {'off'}`
`ShowLineDimensionsOnError`	For internal use.	
`ShowLineWidths`	Deprecated. Use `ShowLineDimensions` instead.	
`ShowLoopsOnError`	Highlight invalid loops graphically.	string — `{'on'} | 'off'`
`ShowModelReferenceBlockIO`	Toggles display of I/O mismatch on block. Set by **Model Block I/O Mismatch** on the **Format > Block Displays** menu of the Model Editor.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ShowModelReference-BlockVersion`	Toggles display of version on block. Set by **Model Block Version** on the **Format > Block Displays** menu of the Model Editor.	string — `'on' | {'off'}`
`Shown`	For internal use.	
`ShowPageBoundaries`	Toggles display of page boundaries on the Model Editor's canvas. Set by **Show Page Boundaries** on the **View** menu of the Model Editor.	string — `'on' | {'off'}`
`ShowPortDataTypes`	Show data types of ports on this model's block diagram. Set by **Port Data Types** on the **Format > Port/Signal Displays** menu of the Model Editor.	string — `'on' | {'off'}`
`ShowPortDataTypesOnError`	For internal use.	
`ShowStorageClass`	Show storage classes of signals on this model's block diagram. Set by **Storage Class** on the **Format > Port/Signal Displays** menu of the Model Editor.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ShowTestPointIcons`	Show test point icons on this model's block diagram. Set by **Testpoint/Logging Indicators** on the **Format > Port/Signal Displays** menu of the Model Editor.	string — `{'on'} | 'off'`
`ShowViewerIcons`	Show viewer icons on this model's block diagram. Set by **Viewer Indicators** on the **Format > Port/Signal Displays** menu of the Model Editor.	string — `{'on'} | 'off'`
`SignalInfNanChecking`	Specifies diagnostic action to take when the value of a block output is `Inf` or `NaN` at the current time step. Set by **Inf or NaN block output** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`SignalLabelMismatchMsg`	Specifies diagnostic action to take when a signal label mismatch occurs. Set by **Signal label mismatch** on the **Connectivity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`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.	string — `{'on'} | 'off'`
`SignalLoggingName`	Name for saving signal logging data to a workspace. Set by the **Signal logging** field on the **Data Import/Export** pane of the Configuration Parameters dialog box.	string — `{'logsout'}`
`SignalRangeChecking`	Select the diagnostic action to take when signals exceed specified minimum or maximum values. Set by **Simulation range checking** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`SignalResolutionControl`	Control which named states and signals get resolved to Simulink signal objects. Set by **Signal resolution** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'UseLocalSettings'} | 'TryResolveAll' | 'TryResolveAll-WithWarning'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SigSpecEnsureSample-TimeMsg`	Specifies diagnostic action to take when the sample time of the source port of a signal specified by a Signal Specification block differs from the signal's destination port. Set by **Enforce sample times specified by Signal Specification blocks** on the **Sample Time Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`SimBlas`	Enables Embedded MATLAB Function blocks in Simulink models and Embedded MATLAB functions in Stateflow charts to speed up low-level matrix operations during simulation. Set by **Use BLAS library for faster simulation** on the **Simulation Target** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SimBuildMode`	Specifies how you build the simulation target for a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Simulation target build mode** on the **Simulation Target** pane of the Configuration Parameters dialog box.	string — `{'sf_incremental_build'} | 'sf_nonincremental_build' | 'sf_make' | 'sf_make_clean' | 'sf_make_clean_objects'`
`SimCompilerOptimization`	Specifies the compiler optimization level during acceleration code generation. Set by **Compiler optimization level** on the **Optimization** pane of the Configuration Parameters dialog box.	string — `'on' | {'off'}`
`SimCtrlC`	Enables responsiveness checks in code generated for Embedded MATLAB function blocks. Set by **"Ensure responsiveness"** on the **Simulation Target** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SimCustomHeaderCode`	Enter code lines to appear near the top of a generated header file for a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Header file** on the **Simulation Target > Custom Code** pane of the Configuration Parameters dialog box.	string — `{''}`
`SimCustomInitializer`	Enter code statements that execute once at the start of simulation for a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Initialize function** on the **Simulation Target > Custom Code** pane of the Configuration Parameters dialog box.	string — `{''}`
`SimCustomSourceCode`	Enter code lines to appear near the top of a generated source code file for a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Source file** on the **Simulation Target > Custom Code** pane of the Configuration Parameters dialog box.	string — `{''}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SimCustomTerminator`	Enter code statements that execute at the end of simulation for a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Terminate function** on the **Simulation Target > Custom Code** pane of the Configuration Parameters dialog box.	string — `{''}`
`SimIntegrity`	Detects violations of memory integrity in code generated for Embedded MATLAB function blocks and stops execution with a diagnostic. Set by **"Ensure memory integrity"** on the **Simulation Target** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`SimReservedNameArray`	Enter the names of variables or functions in the generated code that match the names of variables or functions specified in custom code. This action prevents naming conflicts between identifiers in the generated code and in custom code for a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks.	string array — `{{}}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
	Set by **Reserved names** on the **Simulation Target > Symbols** pane of the Configuration Parameters dialog box.	
`SimulationCommand`	Executes a simulation command. **Note** You cannot use `set_param` to run a simulation in a MATLAB session that does not have a display, i.e., if you used `matlab -nodisplay` to start the session.	string — `'start' | 'stop' | 'pause' | 'continue' | 'step' | 'update' | 'WriteDataLogs' | 'SimParamDialog' | 'connect' | 'disconnect' | 'WriteExtModeParamVect' | 'AccelBuild'`
`SimulationMode`	Indicates whether Simulink software should run in Normal, Accelerator, Rapid Accelerator, or External mode. Set by the **Simulation** menu on the Model Editor.	string — `{'normal'} | 'accelerator' | 'rapid' | 'external'`
`SimulationStatus`	Indicates simulation status.	string — `{'stopped'} | 'updating' | 'initializing' | 'running' | 'paused' | 'terminating' | 'external'`
`SimulationTime`	Current time value for the simulation.	double — `{0}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SimUserIncludeDirs`	Enter a space-separated list of directory paths that contain files you include in the compiled target for a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Include directories** on the **Simulation Target > Custom Code** pane of the Configuration Parameters dialog box.	string — `{''}` **Note** If your list includes any Windows® path strings that contain spaces, each instance must be enclosed in double quotes within the argument string, for example, `'C:\Project "C:\Custom Files"'`
`SimUserLibraries`	Enter a space-separated list of static libraries that contain custom object code to link into the target for a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Libraries** on the **Simulation Target > Custom Code** pane of the Configuration Parameters dialog box.	string — `{''}` **Note** If your list includes any Windows file name strings that contain spaces, each instance must be enclosed in double quotes within the argument string, for example, `'mathutil.lib "op utils.lib"'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SimUserSources`	Enter a space-separated list of source files to compile and link into the target for a model that contains Embedded MATLAB Function blocks, Stateflow charts, or Truth Table blocks. Set by **Source files** on the **Simulation Target > Custom Code** pane of the Configuration Parameters dialog box.	string — `{''}` **Note** If your list includes any Windows file name strings that contain spaces, each instance must be enclosed in double quotes within the argument string, for example, `'algs.c "div alg.c"'`
`SingleTaskRateTransMsg`	Specifies diagnostic action to take when a rate transition takes place between two blocks operating in single-tasking mode. Set by **Single task rate transition** on the **Sample Time Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`Solver`	Solver used for the simulation. Set by the **Solver** drop-down list on the **Solver** pane of the Configuration Parameters dialog box.	string — `'VariableStepDiscrete' | {'ode45'} | 'ode23' | 'ode113' | 'ode15s' | 'ode23s' | 'ode23t' | 'ode23tb' | 'FixedStepDiscrete' | 'ode5' | 'ode4' | 'ode3' | 'ode2' | 'ode1' | 'ode14x'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SolverMode`	Solver mode for this model. This option appears when the solver type is `Fixed-step`. Set by **Tasking mode for periodic sample times** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'Auto'} | 'SingleTasking' | 'MultiTasking'`
`SolverName`	Solver used for the simulation. See `Solver` parameter for more information.	
`SolverPrmCheckMsg`	Enables diagnostics to control when Simulink software automatically selects solver parameters. This option notifies you if: • Simulink software changes a user-modified parameter to make it consistent with other model settings • Simulink software automatically selects solver parameters for the model, such as **FixedStepSize** Set by **Automatic solver parameter selection** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`SolverResetMethod`	This option appears when the solver type is `Variable-step` and the solver is `ode15s (stiff/NDF)`, `ode23t (Mod. stiff/Trapezoidal)`, or `ode23tb (stiff/TR-BDF2)`. Set by **Solver reset method** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'Fast'} | 'Robust'`
`SolverType`	Solver type used for the simulation. Set by **Type** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'Variable-step'} | 'Fixed-step'`
`SortedOrder`	Show the sorted order of this model's blocks on the block diagram. Set by **Sorted Order** on the **Format > Block Displays** menu of the Model Editor.	string — `'on' | {'off'}`
`StartFcn`	Start simulation callback. Set by **Simulation start function** on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" for more information.	string — `{''}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`StartTime`	Simulation start time. Set by **Start time** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'0.0'}`
`StateNameClashWarn`	Select the diagnostic action to take when a name is used for more than one state in the model. Set by **State name clash** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'}`
`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.	string — `{'xout'}`
`StatusBar`	Show or hide the status bar on the Model Editor window. Set by **Status Bar** on the **View** menu of the Model Editor.	string — `{'on'} | 'off'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`StopFcn`	Stop simulation callback. Set by **Simulation stop function** on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" for more information.	string — `{''}`
`StopTime`	Simulation stop time. Set by **Stop time** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'10.0'}`
`StrictBusMsg`	Specifies diagnostic action to take when Simulink software detects a signal that some blocks treat as a mux or vector, while other blocks treat the signal as a bus. Set by **Mux blocks used to create bus signals** and **Bus signal treated as vector** on the **Connectivity Diagnostics** pane of the Configuration Parameters dialog box. See "Avoiding Mux/Bus Mixtures" for more information.	string — `{'None'} | 'Warning' | 'ErrorLevel1' | 'WarnOnBusTreatedAsVector'| 'ErrorOnBusTreatedAsVector'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`Tag`	User-specified text that is assigned to the model's `Tag` parameter and saved with the model.	string — `{''}`
`TargetBitPerChar`	Describes the length in bits of the C `char` data type supported by the hardware used to test generated code. Set by **char** in the **Emulation hardware** section of the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{8}`
`TargetBitPerInt`	Describes the length in bits of the C `int` data type supported by the hardware used to test generated code. Set by **int** in the **Emulation hardware** section of the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{32}`
`TargetBitPerLong`	Describes the length in bits of the C `long` data type supported by the hardware used to test generated code. Set by **long** in the **Emulation hardware** section of the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{32}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`TargetBitPerShort`	Describes the length in bits of the C `short` data type supported by the hardware used to test generated code. Set by **short** in the **Emulation hardware** section of the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{16}`
`TargetEndianess`	Describes the significance of the first byte of a data word of the hardware used to test generated code. Set by **Byte ordering** in the **Emulation hardware** section of the **Hardware Implementation** pane of the Configuration Parameters dialog box.	string — `{'Unspecified'} | 'LittleEndian' | 'BigEndian'`
`TargetFcnLib`	For internal use.	
`TargetHWDeviceType`	Describes the characteristics of the hardware used to test generated code. Set by **Device type** in the **Emulation hardware** section of the **Hardware Implementation** pane of the Configuration Parameters dialog box.	string — `{'32-bit Generic'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`TargetIntDivRoundTo`	Describes how the C compiler that creates test code for this model rounds the result of dividing one signed integer by another to produce a signed integer quotient. Set by **Signed integer division rounds to** in the **Emulation hardware** section of the **Hardware Implementation** pane of the Configuration Parameters dialog box.	string — `'Floor' | 'Zero' | {'Undefined'}`
`TargetShiftRightIntArith`	Describes whether the C compiler that creates test code for this model implements a signed integer right shift as an arithmetic right shift. Set by **Shift right on a signed integer as arithmetic shift** in the **Emulation hardware** section of the **Hardware Implementation** pane of the Configuration Parameters dialog box.	string — `{'on'} | 'off'`
`TargetTypeEmulation WarnSuppressLevel`	Specifies whether Real-Time Workshop software displays or suppresses warning messages when emulating integer sizes in rapid prototyping environments.	integer — `{0}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`TargetWordSize`	Describes the word length in bits of the hardware used to test generated code. Set by **native word size** in the **Emulation hardware** section of the **Hardware Implementation** pane of the Configuration Parameters dialog box.	integer — `{32}`
`TasksWithSamePriorityMsg`	Specifies diagnostic action to take when tasks have equal priority. Set by **Tasks with equal priority** on the **Sample Time Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`TiledPageScale`	Scales the size of the tiled page relative to the model.	string — `{'1'}`
`TiledPaperMargins`	Controls the size of the margins associated with each tiled page. Each element in the vector represents a margin at the particular edge.	vector — `[left, top, right, bottom]`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`TimeAdjustmentMsg`	Specifies diagnostic action to take if Simulink software makes a minor adjustment to a sample hit time while running the model. Set by **Sample hit time adjusting** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning'`
`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 or hide the toolbar on the Model Editor window. Set by **Toolbar** on the **View** menu of the Model Editor.	string — `{'on'} | 'off'`
`TryForcingSFcnDF`	This flag is used for backward compatibility with user S-functions that were written prior to R12.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`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 — `{''}`
`Type`	Simulink object type (read only).	string — `{'block_diagram'}`
`UnconnectedInputMsg`	Unconnected input ports diagnostic. Set by **Unconnected block input ports** on the **Connectivity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`UnconnectedLineMsg`	Unconnected lines diagnostic. Set by **Unconnected line** on the **Connectivity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`UnconnectedOutputMsg`	Unconnected block output ports diagnostic. Set by **Unconnected block output ports** on the **Connectivity Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`UnderSpecifiedDataTypeMsg`	Detect usage of heuristics to assign signal data types. Set by **Underspecified data types** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`UnderspecifiedInitial-izationDetection`	Select how Simulink software handles initialization of initial conditions for conditionally executed subsystems, Merge blocks, subsystem elapsed time, and Discrete-Time Integrator blocks. Set by **Underspecified initialization detection** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'classic'} | 'simplified'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`UniqueDataStoreMsg`	Specifies diagnostic action to take when the model contains multiple Data Store Memory blocks that specify the same data store name. Set by **Duplicate data store names** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`UnknownTsInhSupMsg`	Detect blocks that have not set whether they allow the model containing them to inherit a sample time. Set by **Unspecified inheritability of sample time** on the **Solver Diagnostics** pane of the Configuration Parameters dialog box.	string — `'none' | {'warning'} | 'error'`
`UnnecessaryDatatype-ConvMsg`	Detect unnecessary data type conversion blocks. Set by **Unnecessary type conversions** on the **Type Conversion Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`UpdateHistory`	Specifies when to prompt the user about updating the model history. Set by **Prompt to update model history** on the **History** pane of the Model Properties dialog box or **Prompt to update model history** on the **History** pane of the Model Explorer. See "Model History Controls" for more information.	string — `{'UpdateHistoryNever'} | 'UpdateHistoryWhenSave'`
`UpdateModelReference-Targets`	Specify whether to rebuild simulation and Real-Time Workshop targets for referenced models before updating, simulating, or generating code for this model. Set by **Rebuild options** on the **Model Referencing** pane of the Configuration Parameters dialog box.	string — `'IfOutOfDate' | 'Force' | 'AssumeUpToDate' | {'IfOutOfDateOrStructuralChange'}`
`UseAnalysisPorts`	For internal use.	
`UseIntDivNetSlope`	Use integer division to handle net slopes that are reciprocals of integers.	string — `'on' | {'off'}`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`VectorMatrixConversionMsg`	Detect vector-to-matrix or matrix-to-vector conversions. Set by **Vector/matrix block input conversion** on the **Type Conversion Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'none'} | 'warning' | 'error'`
`Version`	Simulink version used to modify the model (read only).	release version number
`WideLines`	Draws lines that carry vector or matrix signals wider than lines that carry scalar signals. Set by **Wide Nonscalar Lines** on the **Format > Port/Signal Displays** menu of the Model Editor.	string — `'on' | {'off'}`
`WideVectorLines`	Deprecated. Use `WideLines` instead.	
`WriteAfterReadMsg`	Specifies diagnostic action to take when the model attempts to store data in a data store after previously reading data from it in the current time step. Set by **Detect write after read** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'UseLocalSettings'} | 'DisableAll' | 'EnableAllAsWarning' | 'EnableAllAsError'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`WriteAfterWriteMsg`	Specifies diagnostic action to take when the model attempts to store data in a data store twice in succession in the current time step. Set by **Detect write after write** on the **Data Validity Diagnostics** pane of the Configuration Parameters dialog box.	string — `{'UseLocalSettings'} | 'DisableAll' | 'EnableAllAsWarning' | 'EnableAllAsError'`
`ZCThreshold`	Specifies the deadband region used during the detection of zero crossings. Signals falling within this region are defined as having crossed through zero. Set by **Signal threshold** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'auto'} |` any real number greater than or equal to zero
`ZeroCross`	For internal use.	
`ZeroCrossAlgorithm`	Specifies the algorithm to detect zero crossings when you select a variable-step solver. Set by **Algorithm** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'Nonadaptive'} | 'Adaptive'`

Model Parameters in Alphabetical Order (Continued)

Parameter	Description	Values
`ZeroCrossControl`	Enable zero-crossing detection. Set by **Zero-crossing control** on the **Solver** pane of the Configuration Parameters dialog box.	string — `{'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 **View** menu of the Model Editor.	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

In this section...

About Common Block Parameters

This table lists the parameters common to all Simulink blocks, including block callback parameters (see "Using Callback Functions"). Examples of commands that change these parameters follow this table (see "Examples of Setting Block Parameters" on page 8-99).

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
`AttributesFormatString`	String format specified for block annotations in the Block Parameters 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

Common Block Parameters (Continued)

Parameter	Description	Values
`ClipboardFcn`	Function called when block is copied to the clipboard (**Ctrl+C**)	string
`CloseFcn`	Function called when `close_system` is run on block.	string
`CompiledPort-ComplexSignals`	Complexity of port signals after updating diagram.	structure array
`CompiledPortDataTypes`	Data types of port signals after updating diagram.	structure array
`CompiledPortDimensions`	Dimensions of port signals after updating diagram.	structure array
`CompiledPortFrameData`	Frame mode of port signals after updating diagram.	structure array
`CompiledPortWidths`	Structure of port widths after updating diagram.	structure array
`CompiledSampleTime`	Block sample time after updating diagram.	vector [`sample time`, `offset time`]
`CopyFcn`	Function called when block is copied.	string
`DataTypeOverrideCompiled`	For internal use.	

Common Block Parameters (Continued)

Parameter	Description	Values
`DeleteFcn`	Function called when block is deleted. If a block is graphically deleted, you can still undo the operation and call the block's `UndoDeleteFcn`. In addition, for graphically deleted blocks, the block's `DestroyFcn` is still called when the model is closed or any subsystem containing the block is destroyed using `delete_block`.	MATLAB expression
`DestroyFcn`	Function called when block is destroyed. If you run the `delete_block` command for a block, it first calls the block's`DeleteFcn`, then calls the `DestroyFcn` for that block; no undo is possible. The `DestroyFcn` is also called when you close the model or invoke `delete_block` on a subsystem containing the block.	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`	For internal use.	
`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
`ExtModeLoggingSupported`	Enable a block to support uploading of signal data in external mode (for example, with a scope block).	`{'off'} | 'on'`
`ExtModeLoggingTrig`	Enable a block to act as the trigger block for external mode signal uploading.	`{'off'} | 'on'`
`ExtModeUploadOption`	Enable a block to upload signal data in external mode when the **Select all** check box on the External Signal & Triggering dialog box is not selected. A value of `log` indicates the block uploads signals. A value of `none` indicates the block does not upload signals. The value `monitor` is currently not in use. If the **Select all** check box on the External Signal & Triggering dialog box is selected, it overrides this parameter setting.	`{'none'} | 'log' | 'monitor'`
`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'}`

Common Block Parameters (Continued)

Parameter	Description	Values
`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
`HiliteAncestors`	For internal use.	
`InitFcn`	Initialization function for a masked block. Created on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" in the Using Simulink documentation for further information.	MATLAB expression
`InputSignalNames`	Names of input signals.	cell array
`IOSignalStrings`		list
`IOType`	Signal & Scope Manager type.	`{'none'} | 'viewer' | 'siggen'`
`LineHandles`	Handles of lines connected to block.	struct
`LinkStatus`	Link status of block. Updates out-of-date reference blocks when queried using `get_param`.	`{'none'} | 'resolved' | 'unresolved' | 'implicit' | 'inactive' | 'restore' | 'propagate'`
`LoadFcn`	Function called when block is loaded.	MATLAB expression
`MinMaxOverflow-Logging_Compiled`	For internal use.	

Common Block Parameters (Continued)

Parameter	Description	Values
`ModelCloseFcn`	Function called when model is closed. The `ModelCloseFcn` is called prior to the block's `DeleteFcn` and `DestroyFcn` callbacks, if either are set.	MATLAB expression
`ModelParamTableInfo`	For internal use.	
`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 Parameters dialog box opens.	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. The `ParentCloseFcn` of blocks at the root model level is not called when the model is closed.	MATLAB expression

Common Block Parameters (Continued)

Parameter	Description	Values
`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 number of the port for data ports - `'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.	structure array

Common Block Parameters (Continued)

Parameter	Description	Values
	• `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` 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.	structure array

Common Block Parameters (Continued)

Parameter	Description	Values
	• `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` 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.	

Common Block Parameters (Continued)

Parameter	Description	Values
`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 of coordinates (in pixels) not enclosed in quotation marks: `[left top right bottom]` **Note** The origin is located in the upper left corner of the model window. The maximum value for a coordinate is 32767.

Common Block Parameters (Continued)

Parameter	Description	Values
`PostSaveFcn`	Function called after the block is saved. Created on the **Callbacks** pane of the Model Properties dialog box. See "Creating Model Callback Functions" in the Using Simulink documentation for further information.	MATLAB expression
`PreCopyFcn`	Function called before the block is copied. See "Block Callback Parameters" in the Using Simulink documentation for details.	MATLAB expression
`PreDeleteFcn`	Function called before the block is deleted. See "Block Callback Parameters" in the Using Simulink documentation for details.	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 to which this block links.	string `{''}`
`RequirementInfo`	For internal use.	

Common Block Parameters (Continued)

Parameter	Description	Values
`RTWData`	User specified data, used by Real-Time Workshop software.	
`SampleTime`	Value of the sample time parameter.	string
`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`	State perturbation size to use during linearization. See "Linearizing Individual Blocks Using Block Perturbation" in the Simulink Control Design documentation for details.	string
`StaticLinkStatus`	Link status of block. Does not update out-of-date reference blocks when queried using `get_param`.	`{'none'} | 'resolved' | 'unresolved' | 'implicit' | 'inactive' | 'restore' | 'propagate'`
`StopFcn`	Function called at the termination of a simulation.	MATLAB expression
`Tag`	Text that appears in the block label that Simulink software 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

Common Block Parameters (Continued)

Parameter	Description	Values
`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 work as masked subsystems. The tables indicate masked blocks by adding the designation "masked subsystem" after the block type.

Note The type listed for nonmasked blocks is the value of the `BlockType` parameter (see "Common Block Parameters" on page 8-87). The type listed for masked blocks is the value of the `MaskType` parameter (see "Mask Parameters" on page 8-232).

The **Dialog Box Prompt** column indicates the text of the prompt for the parameter on the block 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
- Discontinuities Library Block Parameters
- Discrete Library Block Parameters
- Logic and Bit Operations Library Block Parameters
- Lookup Tables Block Parameters
- Math Operations Library Block Parameters
- Model Verification Library Block Parameters
- Model-Wide Utilities Library Block Parameters
- Ports & Subsystems Library Block Parameters
- Signal Attributes Library Block Parameters
- Signal Routing Library Block Parameters
- Sinks Library Block Parameters
- Sources Library Block Parameters
- User-Defined Functions Library Block Parameters

- Additional Discrete Block Library Parameters
- Additional Math: Increment - Decrement Block Parameters

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	string — `{'none'} | 'rising' | 'falling' | 'either' | 'level' | 'level hold'`
`InitialConditionSource`	Initial condition source	string — `{'internal'} | 'external'`
`InitialCondition`	Initial condition	scalar or vector — `{'0'}`
`LimitOutput`	Limit output	string — `{'off'} | 'on'`
`UpperSaturationLimit`	Upper saturation limit	scalar or vector — `{'inf'}`
`LowerSaturationLimit`	Lower saturation limit	scalar or vector — `{'-inf'}`
`ShowSaturationPort`	Show saturation port	string — `{'off'} | 'on'`
`ShowStatePort`	Show state port	string — `{'off'} | 'on'`
`AbsoluteTolerance`	Absolute tolerance	scalar — `{'auto'}`
`IgnoreLimit`	Ignore limit and reset when linearizing	string — `{'off'} | 'on'`
`ZeroCross`	Enable zero–crossing detection	string — `'off' | {'on'}`
`ContinuousStateAttributes`	State Name	string — `{''}` | variable
State-Space (`StateSpace`)		
`A`	A	matrix — `{'1'}`
`B`	B	matrix — `{'1'}`

Continuous Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
C	C	matrix — {'1'}
D	D	matrix — {'1'}
X0	Initial conditions	vector — {'0'}
AbsoluteTolerance	Absolute tolerance	scalar — {'auto'}
ContinuousStateAttributes	State Name	string — {''} | variable
Transfer Fcn (TransferFcn)		
Numerator	Numerator coefficients	vector or matrix — {'[1]'}
Denominator	Denominator coefficients	vector — {'[1 1]'}
AbsoluteTolerance	Absolute tolerance	scalar — {'auto'}
ContinuousStateAttributes	State Name	string — {''} | variable
Transport Delay (TransportDelay)		
DelayTime	Time delay	scalar or vector — {'1'}
InitialOutput	Initial output	scalar or vector — {'0'}
BufferSize	Initial buffer size	scalar — {'1024'}
FixedBuffer	Use fixed buffer size	string — {'off'} | 'on'
TransDelayFeedthrough	Direct feedthrough of input during linearization	string — {'off'} | 'on'
PadeOrder	Pade order (for linearization)	string — {'0'}
Variable Time Delay (VariableTimeDelay)		
VariableDelayType	Select delay type	string — 'Variable transport delay' | {'Variable time delay'}
MaximumDelay	Maximum delay	scalar or vector — {'10'}
InitialOutput	Initial output	scalar or vector — {'0'}
MaximumPoints	Initial buffer size	scalar — {'1024'}

Continuous Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`FixedBuffer`	Use fixed buffer size	string — `{'off'} | 'on'`
`ZeroDelay`	Handle zero delay	string — `{'off'} | 'on'`
`TransDelayFeedthrough`	Direct feedthrough of input during linearization	string — `{'off'} | 'on'`
`PadeOrder`	Pade order (for linearization)	string — `{'0'}`
`ContinuousStateAttributes`	State Name	string — `{''}` | variable
Variable Transport Delay (`VariableTransportDelay`)		
`VariableDelayType`	Select delay type	string — `{'Variable transport delay'} | 'Variable time delay'`
`MaximumDelay`	Maximum delay	scalar or vector — `{'10'}`
`InitialOutput`	Initial output	scalar or vector — `{'0'}`
`MaximumPoints`	Initial buffer size	scalar — `{'1024'}`
`FixedBuffer`	Use fixed buffer size	string — `{'off'} | 'on'`
`TransDelayFeedthrough`	Direct feedthrough of input during linearization	string — `{'off'} | 'on'`
`PadeOrder`	Pade order (for linearization)	string — `{'0'}`
`AbsoluteTolerance`	Absolute tolerance	scalar — `{'auto'}`
`ContinuousStateAttributes`	State Name	string — `{''}` | variable
Zero-Pole (`ZeroPole`)		
`Zeros`	Zeros	vector — `{'[1]'}`
`Poles`	Poles	vector — `{'[0 -1]'}`
`Gain`	Gain	vector — `{'[1]'}`
`AbsoluteTolerance`	Absolute tolerance	scalar — `{'auto'}`
`ContinuousStateAttributes`	State Name	string — `{''}` | variable

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	string — `'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'}`
`SaturateOnInteger Overflow`	Saturate on integer overflow	string — `'off' | {'on'}`
`LinearizeAsGain`	Treat as gain when linearizing	string — `'off' | {'on'}`
`ZeroCross`	Enable zero-crossing detection	string — `'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	string — `'rising' | 'falling' | {'either'}`
`ShowOutputPort`	Show output port	string — `'off' | {'on'}`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`

Discontinuities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Quantizer (`Quantizer`)		
`QuantizationInterval`	Quantization interval	scalar or vector — `{'0.5'}`
`LinearizeAsGain`	Treat as gain when linearizing	string — `'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	string — `'continuous' | {'inherited'}`
`InitialCondition`	Initial condition	string — `{'0'}`
`LinearizeAsGain`	Treat as gain when linearizing	string — `'off' | {'on'}`
Rate Limiter Dynamic (`Rate Limiter Dynamic`) (masked subsystem)		
Relay (`Relay`)		
`OnSwitchValue`	Switch on point	string — `{'eps'}`
`OffSwitchValue`	Switch off point	string — `{'eps'}`
`OnOutputValue`	Output when on	string — `{'1'}`
`OffOutputValue`	Output when off	string — `{'0'}`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`

Discontinuities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutDataTypeStr	Output data type	string — 'Inherit: Inherit via back propagation' | {'Inherit: All ports same datatype'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16)' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
LockScale	Lock output data type setting against changes by the fixed-point tools	string — {'off'} | 'on'
Saturation (Saturate)		
UpperLimit	Upper limit	scalar or vector — {'0.5'}
LowerLimit	Lower limit	scalar or vector — {'-0.5'}
LinearizeAsGain	Treat as gain when linearizing	string — 'off' | {'on'}
ZeroCross	Enable zero-crossing detection	string — 'off' | {'on'}
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
OutMin	Output minimum	string — {'[]'}
OutMax	Output maximum	string — {'[]'}

Discontinuities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataTypeStr`	Output data type	string — `'Inherit: Inherit via back propagation' | {'Inherit: Same as input'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
Saturation Dynamic (`Saturation Dynamic`) (masked subsystem)		
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Same as second input'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`OutputDataTypeScaling Mode`	Deprecated	
`OutDataType`	Deprecated	
`OutScaling`	Deprecated	

Discontinuities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
LockScale	Lock output data type setting against changes by the fixed-point tools	string — {'off'} | 'on'
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
DoSatur	Saturate on integer overflow	string — {'off'} | 'on'
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'}
OutMin	Output minimum	string — {'[]'}
OutMax	Output maximum	string — {'[]'}
OutDataTypeStr	Output data type	string — {'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutputDataTypeScaling Mode`	Deprecated	
`OutDataType`	Deprecated	
`OutScaling`	Deprecated	
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`DoSatur`	Saturate to max or min when overflows occur	string — `{'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'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutputDataTypeScaling Mode`	Deprecated	
`OutDataType`	Deprecated	
`OutScaling`	Deprecated	
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`DoSatur`	Saturate to max or min when overflows occur	string — `{'off'} | 'on'`
Discrete FIR Filter (`Discrete FIR Filter`)		
`CoefSource`	Coefficient source	string — `{'Dialog parameters'} | 'Input port'`
`FirFiltStruct`	Filter structure	string — `{'Direct form'} | 'Direct form symmetric' | 'Direct form antisymmetric' | 'Direct form transposed' | 'Lattice MA'` **Note** You must have a Signal Processing Blockset license to use a filter structure other than `Direct form`.
`NumCoeffs`	Numerator coefficients	vector — `{'[0.5 0.5]'}`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`IC`	Initial states	scalar or vector — `{'0'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`CoefMin`	Coefficient minimum	string — `{'[]'}`
`CoefMax`	Coefficient maximum	string — `{'[]'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`CoefDataTypeStr`	Coefficient data type	string — `{'Inherit: Same word length as input'} | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16)' | 'fixdt(1,16,0)'`
`ProductDataTypeStr`	Product output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)'`
`AccumDataTypeStr`	Accumulator data type	string — `'Inherit: Same as input' | {'Inherit: Same as product output'} | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)'`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataTypeStr`	Output data type	string — `'Inherit: Same as input' | {'Inherit: Same as accumulator'} | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)'`
`LockScale`	Lock data type settings against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Discrete Filter (`DiscreteFilter`)		
`Numerator`	Numerator coefficients	vector — `{'[1]'}`
`Denominator`	Denominator coefficients	vector — `{'[1 0.5]'}`
`IC`	Initial states	string — `{'0'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'1'}`
`a0EqualsOne`	Optimize by skipping divide by leading denominator coefficient (a0)	string — `{'off'} | 'on'`
`NumCoefMin`	Numerator coefficient minimum	string — `{'[]'}`
`NumCoefMax`	Numerator coefficient maximum	string — `{'[]'}`
`DenCoefMin`	Denominator coefficient minimum	string — `{'[]'}`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`DenCoefMax`	Denominator coefficient maximum	string — `{'[]'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`StateDataTypeStr`	State data type	string — `{'Inherit: Same as input'} | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`NumCoefDataTypeStr`	Numerator coefficient data type	string — `{'Inherit: Inherit via internal rule'} | 'int8' | 'int16' | 'int32' | 'fixdt(1,16)' | 'fixdt(1,16,0)'`
`DenCoefDataTypeStr`	Denominator coefficient data type	string — `{'Inherit: Inherit via internal rule'} | 'int8' | 'int16' | 'int32' | 'fixdt(1,16)' | 'fixdt(1,16,0)'`
`NumProductDataTypeStr`	Numerator product output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`DenProductDataTypeStr`	Denominator product output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`NumAccumDataTypeStr`	Numerator accumulator data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'Inherit: Same as product output' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`DenAccumDataTypeStr`	Denominator accumulator data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'Inherit: Same as product output' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`LockScale`	Lock data type settings against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
`StateIdentifier`	State name	string — `{''}`
`StateMustResolveTo SignalObject`	State name must resolve to Simulink signal object	string — `{'off'} | 'on'`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWStateStorageClass	Real-Time Workshop storage class	string — {'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageType Qualifier	Real-Time Workshop storage type qualifier	string — {''}
Discrete State-Space (DiscreteStateSpace)		
A	A	matrix — {'1'}
B	B	matrix — {'1'}
C	C	matrix — {'1'}
D	D	matrix — {'1'}
X0	Initial conditions	vector — {'0'}
SampleTime	Sample time (-1 for inherited)	string — {'1'}
StateIdentifier	State name	string — {''}
StateMustResolveTo SignalObject	State name must resolve to Simulink signal object	string — {'off'} | 'on'
RTWStateStorageClass	Real-Time Workshop storage class	string — {'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageType Qualifier	Real-Time Workshop storage type qualifier	string — {''}
Discrete Transfer Fcn (DiscreteTransferFcn)		
Numerator	Numerator coefficients	vector — {'[1]'}
Denominator	Denominator coefficients	vector — {'[1 0.5]'}
InitialStates	Initial states	string — {'0'}
SampleTime	Sample time (-1 for inherited)	string — {'1'}

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`aOEqualsOne`	Optimize by skipping divide by leading denominator coefficient (a0)	string — `{'off'} | 'on'`
`NumCoefMin`	Numerator coefficient minimum	string — `{'[]'}`
`NumCoefMax`	Numerator coefficient maximum	string — `{'[]'}`
`DenCoefMin`	Denominator coefficient minimum	string — `{'[]'}`
`DenCoefMax`	Denominator coefficient maximum	string — `{'[]'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`StateDataTypeStr`	State data type	string — `{'Inherit: Same as input'} | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`NumCoefDataTypeStr`	Numerator coefficient data type	string — `{'Inherit: Inherit via internal rule'} | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`DenCoefDataTypeStr`	Denominator coefficient data type	string — `{'Inherit: Inherit via internal rule'} | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`NumProductDataTypeStr`	Numerator product output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`DenProductDataTypeStr`	Denominator product output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`NumAccumDataTypeStr`	Numerator accumulator data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'Inherit: Same as product output' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`DenAccumDataTypeStr`	Denominator accumulator data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'Inherit: Same as product output' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'int8' | 'int16' | 'int32' | 'fixdt(1,16,0)'`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
LockScale	Lock data type settings against changes by the fixed-point tools	string — {'off'} | 'on'
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
SaturateOnIntegerOverflow	Saturate on integer overflow	string — {'off'} | 'on'
StateIdentifier	State name	string — {''}
StateMustResolveToSignalObject	State name must resolve to Simulink signal object	string — {'off'} | 'on'
RTWStateStorageClass	Real-Time Workshop storage class	string — {'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'
RTWStateStorageTypeQualifier	Real-Time Workshop storage type qualifier	string — {''}
Discrete Zero-Pole (DiscreteZeroPole)		
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 — {''}
StateMustResolveToSignalObject	State name must resolve to Simulink signal object	string — {'off'} | 'on'
RTWStateStorageClass	Real-Time Workshop storage class	string — {'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`RTWStateStorageType Qualifier`	Real-Time Workshop storage type qualifier	string — `{''}`
Discrete-Time Integrator (`DiscreteIntegrator`)		
`IntegratorMethod`	Integrator method	string — `{'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	string — `{'none'} | 'rising' | 'falling' | 'either' | 'level' | 'sampled level'`
`InitialConditionSource`	Initial condition source	string — `{'internal'} | 'external'`
`InitialCondition`	Initial condition	scalar or vector — `{'0'}`
`InitialConditionMode`	Use initial condition as initial and reset value for	string — `'State only (most efficient)' | {'State and output'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'1'}`
`LimitOutput`	Limit output	string — `{'off'} | 'on'`
`UpperSaturationLimit`	Upper saturation limit	scalar or vector — `{'inf'}`
`LowerSaturationLimit`	Lower saturation limit	scalar or vector — `{'-inf'}`
`ShowSaturationPort`	Show saturation port	string — `{'off'} | 'on'`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ShowStatePort	Show state port	string — {'off'} | 'on'
IgnoreLimit	Ignore limit and reset when linearizing	string — {'off'} | 'on'
OutMin	Output minimum	string — {'[]'}
OutMax	Output maximum	string — {'[]'}
OutDataTypeStr	Output data type	string — {'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
LockScale	Lock output data type setting against changes by the fixed-point tools	string — {'off'} | 'on'
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
SaturateOnIntegerOverflow	Saturate on integer overflow	string — {'off'} | 'on'
StateIdentifier	State name	string — {''}
StateMustResolveToSignalObject	State name must resolve to Simulink signal object	string — {'off'} | 'on'

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`RTWStateStorageClass`	Real-Time Workshop storage class	string — `{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'`
`RTWStateStorageType Qualifier`	Real-Time Workshop 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	string — `{'off'} | 'on'`
`LinearizeMemory`	Direct feedthrough of input during linearization	string — `{'off'} | 'on'`
`LinearizeAsDelay`	Treat as a unit delay when linearizing with discrete sample time	string — `{'off'} | 'on'`
`StateIdentifier`	State name	string — `{''}`
`StateMustResolveTo SignalObject`	State name must resolve to Simulink signal object	string — `{'off'} | 'on'`
`RTWStateStorageClass`	Real-Time Workshop storage class	string — `{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'`
`RTWStateStorageType Qualifier`	Real-Time Workshop storage type qualifier	string — `{''}`

Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
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	string — `{'Oldest'} | 'Newest'`
`includeCurrent`	Include current input in output vector	string — `{'off'} | 'on'`
Transfer Fcn First Order (`First Order Transfer Fcn`) (masked subsystem)		
`PoleZ`	Pole (in Z plane)	string — `{'0.95'}`
`ICPrevOutput`	Initial condition for previous output	string — `{'0.0'}`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`DoSatur`	Saturate to max or min when overflows occur	string — `{'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`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`DoSatur`	Saturate to max or min when overflows occur	string — `{'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`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`DoSatur`	Saturate to max or min when overflows occur	string — `{'off'} | 'on'`
Unit Delay (`UnitDelay`)		
`X0`	Initial conditions	scalar or vector — `{'0'}`
`SampleTime`	Sample time	string — `{'-1'}`
`StateIdentifier`	State name	string — `{''}`
`StateMustResolveTo SignalObject`	State name must resolve to Simulink signal object	string — `{'off'} | 'on'`
`RTWStateStorageClass`	Real-Time Workshop storage class	string — `{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'`
`RTWStateStorageType Qualifier`	Real-Time Workshop storage type qualifier	string — `{''}`

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	string — `{'AND'} | 'OR' | 'NAND' | 'NOR' | 'XOR' | 'NOT'`
`UseBitMask`	Use bit mask ...	string — `'off' | {'on'}`
`NumInputPorts`	Number of input ports	string — `{'1'}`
`BitMask`	Bit Mask	string — `{'bin2dec('11011001')'}`
`BitMaskRealWorld`	Treat mask as	string — `'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)		

Logic and Bit Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`relop`	Operator	string — `'==' | '~=' | '<' | {'<='} | '>=' | '>'`
`const`	Constant value	string — `{'3.0'}`
`LogicOutDataTypeMode`	Output data type mode	string — `{'uint8'} | 'boolean'`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
Compare To Zero (`Compare To Zero`) (masked subsystem)		
`relop`	Operator	string — `'==' | '~=' | '<' | {'<='} | '>=' | '>'`
`LogicOutDataTypeMode`	Output data type mode	string — `{'uint8'} | 'boolean'`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
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'}`

Logic and Bit Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Extract Bits (`Extract Bits`) (masked subsystem)		
`bitsToExtract`	Bits to extract	string — `{'Upper half'} | 'Lower half' | 'Range starting with most significant bit' | 'Range ending with least significant bit' | 'Range of bits'`
`numBits`	Number of bits	string — `{'8'}`
`bitIdxRange`	Bit indices ([start end], 0-based relative to LSB)	string — `{'[0 7]'}`
`outScalingMode`	Output scaling mode	string — `{'Preserve fixed-point scaling'} | 'Treat bit field as an integer'`
Interval Test (`Interval Test`) (masked subsystem)		
`IntervalClosedRight`	Interval closed on right	string — `'off' | {'on'}`
`uplimit`	Upper limit	string — `{'0.5'}`
`IntervalClosedLeft`	Interval closed on left	string — `'off' | {'on'}`
`lowlimit`	Lower limit	string — `{'-0.5'}`
`LogicOutDataTypeMode`	Output data type mode	string — `'uint8' | {'boolean'}`
Interval Test Dynamic (`Interval Test Dynamic`) (masked subsystem)		
`IntervalClosedRight`	Interval closed on right	string — `'off' | {'on'}`
`IntervalClosedLeft`	Interval closed on left	string — `'off' | {'on'}`
`LogicOutDataTypeMode`	Output data type mode	string — `'uint8' | {'boolean'}`
Logical Operator (`Logic`)		

Logic and Bit Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`Operator`	Operator	string — `{'AND'} | 'OR' | 'NAND' | 'NOR' | 'XOR' | 'NXOR' | 'NOT'`
`Inputs`	Number of input ports	string — `{'2'}`
`IconShape`	Icon shape	string — `{'rectangular'} | 'distinctive'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`AllPortsSameDT`	Require all inputs and output to have the same data type	string — `{'off'} | 'on'`
`OutDataTypeStr`	Output data type	string — `'Inherit: Logical (see Configuration Parameters: Optimization)' | {'boolean'} | 'fixdt(1,16)'`
Relational Operator (`RelationalOperator`)		
`Operator`	Relational operator	string — `'==' | '~=' | '<' | {'<='} | '>=' | '>' | 'isInf' | 'isNaN' | 'isFinite'`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InputSameDT`	Require all inputs to have the same data type	string — `{'off'} | 'on'`
`OutDataTypeStr`	Output data type	string — `'Inherit: Logical (see Configuration Parameters: Optimization)' | {'boolean'} | 'fixdt(1,16)'`

Logic and Bit Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Shift Arithmetic (`Shift Arithmetic`) (masked subsystem)		
`nBitShiftRight`	Number of bits to shift right (use negative value to shift left)	string — `{'8'}`
`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)		
`Formula`	Output formula	string — `'sin(2*pi*u)' | {'cos(2*pi*u)'} | 'exp(j*2*pi*u)' | 'sin(2*pi*u) and cos(2*pi*u)'`
`NumDataPoints`	Number of data points for lookup table	string — `{'(2^5)+1'}`
`OutputWordLength`	Output word length	string — `{'16'}`
Direct Lookup Table (n-D) (`LookupNDDirect`)		
`NumberOfTableDimensions`	Number of table dimensions	string — `'1' | {'2'} | '3' | '4'`
`InputsSelectThisObjectFromTable`	Inputs select this object from table	string — `{'Element'} | 'Column' | '2-D Matrix'`
`TableIsInput`	Make table an input	string — `{'off'} | 'on'`
`Table`	Table data	string — `{'[4 5 6;16 19 20;10 18 23]'}`
`ActionForOutOfRangeInput`	Action for out-of-range input	string — `'None' | {'Warning'} | 'Error'`

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`SampleTime`	Sample time	string — `{'-1'}`
`maskTabDims`	Deprecated	
`explicitNumDims`	Deprecated	
`outDims`	Deprecated	
`tabIsInput`	Deprecated	
`mxTable`	Deprecated	
`clipFlag`	Deprecated	
`samptime`	Deprecated	
Interpolation Using Prelookup (`Interpolation_n-D`)		
`NumberOfTableDimensions`	Number of table dimensions	string — `'1' | {'2'} | '3' | '4'`
`Table`	Table data	string — `{'sqrt([1:11]' * [1:11])'}`
`InterpMethod`	Interpolation method	string — `'None - Flat' | {'Linear'}`
`ExtrapMethod`	Extrapolation method	string — `'None - Clip' | {'Linear'}`
`RangeErrorMode`	Action for out-of-range input	string — `{'None'} | 'Warning' | 'Error'`
`CheckIndexInCode`	Check index in generated code	string — `'off' | {'on'}`
`ValidIndexMayReachLast`	Valid index input may reach last index	string — `{'off'} | 'on'`
`NumSelectionDims`	Number of sub-table selection dimensions	string — `{'0'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`OutMin`	Output minimum	string — `{'[]'}`

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutMax	Output maximum	string — {'[]'}
OutDataTypeStr	Output data type	string — 'Inherit: Inherit via back propagation' | {'Inherit: Inherit from table data'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
LockScale	Lock data type settings against changes by the fixed-point tools	string — {'off'} | 'on'
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
SaturateOnIntegerOverflow	Saturate on integer overflow	string — {'off'} | 'on'
TableMin	Table minimum	string — {'[]'}
TableMax	Table maximum	string — {'[]'}
TableDataTypeStr	Table data type	string — 'Inherit: Inherit from 'Table data'' | {'Inherit: Same as output'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16)' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`IntermediateResultsDataTypeStr`	Intermediate results data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as output' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
Lookup Table (`Lookup`)		
`InputValues`	Vector of input values	vector — `{'[-5:5]'}`
`Table`	Table data	vector — `{'tanh([-5:5])'}`
`LookUpMeth`	Lookup method	string — `{'Interpolation-Extrapolation'} | 'Interpolation-Use End Values' | 'Use Input Nearest' | 'Use Input Below' | 'Use Input Above'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `'Inherit: Inherit via back propagation' | {'Inherit: Same as input'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
SaturateOnInteger Overflow	Saturate on integer overflow	string — {'off'} | 'on'
Lookup Table (2-D) (Lookup2D)		
RowIndex	Row index input values	string — {'[1:3]'}
ColumnIndex	Column index input values	string — {'[1:3]'}
Table	Table data	string — {'[4 5 6;16 19 20;10 18 23]'}
LookUpMeth	Lookup method	string — {'Interpolation-Extrapolation'} | 'Interpolation-Use End Values' | 'Use Input Nearest' | 'Use Input Below' | 'Use Input Above'
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
OutMin	Output minimum	string — {'[]'}
OutMax	Output maximum	string — {'[]'}
InputSameDT	Require all inputs to have the same data type	string — {'off'} | 'on'
OutDataTypeStr	Output data type	string — 'Inherit: Inherit via back propagation' | {'Inherit: Same as first input'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Lookup Table (n-D) (`Lookup_n-D`))		
`NumberOfTableDimensions`	Number of table dimensions	string — `'1' | {'2'} | '3' | '4'`
`Table`	Table	string — `{'[4 5 6;16 19 20;10 18 23]'}`
`BreakpointsForDimension1`	BP 1	string — `{'[10,22,31]'}`
`BreakpointsForDimension2`	BP 2	string — `{'[10,22,31]'}`
`BreakpointsForDimension3`	BP 3	string — `{'[1:3]'}`
`...`	`...`	`...`
`BreakpointsForDimension30`	BP 30	string — `{'[1:3]'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InterpMethod`	Interpolation method	string — `'None - Flat' | {'Linear'} | 'Cubic spline'`
`ExtrapMethod`	Extrapolation method	string — `'None - Clip' | {'Linear'} | 'Cubic spline'`
`UseLastTableValue`	Use last table value for inputs at or above last breakpoint	string — `{'off'} | 'on'`

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
IndexSearchMethod	Index search method	string — 'Evenly spaced points' | 'Linear search' | {'Binary search'}
BeginIndexSearchUsing PreviousIndexResult	Begin index search using previous index result	string — {'off'} | 'on'
UseOneInputPortForAll InputData	Use one input port for all input data	string — {'off'} | 'on'
ActionForOutOfRangeInput	Action for out-of-range input	string — {'None'} | 'Warning' | 'Error'
TableDataTypeStr	Table data	string — 'Inherit: Inherit from 'Table data'' | {'Inherit: Same as output'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16)' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
TableMin	Table data minimum	string — {'[]'}
TableMax	Table data maximum	string — {'[]'}
BreakpointsForDimension1 DataTypeStr	Breakpoints 1	string — {'Inherit: Same as corresponding input'} | 'Inherit: Inherit from 'Breakpoint data'' | 'double' | 'single' |
BreakpointsForDimension1 Min	Breakpoints 1 minimum	string — {'[]'}
BreakpointsForDimension1 Max	Breakpoints 1 maximum	string — {'[]'}

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
BreakpointsForDimension2 DataTypeStr	Breakpoints 2	string — {'Inherit: Same as corresponding input'} | 'Inherit: Inherit from 'Breakpoint data'' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16)' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
BreakpointsForDimension2 Min	Breakpoints 2 minimum	string — {'[]'}
BreakpointsForDimension2 Max	Breakpoints 2 maximum	string — {'[]'}
...	...	...
BreakpointsForDimension30 DataTypeStr	Breakpoints 30	string — {'Inherit: Same as corresponding input'} | 'Inherit: Inherit from 'Breakpoint data'' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16)' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
BreakpointsForDimension30 Min	Breakpoints 30 minimum	string — {'[]'}
BreakpointsForDimension30 Max	Breakpoints 30 maximum	string — {'[]'}
FractionDataTypeStr	Fraction	string — {'Inherit: Inherit via internal rule'} | 'double' | 'single' | 'fixdt(1,16,0)'

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`IntermediateResultsDataTypeStr`	Intermediate results	string — `'Inherit: Inherit via internal rule' | {'Inherit: Same as output'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`OutDataTypeStr`	Output	string — `'Inherit: Inherit via back propagation' | 'Inherit: Inherit from table data' | {'Inherit: Same as first input'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`InputSameDT`	Require all inputs to have the same data type	string — `'off' | {'on'}`
`LockScale`	Lock data type settings against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
`ProcessOutOfRangeInput`	Deprecated	
Lookup Table Dynamic (`Lookup Table Dynamic`) (masked subsystem)		

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
LookUpMeth	Lookup Method	string — 'Interpolation-Extrapolation' | {'Interpolation-Use End Values'} | 'Use Input Nearest' | 'Use Input Below' | 'Use Input Above'
OutDataTypeStr	Output data type	string — {'fixdt('double')'} | 'Inherit: Inherit via back propagation' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
OutputDataTypeScaling Mode	Deprecated	
OutDataType	Deprecated	
OutScaling	Deprecated	
LockScale	Lock output data type setting against changes by the fixed-point tools	string — {'off'} | 'on'
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round'| 'Simplest' | 'Zero'
DoSatur	Saturate to max or min when overflows occur	string — {'off'} | 'on'
Prelookup (PreLookup)		
BreakpointsData	Breakpoint data	string — {'[10:10:110]'}
IndexSearchMethod	Index search method	string — 'Evenly spaced points' | 'Linear search' | {'Binary search'}
BeginIndexSearchUsing PreviousIndexResult	Begin index search using previous index result	string — {'off'} | 'on'

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutputOnlyTheIndex`	Output only the index	string — `{'off'} | 'on'`
`ProcessOutOfRangeInput`	Process out-of-range input	string — `'Clip to range' | {'Linear extrapolation'}`
`UseLastBreakpoint`	Use last breakpoint for input at or above upper limit	string — `{'off'} | 'on'`
`ActionForOutOfRangeInput`	Action for out-of-range input	string — `{'None'} | 'Warning' | 'Error'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`IndexDataTypeStr`	Index data type	string — `'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | {'uint32'} | 'fixdt(1,16)'`
`FractionDataTypeStr`	Fraction data type	string — `{'Inherit: Inherit via internal rule'} | 'double' | 'single' | 'fixdt(1,16,0)'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`BreakpointMin`	Breakpoint minimum	string — `{'[]'}`
`BreakpointMax`	Breakpoint maximum	string — `{'[]'}`

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`BreakpointDataTypeStr`	Breakpoint data type	string — `{'Inherit: Same as input'} | 'Inherit: Inherit from 'Breakpoint data'' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16)' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
Sine (`Sine`) (masked subsystem)		
`Formula`	Output formula	string — `{'sin(2*pi*u)'} | 'cos(2*pi*u)' | 'exp(j*2*pi*u)' | 'sin(2*pi*u) and cos(2*pi*u)'`
`NumDataPoints`	Number of data points for lookup table	string — `{'(2^5)+1'}`
`OutputWordLength`	Output word length	string — `{'16'}`

Math Operations Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Abs (`Abs`)		
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`OutMax`	Output maximum	string — `{'[]'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutDataTypeStr	Output data type	string — 'Inherit: Inherit via internal rule' | 'Inherit: Inherit via back propagation' | {'Inherit: Same as input'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
LockScale	Lock output data type setting against changes by the fixed-point tools	string — {'off'} | 'on'
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
SaturateOnIntegerOverflow	Saturate on integer overflow	string — {'off'} | 'on'
Add (Sum)		
IconShape	Icon shape	string — {'rectangular'} | 'round'
Inputs	List of signs	string — {'++'}
CollapseMode	Sum over	string — {'All dimensions'} | 'Specified dimension'
CollapseDim	Dimension	string — {'1'}
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
InputSameDT	Require all inputs to have the same data type	string — {'off'} | 'on'

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`AccumDataTypeStr`	Accumulator data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as first input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'Inherit: Same as first input' | 'Inherit: Same as accumulator' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock data type settings against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SaturateOnInteger Overflow	Saturate on integer overflow	string — {'off'} | 'on'
Algebraic Constraint (Algebraic Constraint) (masked subsystem)		
z0	Initial guess	string — {'0'}
Assignment (Assignment)		
NumberOfDimensions	Number of output dimensions	string — {'1'}
IndexMode	Index mode	string — 'Zero-based' | {'One-based'}
OutputInitialize	Initialize output (Y)	string — {'Initialize using input port <Y0>'} | 'Specify size for each dimension in table'
IndexOptionArray	Index Option	string — 'Assign all' | {'Index vector (dialog)'} | 'Index vector (port)' | 'Starting index (dialog)' | 'Starting index (port)'
IndexParamArray	Index	cell array
OutputSizeArray	Output Size	cell array
DiagnosticForDimensions	Action if any output element is not assigned	string — 'Error' | 'Warning' | {'None'}
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
IndexOptions	See IndexOptionArray parameter for more information.	
Indices	See IndexParamArray parameter for more information.	

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutputSizes`	See `OutputSizeArray` parameter for more information.	
Bias (`Bias`)		
`Bias`	Bias	string — `{'0.0'}`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Complex to Magnitude-Angle (`ComplexToMagnitudeAngle`)		
`Output`	Output	string — `'Magnitude' | 'Angle' | {'Magnitude and angle'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
Complex to Real-Imag (`ComplexToRealImag`)		
`Output`	Output	string — `'Real' | 'Imag' | {'Real and imag'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
Divide (`Product`)		
`Inputs`	Number of inputs	string — `{'*/'}`
`Multiplication`	Multiplication	string — `{'Element-wise(.*)'} | 'Matrix(*)'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InputSameDT`	Require all inputs to have same data type	string — `{'off'} | 'on'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'Inherit: Same as first input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Dot Product (`Dot Product`)		
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InputSameDT`	Require all inputs to have same data type	string — `'off' | {'on'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'Inherit: Same as first input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`OutputDataTypeScalingMode`	Deprecated	
`OutDataType`	Deprecated	
`OutScaling`	Deprecated	
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Gain (`Gain`)		
`Gain`	Gain	string — `{'1'}`
`Multiplication`	Multiplication	string — `{'Element-wise(K.*u)'} | 'Matrix(K*u)' | 'Matrix(u*K)' | 'Matrix(K*u) (u vector)'`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
OutMin	Output minimum	string — {'[]'}
OutMax	Output maximum	string — {'[]'}
OutDataTypeStr	Output data type	string — {'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'Inherit: Same as input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
LockScale	Lock output data type setting against changes by the fixed-point tools	string — {'off'} | 'on'
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
SaturateOnIntegerOverflow	Saturate on integer overflow	string — {'off'} | 'on'
ParamMin	Parameter minimum	string — {'[]'}
ParamMax	Parameter maximum	string — {'[]'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`ParamDataTypeStr`	Parameter data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16)' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
Magnitude-Angle to Complex (`MagnitudeAngleToComplex`)		
`Input`	Input	string — `'Magnitude' | 'Angle' | {'Magnitude and angle'}`
`ConstantPart`	Magnitude or Angle	string — `{'0'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
Math Function (`Math`)		
`Operator`	Function	string — `{'exp'} | 'log' | '10^u' | 'log10' | 'magnitude^2' | 'square' | 'sqrt' | '1/sqrt' | 'pow' | 'conj' | 'reciprocal' | 'hypot' | 'rem' | 'mod' | 'transpose' | 'hermitian'`
`OutputSignalType`	Output signal type	string — `{'auto'} | 'real' | 'complex'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataTypeStr`	Output data type	string — `'Inherit: Inherit via internal rule' | 'Inherit: Inherit via back propagation' | {'Inherit: Same as first input'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `'off' | {'on'}`
`IntermediateResultsDataTypeStr`	Intermediate results data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit from input' | 'Inherit: Inherit from output'`
`AlgorithmType`	Method	string — `'Exact' | {'Newton-Raphson'}`
`Iterations`	Number of iterations	string — `{'3'}`
Matrix Concatenate (`Concatenate`)		
`NumInputs`	Number of inputs	string — `{'2'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`Mode`	Mode	string — `'Vector' | {'Multidimensional array'}`
`ConcatenateDimension`	Concatenate dimension	string — `{'2'}`
MinMax (`MinMax`)		
`Function`	Function	string — `{'min'} | 'max'`
`Inputs`	Number of input ports	string — `{'1'}`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InputSameDT`	Require all inputs to have the same data type	string — `{'off'} | 'on'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
SaturateOnInteger Overflow	Saturate on integer overflow	string — {'off'} | 'on'
MinMax Running Resettable (MinMax Running Resettable) (masked subsystem)		
Function	Function	string — {'min'} | 'max'
vinit	Initial condition	string — {'0.0'}
Permute Dimensions (PermuteDimensions)		
Order	Order	string — {'[2,1]'}
Polynomial (Polyval)		
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	string — {'Element-wise(.*)'} | 'Matrix(*)'
CollapseMode	Multiply over	string — {'All dimensions'} | 'Specified dimension'
CollapseDim	Dimension	string — {'1'}
SampleTime	Sample time (-1 for inherited)	string — {'-1'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`InputSameDT`	Require all inputs to have same data type	string — `{'off'} | 'on'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'Inherit: Same as first input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | 'Floor' | 'Nearest' | 'Round' | 'Simplest' | {'Zero'}`
`SaturateOnInteger Overflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Product of Elements (`Product`)		
`Inputs`	Number of inputs	string — `{'*'}`
`Multiplication`	Multiplication	string — `{'Element-wise(.*)'} | 'Matrix(*)'`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`CollapseMode`	Multiply over	string — `{'All dimensions'} | 'Specified dimension'`
`CollapseDim`	Dimension	string — `{'1'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InputSameDT`	Require all inputs to have same data type	string — `{'off'} | 'on'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'Inherit: Same as first input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Real-Imag to Complex (`RealImagToComplex`)		

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`Input`	Input	string — `'Real' | 'Imag' | {'Real and imag'}`
`ConstantPart`	Real part or Imag part	string — `{'0'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
Reshape (`Reshape`)		
`OutputDimensionality`	Output dimensionality	string — `{'1-D array'} | 'Column vector (2-D)' | 'Row vector (2-D)' | 'Customize' | 'Derive from reference input port'`
`OutputDimensions`	Output dimensions	string — `{'[1,1]'}`
Rounding Function (`Rounding`)		
`Operator`	Function	string — `{'floor'} | 'ceil' | 'round' | 'fix'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
Sign (`Signum`)		
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
Sine Wave Function (`Sin`)		
`SineType`	Sine type	string — `{'Time based'} | 'Sample based'`
`TimeSource`	Time (t)	string — `'Use simulation time' | {'Use external signal'}`
`Amplitude`	Amplitude	string — `{'1'}`
`Bias`	Bias	string — `{'0'}`
`Frequency`	Frequency (rad/sec)	string — `{'1'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`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	string — `'off' | {'on'}`
Slider Gain (`Slider Gain`) (masked subsystem)		
`low`	Low	string — `{'0'}`
`gain`	Gain	string — `{'1'}`
`high`	High	string — `{'2'}`
Squeeze (`Squeeze`) (masked subsystem)		
None	None	None
Subtract (`Sum`)		
`IconShape`	Icon shape	string — `{'rectangular'} | 'round'`
`Inputs`	List of signs	string — `{'+-'}`
`CollapseMode`	Sum over	string — `{'All dimensions'} | 'Specified dimension'`
`CollapseDim`	Dimension	string — `{'1'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InputSameDT`	Require all inputs to have the same data type	string — `{'off'} | 'on'`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`AccumDataTypeStr`	Accumulator data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as first input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'Inherit: Same as first input' | 'Inherit: Same as accumulator' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock data type settings against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SaturateOnInteger Overflow	Saturate on integer overflow	string — {'off'} | 'on'
Sum (Sum)		
IconShape	Icon shape	string — 'rectangular' | {'round'}
Inputs	List of signs	string — {'|++'}
CollapseMode	Sum over	string — {'All dimensions'} | 'Specified dimension'
CollapseDim	Dimension	string — {'1'}
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
InputSameDT	Require all inputs to have the same data type	string — {'off'} | 'on'
AccumDataTypeStr	Accumulator data type	string — {'Inherit: Inherit via internal rule'} | 'Inherit: Same as first input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
OutMin	Output minimum	string — {'[]'}
OutMax	Output maximum	string — {'[]'}

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'Inherit: Same as first input' | 'Inherit: Same as accumulator' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock data type settings against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnInteger Overflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Sum of Elements (`Sum`)		
`IconShape`	Icon shape	string — `{'rectangular'} | 'round'`
`Inputs`	List of signs	string — `{'+'}`
`CollapseMode`	Sum over	string — `{'All dimensions'} | 'Specified dimension'`
`CollapseDim`	Dimension	string — `{'1'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`InputSameDT`	Require all inputs to have the same data type	string — `{'off'} | 'on'`
`AccumDataTypeStr`	Accumulator data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Same as first input' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'Inherit: Same as first input' | 'Inherit: Same as accumulator' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock data type settings against changes by the fixed-point tools	string — `{'off'} | 'on'`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnInteger Overflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Trigonometric Function (`Trigonometry`)		
`Operator`	Function	string — `{'sin'} | 'cos' | 'tan' | 'asin' | 'acos' | 'atan' | 'atan2' | 'sinh' | 'cosh' | 'tanh' | 'asinh' | 'acosh' | 'atanh' | 'sincos'`
`OutputSignalType`	Output signal type	string — `{'auto'} | 'real' | 'complex'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
Unary Minus (`Unary Minus`)		
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`SaturateOnInteger Overflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
Vector Concatenate (`Concatenate`)		
`NumInputs`	Number of inputs	string — `{'2'}`
`Mode`	Mode	string — `{'Vector'} | 'Multidimensional array'`
Weighted Sample Time Math (`Sample Time Math`)		
`TsampMathOp`	Operation	string — `{'+'} | '-' | '*' | '/' | 'Ts Only' | '1/Ts Only'`
`weightValue`	Weight value	string — `{'1.0'}`

Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
TsampMathImp	Implement using	string — {'Online Calculations'} | 'Offline Scaling Adjustment'
OutDataTypeStr	Output data type	string — {'Inherit via internal rule'} | 'Inherit via back propagation'
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
SaturateOnIntegerOverflow	Saturate on integer overflow	string — {'off'} | 'on'
OutputDataTypeScalingMode	Deprecated	
DoSatur	Deprecated	

Model Verification Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Assertion (Assertion)		
Enabled	Enable assertion	string — 'off' | {'on'}
AssertionFailFcn	Simulation callback when assertion fails	string — {''}
StopWhenAssertionFail	Stop simulation when assertion fails	string — 'off' | {'on'}
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
Check Dynamic Gap (Checks_DGap) (masked subsystem)		
enabled	Enable assertion	string — 'off' | {'on'}

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	string — `'off' | {'on'}`
`export`	Output assertion signal	string — `{'off'} | 'on'`
`icon`	Select icon type	string — `{'graphic'} | 'text'`
Check Dynamic Range (`Checks_DRange`) (masked subsystem)		
`enabled`	Enable assertion	string — `'off' | {'on'}`
`callback`	Simulation callback when assertion fails (optional)	string — `{''}`
`stopWhenAssertionFail`	Stop simulation when assertion fails	string — `'off' | {'on'}`
`export`	Output assertion signal	string — `{'off'} | 'on'`
`icon`	Select icon type	string — `{'graphic'} | 'text'`
Check Static Gap (`Checks_SGap`) (masked subsystem)		
`max`	Upper bound	string — `{'100'}`
`max_included`	Inclusive upper bound	string — `'off' | {'on'}`
`min`	Lower bound	string — `{'0'}`
`min_included`	Inclusive lower bound	string — `'off' | {'on'}`
`enabled`	Enable assertion	string — `'off' | {'on'}`
`callback`	Simulation callback when assertion fails (optional)	string — `{''}`
`stopWhenAssertionFail`	Stop simulation when assertion fails	string — `'off' | {'on'}`
`export`	Output assertion signal	string — `{'off'} | 'on'`

Model Verification Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`icon`	Select icon type	string — `{'graphic'} | 'text'`
Check Static Range (`Checks_SRange`) (masked subsystem)		
`max`	Upper bound	string — `{'100'}`
`max_included`	Inclusive upper bound	string — `'off' | {'on'}`
`min`	Lower bound	string — `{'0'}`
`min_included`	Inclusive lower bound	string — `'off' | {'on'}`
`enabled`	Enable assertion	string — `'off' | {'on'}`
`callback`	Simulation callback when assertion fails (optional)	string — `{''}`
`stopWhenAssertionFail`	Stop simulation when assertion fails	string — `'off' | {'on'}`
`export`	Output assertion signal	string — `{'off'} | 'on'`
`icon`	Select icon type	string — `{'graphic'} | 'text'`
Check Discrete Gradient (`Checks_Gradient`) (masked subsystem)		
`gradient`	Maximum gradient	string — `{'1'}`
`enabled`	Enable assertion	string — `'off' | {'on'}`
`callback`	Simulation callback when assertion fails (optional)	string — `{''}`
`stopWhenAssertionFail`	Stop simulation when assertion fails	string — `'off' | {'on'}`
`export`	Output assertion signal	string — `{'off'} | 'on'`
`icon`	Select icon type	string — `{'graphic'} | 'text'`
Check Dynamic Lower Bound (`Checks_DMin`) (masked subsystem)		
`Enabled`	Enable assertion	string — `'off' | {'on'}`

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	string — `'off' | {'on'}`
`export`	Output assertion signal	string — `{'off'} | 'on'`
`icon`	Select icon type	string — `{'graphic'} | 'text'`
Check Dynamic Upper Bound (`Checks_DMax`) (masked subsystem)		
`enabled`	Enable assertion	string — `'off' | {'on'}`
`callback`	Simulation callback when assertion fails (optional)	string — `{''}`
`stopWhenAssertionFail`	Stop simulation when assertion fails	string — `'off' | {'on'}`
`export`	Output assertion signal	string — `{'off'} | 'on'`
`icon`	Select icon type	string — `{'graphic'} | 'text'`
Check Input Resolution (`Checks_Resolution`) (masked subsystem)		
`resolution`	Resolution	string — `{'1'}`
`enabled`	Enable assertion	string — `'off' | {'on'}`
`callback`	Simulation callback when assertion fails (optional)	string — `{''}`
`stopWhenAssertionFail`	Stop simulation when assertion fails	string — `'off' | {'on'}`
`export`	Output assertion signal	string — `{'off'} | 'on'`
Check Static Lower Bound (`Checks_SMin`) (masked subsystem)		
`min`	Lower bound	string — `{'0'}`
`min_included`	Inclusive boundary	string — `'off' | {'on'}`

Model Verification Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
enabled	Enable assertion	string — 'off' | {'on'}
callback	Simulation callback when assertion fails (optional)	string — {''}
stopWhenAssertionFail	Stop simulation when assertion fails	string — 'off' | {'on'}
export	Output assertion signal	string — {'off'} | 'on'
icon	Select icon type	string — {'graphic'} | 'text'
Check Static Upper Bound (Checks_SMax) (masked subsystem)		
max	Upper bound	string — {'0'}
max_included	Inclusive boundary	string — 'off' | {'on'}
enabled	Enable assertion	string — 'off' | {'on'}
callback	Simulation callback when assertion fails (optional)	string — {''}
stopWhenAssertionFail	Stop simulation when assertion fails	string — 'off' | {'on'}
export	Output assertion signal	string — {'off'} | 'on'
icon	Select icon type	string — {'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	Real-Time Workshop Embedded Coder Flag	string — {''}

Model-Wide Utilities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`DocumentType`	Document Type	string — `{'Text'} | 'RTF' | 'HTML'`
Model Info (`CMBlock`) (masked subsystem)		
`InitialSaveTempField`	InitialSaveTempField	string — `{''}`
`InitialBlockCM`	InitialBlockCM	string — `{'None'}`
`BlockCM`	BlockCM	string — `{'None'}`
`Frame`	Show block frame	string — `'off' | {'on'}`
`SaveTempField`	SaveTempField	string — `{''}`
`DisplayStringWithTags`	DisplayStringWithTags	string — `{'Model Info'}`
`MaskDisplayString`	MaskDisplayString	string — `{'Model Info'}`
`HorizontalTextAlignment`	Horizontal text alignment	string — `{'Center'}`
`LeftAlignmentValue`	LeftAlignmentValue	string — `{'0.5'}`
`SourceBlockDiagram`	SourceBlockDiagram	string — `{'untitled'}`
`TagMaxNumber`	TagMaxNumber	string — `{'20'}`
`CMTag1`	CMTag1	string — `{''}`
`CMTag2`	CMTag2	string — `{''}`
`CMTag3`	CMTag3	string — `{''}`
`CMTag4`	CMTag4	string — `{''}`
`CMTag5`	CMTag5	string — `{''}`
`CMTag6`	CMTag6	string — `{''}`
`CMTag7`	CMTag7	string — `{''}`
`CMTag8`	CMTag8	string — `{''}`
`CMTag9`	CMTag9	string — `{''}`
`CMTag10`	CMTag10	string — `{''}`
`CMTag11`	CMTag11	string — `{''}`

Model-Wide Utilities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`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)		
`TriggerType`	Trigger type	string — `{'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
Action Port (`ActionPort`)		
`InitializeStates`	States when execution is resumed	string — `{'held'} | 'reset'`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`PropagateVarSize`	Propagate sizes of variable-size signals	string — `{'Only when execution is resumed'} | 'During execution'`
Atomic Subsystem (`SubSystem`)		
`ShowPortLabels`	Show port labels **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{''}`
`MemberBlocks`	Member blocks	string — `{''}`
`Permissions`	Read/Write permissions	string — `{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'`
`ErrorFcn`	Name of error callback function	string — `{''}`
`PermitHierarchical Resolution`	Permit hierarchical resolution	string — `{'All'} | 'ExplicitOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	string — `'off' | {'on'}`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	string — `{'off'} | 'on'`
`PropExecContext OutsideSubsystem`	Propagate execution context across subsystem boundary	string — `{'off'} | 'on'`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`RTWSystemCode`	Real-Time Workshop system code	string — `{'Auto'} | 'Inline' | 'Function' | 'Reusable function'`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `{'Auto'} | 'Use subsystem name' | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string — `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'`
`RTWFileName`	Real-Time Workshop filename (no extension)	string — `{''}`
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'off'} | 'on'` Read-only
Code Reuse Subsystem (`SubSystem`)		

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`ShowPortLabels`	Show port labels **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{''}`
`MemberBlocks`	Member blocks	string — `{''}`
`Permissions`	Read/Write permissions	string — `{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'`
`ErrorFcn`	Name of error callback function	string — `{''}`
`PermitHierarchical Resolution`	Permit hierarchical resolution	string — `{'All'} | 'ExplicitOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	string — `'off' | {'on'}`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	string — `{'off'} | 'on'`
`PropExecContext OutsideSubsystem`	Propagate execution context across subsystem boundary	string — `{'off'} | 'on'`
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`RTWSystemCode`	Real-Time Workshop system code	string — `'Auto' | 'Inline' | 'Function' | {'Reusable function'}`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `'Auto' | {'Use subsystem name'} | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string — `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `'Auto' | 'Use subsystem name' | {'Use function name'} | 'User specified'`
`RTWFileName`	Real-Time Workshop filename (no extension)	string — `{''}`
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'off'} | 'on'` Read-only
Configurable Subsystem (`SubSystem`)		

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`ShowPortLabels`	Show port labels **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{'self'}`
`MemberBlocks`	Member blocks	string — `{''}`
`Permissions`	Read/Write permissions	string — `{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'`
`ErrorFcn`	Name of error callback function	string — `{''}`
`PermitHierarchical Resolution`	Permit hierarchical resolution	string — `{'All'} | 'ExplicitOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	string — `{'off'} | 'on'`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	string — `{'off'} | 'on'`
`PropExecContext OutsideSubsystem`	Propagate execution context across subsystem boundary	string — `{'off'} | 'on'`
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`RTWSystemCode`	Real-Time Workshop system code	string — `{'Auto'} | 'Inline' | 'Function' | 'Reusable function'`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `{'Auto'} | 'Use subsystem name' | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string — `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'`
`RTWFileName`	Real-Time Workshop Workshop (no extension)	string — `{''}`
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'on'} | 'off'` Read-only
Enable (`EnablePort`)		
`StatesWhenEnabling`	States when enabling	string — `{'held'} | 'reset'`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`PropagateVarSize`	Propagate sizes of variable-size signals	string — `{'Only when enabling'} | 'During execution'`
`ShowOutputPort`	Show output port	string — `{'off'} | 'on'`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
Enabled and Triggered Subsystem (`SubSystem`)		
`ShowPortLabels`	Show port labels **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{''}`
`MemberBlocks`	Member blocks	string — `{''}`
`Permissions`	Read/Write permissions	string — `{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'`
`ErrorFcn`	Name of error callback function	string — `{''}`
`PermitHierarchical Resolution`	Permit hierarchical resolution	string — `{'All'} | 'ExplicitOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	string — `'off' | {'on'}`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	string — `{'off'} | 'on'`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`PropExecContext OutsideSubsystem`	Propagate execution context across subsystem boundary	string — `{'off'} | 'on'`
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`RTWSystemCode`	Real-Time Workshop system code	string — `{'Auto'} | 'Inline' | 'Function' | 'Reusable function'`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `{'Auto'} | 'Use subsystem name' | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string — `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'`
`RTWFileName`	Real-Time Workshop filename (no extension)	string — `{''}`
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
IsSubsystemVirtual		boolean — {'off'} | 'on' Read-only
Enabled Subsystem (SubSystem)		
ShowPortLabels	Show port labels **Note** The values 'off' and 'on' are for backward compatibility only and should not be used in new models or when updating existing models.	string — 'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'
BlockChoice	Block choice	string — {''}
TemplateBlock	Template block	string — {''}
MemberBlocks	Member blocks	string — {''}
Permissions	Read/Write permissions	string — {'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string — {''}
PermitHierarchical Resolution	Permit hierarchical resolution	string — {'All'} | 'ExplicitOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	string — 'off' | {'on'}
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	string — {'off'} | 'on'
PropExecContext OutsideSubsystem	Propagate execution context across subsystem boundary	string — {'off'} | 'on'
CheckFcnCallInp InsideContextMsg	Warn if function-call inputs are context-specific	string — {'off'} | 'on'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SystemSampleTime	Sample time (-1 for inherited)	string — {'-1'}
RTWSystemCode	Real-Time Workshop system code	string — {'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	Real-Time Workshop function name options	string — {'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	Real-Time Workshop function name	string — {''}
RTWFileNameOpts	Real-Time Workshop filename options	string — {'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	Real-Time Workshop filename (no extension)	string — {''}
DataTypeOverride	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — {'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'
MinMaxOverflowLogging	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — {'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'
IsSubsystemVirtual		boolean — {'off'} | 'on' Read-only
For Iterator (ForIterator)		
ResetStates	States when starting	string — {'held'} | 'reset'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`IterationSource`	Iteration limit source	string — `{'internal'} | 'external'`
`IterationLimit`	Iteration limit	string — `{'5'}`
`ExternalIncrement`	Set next i (iteration variable) externally	string — `{'off'} | 'on'`
`ShowIterationPort`	Show iteration variable	string — `'off' | {'on'}`
`IndexMode`	Index mode	string — `'Zero-based' | {'One-based'}`
`IterationVariable DataType`	Iteration variable data type	string — `{'int32'} | 'int16' | 'int8' | 'double'`
For Iterator Subsystem (`SubSystem`)		
`ShowPortLabels`	Show port labels **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{''}`
`MemberBlocks`	Member blocks	string — `{''}`
`Permissions`	Read/Write permissions	string — `{'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	string — `{'All'} | 'ExplicitOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	string — `'off' | {'on'}`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	string — `{'off'} | 'on'`
`PropExecContext OutsideSubsystem`	Propagate execution context across subsystem boundary	string — `{'off'} | 'on'`
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`RTWSystemCode`	Real-Time Workshop system code	string — `{'Auto'} | 'Inline' | 'Function' | 'Reusable function'`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `{'Auto'} | 'Use subsystem name' | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string — `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'`
`RTWFileName`	Real-Time Workshop filename (no extension)	string — `{''}`
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation . Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'off'} | 'on'` Read-only
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 **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{''}`
`MemberBlocks`	Member blocks	string — `{''}`
`Permissions`	Read/Write permissions	string — `{'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	string — `{'All'} | 'ExplicitOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	string — `'off' | {'on'}`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	string — `{'off'} | 'on'`
`PropExecContext OutsideSubsystem`	Propagate execution context across subsystem boundary	string — `{'off'} | 'on'`
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`RTWSystemCode`	Real-Time Workshop system code	string — `{'Auto'} | 'Inline' | 'Function' | 'Reusable function'`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `{'Auto'} | 'Use subsystem name' | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string — `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'`
`RTWFileName`	Real-Time Workshop filename (no extension)	string — `{''}`
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'off'} | 'on'` Read-only
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	string — `'off' | {'on'}`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
If Action Subsystem (`SubSystem`)		
`ShowPortLabels`	Show port labels **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{''}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
MemberBlocks	Member blocks	string — {''}
Permissions	Read/Write permissions	string — {'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string — {''}
PermitHierarchical Resolution	Permit hierarchical resolution	string — {'All'} | 'ExplicitOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	string — 'off' | {'on'}
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	string — {'off'} | 'on'
PropExecContext OutsideSubsystem	Propagate execution context across subsystem boundary	string — {'off'} | 'on'
CheckFcnCallInp InsideContextMsg	Warn if function-call inputs are context-specific	string — {'off'} | 'on'
SystemSampleTime	Sample time (-1 for inherited)	string — {'-1'}
RTWSystemCode	Real-Time Workshop system code	string — {'Auto'} | 'Inline' | 'Function' | 'Reusable function'
RTWFcnNameOpts	Real-Time Workshop function name options	string — {'Auto'} | 'Use subsystem name' | 'User specified'
RTWFcnName	Real-Time Workshop function name	string — {''}
RTWFileNameOpts	Real-Time Workshop filename options	string — {'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'
RTWFileName	Real-Time Workshop filename (no extension)	string — {''}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'off'} | 'on'` Read-only
In1 (`Inport`)		
`Port`	Port number	string — `{'1'}`
`IconDisplay`	Icon display	string — `'Signal name' | {'Port number'} | 'Port number and signal name'`
`LatchByDelaying OutsideSignal`	Latch input by delaying outside signal	string — `{'off'} | 'on'`
`LatchByCopying InsideSignal`	Latch input by copying inside signal	string — `{'off'} | 'on'`
`Interpolate`	Interpolate data	string — `'off' | {'on'}`
`UseBusObject`	Specify properties via bus object	string — `{'off'} | 'on'`
`BusObject`	Bus object for specifying bus properties	string — `{'BusObject'}`
`BusOutputAsStruct`	Output as nonvirtual bus	string — `{'off'} | 'on'`
`PortDimensions`	Port dimensions (-1 for inherited)	string — `{'-1'}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
OutMin	Minimum	string — {'[]'}
OutMax	Maximum	string — {'[]'}
OutDataTypeStr	Data type	string — {'Inherit: auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum: <class name>'
LockScale	Lock output data type setting against changes by the fixed-point tools	string — {'off'} | 'on'
SignalType	Signal type	string — {'auto'} | 'real' | 'complex'
SamplingMode	Sampling mode	string — {'auto'} | 'Sample based' | 'Frame based'
Model (ModelReference)		
ModelNameDialog	The name of the referenced model exactly as you typed it in, with any surrounding whitespace removed. When you set ModelNameDialog programmatically or with the GUI, Simulink automatically sets the values of ModelName and ModelFile based on the value of ModelNameDialog.	string — {'<Enter Model Name>'}

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`ModelName`	The value of `ModelNameDialog` stripped of any extension (`.mdl` or `.mdlp`) that you provided. For backward compatibility, setting `ModelName` programatically actually sets `ModelNameDialog`, which then sets `ModelName` as described. You cannot use `get_param` to obtain the `ModelName` of a protected model, because the name without a suffix would be ambiguous. Use `get_param` on `ModelFile` instead. You can test `ProtectedModel` to determine programmatically whether a referenced model is protected.	string — Set automatically when `ModelNameDialog` is set.
`ModelFile`	The value of `ModelNameDialog` with an extension (`.mdl` or `.mdlp`). If the user omits the extension, Simulink searches the MATLAB path first for *`ModelName`*`.mdl`, and if that search fails, again for *`ModelName`*`.mdlp`. The suffix of the first match Simulink finds becomes the suffix of `ModelFile`. Setting `ModelFile` programatically actually sets `ModelNameDialog`, which then sets `ModelFile` as described.	string — Set automatically when `ModelNameDialog` is set.

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`ProtectedModel`	Read-only boolean indicating whether the model referenced by the block is protected (`on`) or unprotected (`off`).	boolean — `'off' | 'on'` — Set automatically when `ModelNameDialog` is set.
`ParameterArgumentNames`	Model arguments	string — `{''}`
`ParameterArgumentValues`	Model argument values (for this instance)	string — `{''}`
`SimulationMode`	Specifies whether to simulate the model by generating and executing code or by interpreting the model in Simulink software.	string — `{'Accelerator'} | 'Normal' | 'Processor-in-the-loop (PIL)'`
`Variant`	Specifies whether the Model block references variant models.	string — `{'off'} | 'on'`
`Variants`	An array of `variant` structures where each element specifies one variant. The structure fields are as follows:	array — [ ]
	`variant.Name`– The name of the `Simulink.Variant` object that represents the variant to which this element applies.	string — `{''}`
	`variant.ModelName`– The name of the referenced model associated with the specified variant object in this Model block.	string — `{''}`
	`variant.ParameterArgumentNames`– Noneditable string containing the names of the model arguments for which the	string — `{''}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
	Model block must supply values.	
	`variant.ParameterArgumentValues` The values to supply for the model arguments when this variant is the active variant.	string — `{''}`
	`variant.SimulationMode`– The execution mode to use when this variant is the active variant.	string — `{'Accelerator'} | 'Normal' | 'Processor-in-the-loop (PIL)'`
`OverrideUsingVariant`	Whether to override the variant conditions and make a specified variant the active variant, and if so, the name of that variant.	string — `{''}` The value is the empty string if no overriding variant object is specified; or the name of the overriding object.
`ActiveVariant`	The variant that is currently active, either because its variant condition is `true` or `OverrideUsingVariant` has overridden the variant conditions and specified this variant.	string — `{''}` The value is the empty string if no variant is active; or the name of the active variant.

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`GeneratePreprocessorConditionals`	Locally controls whether generated code contains preprocessor conditionals. This parameter is relevant only to code generation, and has no effect on the behavior of a model in Simulink. The parameter is available only for an ERT target when **Inline parameters** is selected. See "Generating Code Variants for Variant Models" for more information.	string — `{'off'} | 'on'`
`AvailSigsInstanceProps`		handle vector — `{''}`
`AvailSigsDefaultProps`		handle vector — `{''}`
`DefaultDataLogging`		string — `{'off'} | 'on'`
Out1 (`Outport`)		
Port	Port number	string — `{'1'}`
`IconDisplay`	Icon display	string — `'Signal name' | {'Port number'} | 'Port number and signal name'`
`UseBusObject`	Specify properties via bus object	string — `{'off'} | 'on'`
`BusObject`	Bus object for validating input bus	string — `{'BusObject'}`
`BusOutputAsStruct`	Output as nonvirtual bus in parent model	string — `{'off'} | 'on'`
`PortDimensions`	Port dimensions (-1 for inherited)	string — `{'-1'}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`VarSizeSig`	Variable-size signal	string — `{'Inherit'} | 'No' | 'Yes'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`OutMin`	Minimum	string — `{'[]'}`
`OutMax`	Maximum	string — `{'[]'}`
`OutDataTypeStr`	Data type	string — `{'Inherit: auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum: <class name>'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`SignalType`	Signal type	string — `{'auto'} | 'real' | 'complex'`
`SamplingMode`	Sampling mode	string — `{'auto'} | 'Sample based' | 'Frame based'`
`OutputWhenDisabled`	Output when disabled	string — `{'held'} | 'reset'`
`InitialOutput`	Initial output	string — `{'[]'}`
Subsystem (`SubSystem`)		

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`ShowPortLabels`	Show port labels **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{''}`
`MemberBlocks`	Member blocks	string — `{''}`
`Permissions`	Read/Write permissions	string — `{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'`
`ErrorFcn`	Name of error callback function	string — `{''}`
`PermitHierarchical Resolution`	Permit hierarchical resolution	string — `{'All'} | 'ExplicitOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	string — `{'off'} | 'on'`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	string — `{'off'} | 'on'`
`PropExecContext OutsideSubsystem`	Propagate execution context across subsystem boundary	string — `{'off'} | 'on'`
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`RTWSystemCode`	Real-Time Workshop system code	string — `{'Auto'} | 'Inline' | 'Function' | 'Reusable function'`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `{'Auto'} | 'Use subsystem name' | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'`
`RTWFileName`	Real-Time Workshop filename (no extension)	string — `{''}`
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'on'} | 'off'` Read-only
`Virtual`	For internal use	
Switch Case (`SwitchCase`)		
`CaseConditions`	Case conditions (e.g., {1,[2,3]})	string — `{'{1}'}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ShowDefaultCase	Show default case	string — 'off' | {'on'}
ZeroCross	Enable zero-crossing detection	string — 'off' | {'on'}
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
CaseShowDefault	Deprecated	
Switch Case Action Subsystem (SubSystem)		
ShowPortLabels	Show port labels **Note** The values 'off' and 'on' are for backward compatibility only and should not be used in new models or when updating existing models.	string — 'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'
BlockChoice	Block choice	string — {''}
TemplateBlock	Template block	string — {''}
MemberBlocks	Member blocks	string — {''}
Permissions	Read/Write permissions	string — {'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'
ErrorFcn	Name of error callback function	string — {''}
PermitHierarchical Resolution	Permit hierarchical resolution	string — {'All'} | 'ExplicitOnly' | 'None'
TreatAsAtomicUnit	Treat as atomic unit	string — 'off' | {'on'}
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	string — {'off'} | 'on'
PropExecContext OutsideSubsystem	Propagate execution context across subsystem boundary	string — {'off'} | 'on'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`RTWSystemCode`	Real-Time Workshop system code	string — `{'Auto'} | 'Inline' | 'Function' | 'Reusable function'`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `{'Auto'} | 'Use subsystem name' | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string — `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'`
`RTWFileName`	Real-Time Workshop filename (no extension)	string — `{''}`
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'off'} | 'on'` Read-only
Trigger (`TriggerPort`)		

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`TriggerType`	Trigger type	string — `{'rising'} | 'falling' | 'either' | 'function-call'`
`StatesWhenEnabling`	States when enabling	string — `{'held'} | 'reset' | 'inherit'`
`PropagateVarSize`	Propagate sizes of variable-size signals	string — `{'During execution'} | 'Only when enabling'`
`ShowOutputPort`	Show output port	string — `{'off'} | 'on'`
`OutputDataType`	Output data type	string — `{'auto'} | 'double' | 'int8'`
`SampleTimeType`	Sample time type	string — `{'triggered'} | 'periodic'`
`SampleTime`	Sample time	string — `{'1'}`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
Triggered Subsystem (`SubSystem`)		
`ShowPortLabels`	Show port labels **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{''}`
`MemberBlocks`	Member blocks	string — `{''}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`Permissions`	Read/Write permissions	string — `{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'`
`ErrorFcn`	Name of error callback function	string — `{''}`
`PermitHierarchical Resolution`	Permit hierarchical resolution	string — `{'All'} | 'ExplicitOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	string — `'off' | {'on'}`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	string — `{'off'} | 'on'`
`PropExecContext OutsideSubsystem`	Propagate execution context across subsystem boundary	string — `{'off'} | 'on'`
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`RTWSystemCode`	Real-Time Workshop system code	string — `{'Auto'} | 'Inline' | 'Function' | 'Reusable function'`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `{'Auto'} | 'Use subsystem name' | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string — `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'`
`RTWFileName`	Real-Time Workshop filename (no extension)	string — `{''}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'off'} | 'on'` Read-only
While Iterator (`WhileIterator`)		
`MaxIters`	Maximum number of iterations (-1 for unlimited)	string — `{'5'}`
`WhileBlockType`	While loop type	string — `{'while'} | 'do-while'`
`ResetStates`	States when starting	string — `{'held'} | 'reset'`
`ShowIterationPort`	Show iteration number port	string — `{'off'} | 'on'`
`OutputDataType`	Output data type	string — `{'int32'} | 'int16' | 'int8' | 'double'`
While Iterator Subsystem (`SubSystem`)		

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`ShowPortLabels`	Show port labels **Note** The values `'off'` and `'on'` are for backward compatibility only and should not be used in new models or when updating existing models.	string — `'none' | {'FromPortIcon'} | 'FromPortBlockName' | 'SignalName' | 'off' | 'on'`
`BlockChoice`	Block choice	string — `{''}`
`TemplateBlock`	Template block	string — `{''}`
`MemberBlocks`	Member blocks	string — `{''}`
`Permissions`	Read/Write permissions	string — `{'ReadWrite'} | 'ReadOnly' | 'NoReadOrWrite'`
`ErrorFcn`	Name of error callback function	string — `{''}`
`PermitHierarchical Resolution`	Permit hierarchical resolution	string — `{'All'} | 'ExplicitOnly' | 'None'`
`TreatAsAtomicUnit`	Treat as atomic unit	string — `'off' | {'on'}`
`MinAlgLoopOccurrences`	Minimize algebraic loop occurrences	string — `{'off'} | 'on'`
`PropExecContext OutsideSubsystem`	Propagate execution context across subsystem boundary	string — `{'off'} | 'on'`
`CheckFcnCallInp InsideContextMsg`	Warn if function-call inputs are context-specific	string — `{'off'} | 'on'`
`SystemSampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`RTWSystemCode`	Real-Time Workshop system code	string — `{'Auto'} | 'Inline' | 'Function' | 'Reusable function'`
`RTWFcnNameOpts`	Real-Time Workshop function name options	string — `{'Auto'} | 'Use subsystem name' | 'User specified'`
`RTWFcnName`	Real-Time Workshop function name	string — `{''}`
`RTWFileNameOpts`	Real-Time Workshop filename options	string — `{'Auto'} | 'Use subsystem name' | 'Use function name' | 'User specified'`
`RTWFileName`	Real-Time Workshop filename (no extension)	string — `{''}`
`DataTypeOverride`	Specifies data type used to override fixed-point data types. Set by **Data type override** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'ScaledDoubles' | 'TrueDoubles' | 'TrueSingles' | 'ForceOff'`
`MinMaxOverflowLogging`	Setting for fixed-point instrumentation. Set by **Fixed-point instrumentation mode** on the Fixed-Point Tool.	string — `{'UseLocalSettings'} | 'MinMaxAndOverflow' | 'OverflowOnly' | 'ForceOff'`
`IsSubsystemVirtual`		boolean — `{'off'} | 'on'` Read-only

Signal Attributes Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Bus to Vector (`BusToVector`)		

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Data Type Conversion (`DataTypeConversion`)		
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via back propagation'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum: <class name>'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`ConvertRealWorld`	Input and output to have equal	string — `{'Real World Value (RWV)'} | 'Stored Integer (SI)'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnInteger Overflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
Data Type Conversion Inherited (`Conversion Inherited`) (masked subsystem)		
`ConvertRealWorld`	Input and Output to have equal	string — `{'Real World Value'} | 'Stored Integer'`

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
DoSatur	Saturate to max or min when overflows occur	string — {'off'} | 'on'
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	string — 'Specify via dialog' | {'Inherit via propagation rule'}
PropDataType	1.1. Propagated data type (e.g., fixdt(1,16), fixdt('single'))	string — {'fixdt(1,16)'}
IfRefDouble	1.1. If any reference input is double, output is	string — {'double'} | 'single'
IfRefSingle	1.2. If any reference input is single, output is	string — 'double' | {'single'}
IsSigned	1.3. Is-Signed	string — 'IsSigned1' | 'IsSigned2' | {'IsSigned1 or IsSigned2'} | 'TRUE' | 'FALSE'
NumBitsBase	1.4.1. Number-of-Bits: Base	string — 'NumBits1' | 'NumBits2' | {'max([NumBits1 NumBits2])'} | 'min([NumBits1 NumBits2])' | 'NumBits1+NumBits2'

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`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	string — `'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]'}`
`SlopeBase`	2.1.1. Slope: Base	string — `'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'}`

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
BiasBase	2.2.1. Bias: Base	string — {'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'}
Probe (Probe)		
ProbeWidth	Probe width	string — 'off' | {'on'}
ProbeSampleTime	Probe sample time	string — 'off' | {'on'}
ProbeComplexSignal	Detect complex signal	string — 'off' | {'on'}
ProbeSignalDimensions	Probe signal dimensions	string — 'off' | {'on'}
ProbeFramedSignal	Detect framed signal	string — 'off' | {'on'}
ProbeWidthDataType	Data type for width	string — {'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
`ProbeSampleTimeDataType`	Data type for sample time	string — `{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Same as input'`
`ProbeComplexityDataType`	Data type for signal complexity	string — `{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Same as input'`
`ProbeDimensionsDataType`	Data type for signal dimensions	string — `{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Same as input'`
`ProbeFrameDataType`	Data type for signal frames	string — `{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'Same as input'`
Rate Transition (`RateTransition`)		
`Integrity`	Ensure data integrity during data transfer	string — `'off' | {'on'}`
`Deterministic`	Ensure deterministic data transfer (maximum delay)	string — `'off' | {'on'}`
`X0`	Initial conditions	string — `{'0'}`

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutPortSampleTimeOpt	Output port sample time options	string — {'Specify'} | 'Inherit' | 'Multiple of input port sample time'
OutPortSampleTimeMultiple	Sample time multiple (>0)	string — {'1'}
OutPortSampleTime	Output port sample time	string — {'-1'}
Signal Conversion (SignalConversion)		
ConversionOutput	Output	string — {'Contiguous copy'} | 'Bus copy' | 'Virtual bus' | 'Nonvirtual bus'
OverrideOpt	Exclude this block from 'Block reduction' optimization	string — {'off'} | 'on'
Signal Specification (SignalSpecification)		
Dimensions	Dimensions (-1 for inherited)	string — {'-1'}
VarSizeSig	Variable-size signal	string — {'Inherit'} | 'No' | 'Yes'
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
OutMin	Minimum	string — {'[]'}
OutMax	Maximum	string — {'[]'}
OutDataTypeStr	Data type	string — {'Inherit: auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum: <class name>'

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`SignalType`	Signal type	string — `{'auto'} | 'real' | 'complex'`
`SamplingMode`	Sampling mode	string — `{'auto'} | 'Sample based' | 'Frame based'`
Weighted Sample Time (`Sample Time Math`)		
`TsampMathOp`	Operation	string — `'+' | '-' | '*' | '/' | {'Ts Only'} | '1/Ts Only'`
`weightValue`	Weight value	string — `{'1.0'}`
`TsampMathImp`	Implement using	string — `{'Online Calculations'} | 'Offline Scaling Adjustment'`
`OutDataTypeStr`	Output data type	string — `{'Inherit via internal rule'} | 'Inherit via back propagation'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
`OutputDataTypeScalingMode`	Deprecated	
`DoSatur`	Deprecated	
Width (`Width`)		

Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutputDataTypeScaling Mode`	Output data type mode	string — `{'Choose intrinsic data type'} | 'Inherit via back propagation' | 'All ports same datatype'`
`DataType`	Output data type	string — `{'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32'`

Signal Routing Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Bus Assignment (`BusAssignment`)		
`AssignedSignals`	Signals that are being assigned	string — `{''}`
`InputSignals`	Signals in the bus	matrix — `{'{}'}`
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`		string — `'none' | 'signals' | {'bar'}`
`UseBusObject`	Specify properties via bus object	string — `{'off'} | 'on'`

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`BusObject`	Bus object for specifying bus properties	string — `{'BusObject'}`
`NonVirtualBus`	Output as nonvirtual bus	string — `{'off'} | 'on'`
Bus Selector (`BusSelector`)		
`OutputSignals`	Specifies the names of the input bus signals selected for output. Corresponds to the **Selected signals** list on the block parameter dialog box.	string — `{'signal1,signal2'}`
`OutputAsBus`	Output as bus	string — `{'off'} | 'on'`
`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'}`
`ReadBeforeWriteMsg`	Detect read before write	string — `'none' | {'warning'} | 'error'`
`WriteAfterWriteMsg`	Detect write after write	string — `'none' | {'warning'} | 'error'`
`WriteAfterReadMsg`	Detect write after read	string — `'none' | {'warning'} | 'error'`
`InitialValue`	Initial value	string — `{'0'}`
`StateMustResolveTo SignalObject`	Data store name must resolve to Simulink signal object	string — `{'off'} | 'on'`
`RTWStateStorageClass`	Real-Time Workshop storage class	string — `{'Auto'} | 'ExportedGlobal' | 'ImportedExtern' | 'ImportedExternPointer'`

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWStateStorageType Qualifier	Real-Time Workshop type qualifier	string — {''}
VectorParams1D	Interpret vector parameters as 1-D	string — 'off' | {'on'}
ShowAdditionalParam	Show additional parameters	string — {'off'} | 'on'
OutMin	Minimum	string — {'[]'}
OutMax	Maximum	string — {'[]'}
OutDataTypeStr	Data type	string — {'Inherit: auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum: <class name>'
LockScale	Lock output data type setting against changes by the fixed-point tools	string — {'off'} | 'on'
SignalType	Signal type	string — {'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'}

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`DisplayOption`	Display option	string — `'none' | {'bar'}`
`BusSelectionMode`	Bus selection mode	string — `{'off'} | 'on'`
Environment Controller (`Environment Controller`) (masked subsystem)		
From (`From`)		
`GotoTag`	Goto tag	string — `{'A'}`
`IconDisplay`	Icon display	string — `'Signal name' | {'Tag'} | 'Tag and signal name'`
Goto (`Goto`)		
`GotoTag`	Tag	string — `{'A'}`
`IconDisplay`	Icon display	string — `'Signal name' | {'Tag'} | 'Tag and signal name'`
`TagVisibility`	Tag visibility	string — `{'local'} | 'scoped' | 'global'`
Goto Tag Visibility (`GotoTagVisibility`)		
`GotoTag`	Goto tag	string — `{'A'}`
Index Vector (`Multi-Port Switch`)		
`Inputs`	Number of inputs	string — `{'1'}`
`zeroidx`	Use zero-based indexing	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InputSameDT`	Require all data port inputs to have the same data type	string — `{'off'} | 'on'`

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutDataTypeStr	Output data type	string — {'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'
RndMeth	Integer rounding mode	string — 'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'
SaturateOnIntegerOverflow	Saturate on integer overflow	string — {'off'} | 'on'
Manual Switch (Manual Switch) (masked subsystem)		
sw	Current setting	string — {'1'}
action	Action	string — {'0'}
AllowDiffInputSignals	Allow different input sizes (Results in variable-size output signal)	string — {'off'} | 'on'
Merge (Merge)		
Inputs	Number of inputs	string — {'2'}
InitialOutput	Initial output	string — {'[]'}
AllowUnequalInputPortWidths	Allow unequal port widths	string — {'off'} | 'on'
InputPortOffsets	Input port offsets	string — {'[]'}
Multiport Switch (Multi-Port Switch)		
Inputs	Number of inputs	string — {'3'}
zeroidx	Use zero-based indexing	string — {'off'} | 'on'

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InputSameDT`	Require all data port inputs to have the same data type	string — `{'off'} | 'on'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnInteger Overflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
`AllowDiffInputSignals`	Allow different input sizes (Results in variable-size output signal)	string — `{'off'} | 'on'`
Mux (`Mux`)		
`Inputs`	Number of inputs	string — `{'2'}`
`DisplayOption`	Display option	string — `'none' | 'signals' | {'bar'}`

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
UseBusObject	For internal use	
BusObject	For internal use	
NonVirtualBus	For internal use	
Selector (Selector)		
NumberOfDimensions	Number of input dimensions	string — {'1'}
IndexMode	Index mode	string — 'Zero-based' | {'One-based'}
IndexOptionArray	Index Option	string — 'Select all' | {'Index vector (dialog)'} | 'Index vector (port)' | 'Starting index (dialog)' | 'Starting index (port)'
IndexParamArray	Index	cell array
OutputSizeArray	Output Size	cell array
InputPortWidth	Input port size	string — {'1'}
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
IndexOptions	See IndexOptionArray parameter for more information.	
Indices	See IndexParamArray parameter for more information.	
OutputSizes	See OutputSizeArray parameter for more information.	
Switch (Switch)		
Criteria	Criteria for passing first input	string — {'u2 >= Threshold'} | 'u2 > Threshold' | 'u2 ~= 0'

Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`Threshold`	Threshold	string — `{'0'}`
`ZeroCross`	Enable zero-crossing detection	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`InputSameDT`	Require all data port inputs to have the same data type	string — `{'off'} | 'on'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit via internal rule'} | 'Inherit: Inherit via back propagation' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum: <class name>'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`SaturateOnIntegerOverflow`	Saturate on integer overflow	string — `{'off'} | 'on'`
`AllowDiffInputSignals`	Allow different input sizes (Results in variable-size output signal)	string — `{'off'} | 'on'`

Sinks Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Display (Display)		
Format	Format	string — {'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	string — {'off'} | 'on'
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
Floating Scope (Scope)		
Floating		string — 'off' | {'on'}
Location		vector — {'[376 294 700 533]'}
Open		string — {'off'} | 'on'
NumInputPorts		string {'1'}
TickLabels		string — 'on' | 'off' | {'OneTimeTick'}
ZoomMode		string — {'on'} | 'xonly' | 'yonly'
AxesTitles		string
Grid		string — 'off' | {'on'} | 'xonly' | 'yonly'
TimeRange		string — {'auto'}
YMin		string — {'-5'}
YMax		string — {'5'}

Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SaveToWorkspace		string — {'off'} | 'on'
SaveName		string — {'ScopeData'}
DataFormat		string — {'StructureWithTime'} | 'Structure' | 'Array'
LimitDataPoints		string — 'off' | {'on'}
MaxDataPoints		string — {'5000'}
Decimation		string — {'1'}
SampleInput		string — {'off'} | 'on'
SampleTime		string — {'0'}
Out1 (Outport)		
Port	Port number	string — {'1'}
IconDisplay	Icon display	string — 'Signal name' | {'Port number'} | 'Port number and signal name'
UseBusObject	Specify properties via bus object	string — {'off'} | 'on'
BusObject	Bus object for specifying bus properties	string — {'BusObject'}
BusOutputAsStruct	Output as nonvirtual bus in parent model	string — {'off'} | 'on'
PortDimensions	Port dimensions (-1 for inherited)	string — {'-1'}
VarSizeSig	Variable-size signal	string — {'Inherit'} | 'No' | 'Yes'
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
OutMin	Minimum	string — {'[]'}
OutMax	Maximum	string — {'[]'}

Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutDataTypeStr	Data type	string — {'Inherit: auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum: <class name>'
LockScale	Lock output data type setting against changes by the fixed-point tools	string — {'off'} | 'on'
SignalType	Signal type	string — {'auto'} | 'real' | 'complex'
SamplingMode	Sampling mode	string — {'auto'} | 'Sample based' | 'Frame based'
OutputWhenDisabled	Output when disabled	string — {'held'} | 'reset'
InitialOutput	Initial output	string — {'[]'}
Scope (Scope)		
Floating		string — {'off'} | 'on'
Location		vector — {'[188 390 512 629]'}
Open		string — {'off'} | 'on'
NumInputPorts		string — {'1'}
TickLabels		string — 'on' | 'off' | {'OneTimeTick'}
ZoomMode		string — {'on'} | 'xonly' | 'yonly'
AxesTitles		string

Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Grid		string — 'off' | {'on'} | 'xonly' | 'yonly'
TimeRange		string — {'auto'}
YMin		string — {'-5'}
YMax		string — {'5'}
SaveToWorkspace		string — {'off'} | 'on'
SaveName		string — {'ScopeData1'}
DataFormat		string — {'StructureWithTime'} | 'Structure' | 'Array'
LimitDataPoints		string — 'off' | {'on'}
MaxDataPoints		string — {'5000'}
Decimation		string — {'1'}
SampleInput		string — {'off'} | 'on'
SampleTime		string — {'0'}
Stop Simulation		
Terminator		
To File (ToFile)		
Filename	Filename	string — {'untitled.mat'}
MatrixName	Variable name	string — {'ans'}
Decimation	Decimation	string — {'1'}
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'}

Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string — {'-1'}
SaveFormat	Save format	string — 'Structure With Time' | {'Structure'} | 'Array'
FixptAsFi	Log fixed-point data as an fi object	string — {'off'} | 'on'
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	string — 'off' | {'on'}
Chirp Signal (chirp) (masked subsystem)		
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	string — 'off' | {'on'}

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Clock (`Clock`)		
`DisplayTime`	Display time	string — `{'off'} | 'on'`
`Decimation`	Decimation	string — `{'10'}`
Constant (`Constant`)		
`Value`	Constant value	string — `{'1'}`
`VectorParams1D`	Interpret vector parameters as 1-D	string — `'off' | {'on'}`
`SamplingMode`	Sampling mode	string — `{'Sample based'} | 'Frame based'`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `{'Inherit: Inherit from 'Constant value''} | 'Inherit: Inherit via back propagation' | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'fixdt(1,16)' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum: <class name>'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`SampleTime`	Sample time	string — `{'inf'}`
`FramePeriod`	Frame period	string — `{'inf'}`
Counter Free-Running (`Counter Free-Running`) (masked subsystem)		

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
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'}
Enumerated Constant (EnumeratedConstant)		
OutDataTypeStr	Output data type	string — {'Enum: SlDemoSign'}
Value	Value	string — 'SlDemoSign.Positive' | {'SlDemoSign.Zero'} | 'SlDemoSign.Negative'
SampleTime	Sample time	string — {'-1'}
From File (FromFile)		
FileName	File name	string — {'untitled.mat'}
SampleTime	Sample time	string — {'0'}
From Workspace (FromWorkspace)		
VariableName	Data	string — {'simin'}
SampleTime	Sample time	string — {'0'}
Interpolate	Interpolate data	string — 'off' | {'on'}
ZeroCross	Enable zero-crossing detection	string — 'off' | {'on'}
OutputAfterFinalValue	Form output after final data value by	string — {'Extrapolation'} | 'Setting to zero' | 'Holding final value' | 'Cyclic repetition'

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Ground		
In1 (Inport)		
`Port`	Port number	string — `{'1'}`
`IconDisplay`	Icon display	string — `'Signal name' | {'Port number'} | 'Port number and signal name'`
`UseBusObject`	Specify properties via bus object	string — `{'off'} | 'on'`
`BusObject`	Bus object for validating input bus	string — `{'BusObject'}`
`BusOutputAsStruct`	Output as nonvirtual bus	string — `{'off'} | 'on'`
`PortDimensions`	Port dimensions (-1 for inherited)	string — `{'-1'}`
`VarSizeSig`	Variable-size signal	string — `{'Inherit'} | 'No' | 'Yes'`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
`OutMin`	Minimum	string — `{'[]'}`
`OutMax`	Maximum	string — `{'[]'}`
`OutDataTypeStr`	Data type	string — `{'Inherit: auto'} | 'double' | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'boolean' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)' | 'Enum: <class name>'`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SignalType	Signal type	string — {'auto'} | 'real' | 'complex'
SamplingMode	Sampling mode	string — {'auto'} | 'Sample based' | 'Frame based'
LatchByDelaying OutsideSignal	Latch input by delaying outside signal	string — {'off'} | 'on'
LatchByCopying InsideSignal	Latch input by copying inside signal	string — {'off'} | 'on'
Interpolate	Interpolate data	string — 'off' | {'on'}
Pulse Generator (DiscretePulseGenerator)		
PulseType	Pulse type	string — {'Time based'} | 'Sample based'
TimeSource	Time (t)	string — {'Use simulation time'} | 'Use external signal'
Amplitude	Amplitude	string — {'1'}
Period	Period	string — {'10'}
PulseWidth	Pulse width	string — {'5'}
PhaseDelay	Phase delay	string — {'0'}
SampleTime	Sample time	string — {'1'}
VectorParams1D	Interpret vector parameters as 1-D	string — 'off' | {'on'}
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	string — 'off' | {'on'}

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Random Number (`RandomNumber`)		
`Mean`	Mean	string — `{'0'}`
`Variance`	Variance	string — `{'1'}`
`Seed`	Seed	string — `{'0'}`
`SampleTime`	Sample time	string — `{'0.1'}`
`VectorParams1D`	Interpret vector parameters as 1-D	string — `'off' | {'on'}`
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].''}`
`TimeValues`	Vector of time values	string — `{'[0 0.1 0.5 0.6 1].''}`
`LookUpMeth`	Lookup Method	string — `{'Interpolation-Use End Values'} | 'Use Input Nearest' | 'Use Input Below' | 'Use Input Above'`
`tsamp`	Sample time	string — `{'0.01'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataTypeStr`	Output data type	string — `'Inherit: Inherit via back propagation' | {'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`OutputDataTypeScaling Mode`	Deprecated	
`OutDataType`	Deprecated	
`OutScaling`	Deprecated	
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
Repeating Sequence Stair (`Repeating Sequence Stair`) (masked subsystem)		
`OutValues`	Vector of output values	string — `{'[3 1 4 2 1].''}`
`tsamp`	Sample time	string — `{'-1'}`
`OutMin`	Output minimum	string — `{'[]'}`
`OutMax`	Output maximum	string — `{'[]'}`
`OutDataTypeStr`	Output data type	string — `'Inherit: Inherit via back propagation' | {'double'} | 'single' | 'int8' | 'uint8' | 'int16' | 'uint16' | 'int32' | 'uint32' | 'fixdt(1,16)' | 'fixdt(1,16,0)' | 'fixdt(1,16,2^0,0)'`
`OutputDataTypeScaling Mode`	Deprecated	

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`OutDataType`	Deprecated	
`ConRadixGroup`	Deprecated	
`OutScaling`	Deprecated	
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
Signal Builder (`Sigbuilder block`) (masked subsystem)		
Signal Generator (`SignalGenerator`)		
`WaveForm`	Wave form	string — `{'sine'} | 'square' | 'sawtooth' | 'random'`
`TimeSource`	Time (t)	string — `{'Use simulation time'} | 'Use external signal'`
`Amplitude`	Amplitude	string — `{'1'}`
`Frequency`	Frequency	string — `{'1'}`
`Units`	Units	string — `'rad/sec' | {'Hertz'}`
`VectorParams1D`	Interpret vector parameters as 1-D	string — `'off' | {'on'}`
Sine Wave (`Sin`)		
`SineType`	Sine type	string — `{'Time based'} | 'Sample based'`
`TimeSource`	Time (t)	string — `{'Use simulation time'} | 'Use external signal'`
`Amplitude`	Amplitude	string — `{'1'}`
`Bias`	Bias	string — `{'0'}`

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
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	string — 'off' | {'on'}
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	string — 'off' | {'on'}
ZeroCross	Enable zero-crossing detection	string — 'off' | {'on'}
Uniform Random Number (UniformRandomNumber)		
Minimum	Minimum	string — {'-1'}
Maximum	Maximum	string — {'1'}
Seed	Seed	string — {'0'}
SampleTime	Sample time	string — {'0'}
VectorParams1D	Interpret vector parameters as 1-D	string — 'off' | {'on'}

User-Defined Functions Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Embedded MATLAB Fcn (Stateflow) (masked subsystem)		

User-Defined Functions Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Fcn (`Fcn`)		
`Expr`	Expression	string — `{'sin(u(1)*exp(2.3*(-u(2))))'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
Level-2 M-file S-Function (`M-S-Function`)		
`FunctionName`	M-file name	string — `{'mlfile'}`
`Parameters`	Parameters	string — `{''}`
MATLAB Fcn (`MATLABFcn`)		
`MATLABFcn`	MATLAB function	string — `{'sin'}`
`OutputDimensions`	Output dimensions	string — `{'-1'}`
`OutputSignalType`	Output signal type	string — `{'auto'} | 'real' | 'complex'`
`Output1D`	Collapse 2-D results to 1-D	string — `'off' | {'on'}`
`SampleTime`	Sample time (-1 for inherited)	string — `{'-1'}`
S-Function (`S-Function`)		
`FunctionName`	S-function name	string — `{'system'}`
`Parameters`	S-function parameters	string — `{''}`
`SFunctionModules`	S-function modules	string — `{''}`
S-Function Builder (`S-Function Builder`) (masked subsystem)		
`FunctionName`	S-function name	string — `{'system'}`
`Parameters`	S-function parameters	string — `{''}`
`SFunctionModules`	S-function modules	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]'}`
`X0`	Initial condition for state	string — `{'0.0'}`
`InternalDataType`	Data type for internal calculations	string — `{'fixdt('double')'}`
`StateEqScaling`	Scaling for State Equation AX+BU	string — `{'2^0'}`
`OutputEqScaling`	Scaling for Output Equation CX+DU	string — `{'2^0'}`
`LockScale`	Lock output data type setting against changes by the fixed-point tools	string — `{'off'} | 'on'`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`DoSatur`	Saturate to max or min when overflows occur	string — `{'off'} | 'on'`
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'}`

Additional Discrete Block Library Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`DoSatur`	Saturate to max or min when overflows occur	string — `{'off'} | 'on'`
Transfer Fcn Direct Form II Time Varying (`Transfer Fcn Direct Form II Time Varying`) (masked subsystem)		
`vinit`	Initial condition	string — `{'0.0'}`
`RndMeth`	Integer rounding mode	string — `'Ceiling' | 'Convergent' | {'Floor'} | 'Nearest' | 'Round' | 'Simplest' | 'Zero'`
`DoSatur`	Saturate to max or min when overflows occur	string — `{'off'} | 'on'`
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)		

Additional Discrete Block Library Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
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'}
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)		
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

In this section...

About Mask Parameters

This section lists parameters that describe masked blocks. You can use these descriptive parameters with `get_param` and `set_param` to obtain and specify the properties of a block mask.

The descriptive mask parameters listed in this section apply to all masks, and provide access to all mask properties. Be careful not to confuse these descriptive mask parameters with the mask-specific parameters defined for an individual mask in the Mask Editor **Parameters** pane.

See "Working with Block Masks" for information about block masks and the Mask Editor.

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 `{'[]'}`

Mask Parameters (Continued)

Parameter	Description/Prompt	Values
`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 & Ports** pane of the Mask Editor dialog box.	string `{''}`
`MaskEditorHandle`	For internal use only.	
`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 `{''}`
`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 **Block Frame** option on the **Icon & Ports** pane of the Mask Editor dialog box.	`{'on'} | 'off'`

Mask Parameters (Continued)

Parameter	Description/Prompt	Values
`MaskIconOpaque`	Set the transparency of the icon (Opaque is `on`, Transparent is `off`). Set by the **Icon Transparency** option on the **Icon & Ports** 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 **Icon Rotation** option on the **Icon & Ports** pane of the Mask Editor dialog box.	`'on' | {'off'}`
`MaskIconUnits`	Set the units for the drawing commands. Set by the **Icon Units** option on the **Icon & Ports** 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 `{''}`
`MaskNames`	Cell array of mask dialog parameter names. Set inside the **Variable** column in the **Parameters** pane of the Mask Editor dialog box.	matrix `{'[]'}`
`MaskPortRotate`	Specify the port rotation policy for the masked block. Set in the **Port Rotation** area on the **Icon & Ports** pane of the Mask Editor dialog box.	`{'default} | 'physical'`

Mask Parameters (Continued)

Parameter	Description/Prompt	Values
`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 `{''}` See Note 1.
`MaskPropertyNameString`	Pipe-delimited version of **MaskNames**.	string `{''}`
`MaskRunInitForIconRedraw`	For internal use only.	
`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 `{''}` See Note 2.
`MaskTabNameString`	For internal use only.	
`MaskTabNames`	For internal use only.	

Mask Parameters (Continued)

Parameter	Description/Prompt	Values
`MaskToolTipsDisplay`	Determines which mask dialog parameters to display in the data tip for this masked block (see "Block Data Tips" in the Simulink documentation). Specify as a cell array of `'on'` or `'off'` values, each of which indicates whether to display the parameter named at the corresponding position in the cell array returned by `MaskNames`.	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 `{'[]'}`
`MaskTunableValueString`	Comma-delimited string version of `MaskTunableValues`.	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 `{''}` See Note 3.

Mask Parameters (Continued)

Parameter	Description/Prompt	Values
`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 only.	
`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 `{''}` See Note 4.
`MaskVisibilities`	Specifies visibility of parameters. Set with the **Show parameter** check box in the **Options for selected parameter** area on the **Parameters** pane of the Mask Editor dialog box.	matrix `{'[]'}`
`MaskVisibilityString`	Delimited string version of `MaskVisibilities`.	string `{''}`
`MaskWSVariables`	List of the variables defined in the mask workspace (read only).	matrix `{'[]'}`

Notes on Mask Parameter Storage

1 The `MaskPromptString` parameter stores the **Prompt** field values for all mask dialog box parameters as a string, with individual values separated by vertical bars (|), for example:

```
"Slope:|Intercept:"
```

2. The `MaskStyleString` parameter stores the **Type** field values for all mask dialog box parameters as a string, with individual values separated by commas. The **Popup strings** values appear after the `popup` type, as shown in this example:

   ```
   "edit,checkbox,popup(red|blue|green)"
   ```

3. The `MaskValueString` parameter stores the values of all mask dialog box parameters 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:

   ```
   "2|5"
   ```

4. The `MaskVariables` parameter stores the **Variable** field values for all mask dialog box parameters as a string, with individual assignments separated by semicolons. A sequence number indicates the prompt that is associated with a variable. A special character preceding the sequence number indicates whether the parameter value is evaluated or used literally. An at-sign (`@`) indicates evaluation; an ampersand (`&`) indicates literal usage. For example:

   ```
   "a=@1;b=&2;"
   ```

   This string defines two **Variable** field values:

   - The value entered in the first parameter field is evaluated in the MATLAB workspace, and the result is assigned to variable `a` in the mask workspace.
   - The value entered in the second parameter field is not evaluated, but is assigned literally to variable `b` in the mask workspace.

9

Model File Format

Model File Contents

In this section...

Note Model file formatting is a topic for advanced users. The MathWorks does not recommend editing the model file. In the next release, The MathWorks will deprecate this chapter.

About Model File Formats

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.

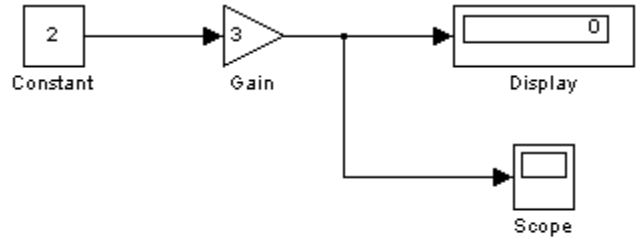
```
Model {
  <Model Parameter Name> <Model Parameter Value>
  ...
  Array {
    Simulink.ConfigSet {
      $ObjectID <Object ID>
      <ConfigSet Parameter Name> <ConfigSet Parameter Value>
      ...
    }
  }
```

```
Simulink.ConfigSet {
  $PropName "ActiveConfigurationSet"
  $ObjectID <Object ID>
}
BlockDefaults {
  <Block Parameter Name> <Block Parameter Value>
  ...
}
BlockParameterDefaults {
  Block {
    <Block Parameter Name> <Block Parameter Value>
    ...
  }
}
AnnotationDefaults {
  <Annotation Parameter Name> <Annotation Parameter Value>
  ...
}
LineDefaults {
  <Line Parameter Name> <Line 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>
    ...
```

```
      }
    }
  }
```

See Chapter 8, “Model and Block Parameters” for descriptions of model and block parameters.

This reference contains examples of each section, extracted from the model file of the following model:




```
This model generates...
```

Model Section

The `Model` section, located at the top of the model file, contains all other sections of the model file and defines the values for model-level parameters. These parameters include the model name, the version of Simulink software last used to modify the model, and configuration set parameters (see “Setting Up Configuration Sets” in the online Simulink documentation) among others.

The following example shows parts of the `Model` section for a model.

```
Model {
  Name                "my_model"
  Version             6.4
  MdlSubVersion       0
  GraphicalInterface {
    NumRootInports          0
    NumRootOutports         0
    ParameterArgumentNames  ""
    ComputedModelVersion    "1.10"
    NumModelReferences      0
```

```
    NumTestPointedSignals   0
  }
  SavedCharacterEncoding  "windows-1252"
  SaveDefaultBlockParams  on
  ...
  Array {
    Type                "Handle"
    Dimension           2
    Simulink.ConfigSet {
      $ObjectID         1
      Version           "1.2.0"
      Array {
        Type            "Handle"
        Dimension       7
        Simulink.SolverCC {
        ...
        }
      ...
      }
    ...
    }
  ...
  }
  ...
}
```

Simulink.ConfigSet Section

The `Simulink.ConfigSet` section appears after the configuration set parameters. This section identifies the active configuration set for the model (see "The Active Set" in the online Simulink documentation).

The following example shows the `Simulink.ConfigSet` section for a model.

```
Simulink.ConfigSet {
  $PropName           "ActiveConfigurationSet"
  $ObjectID           1
}
```

BlockDefaults Section

The `BlockDefaults` section appears after the `Simulink.ConfigSet` section. This section defines the default values for common block parameters in the model. These values can be overridden by individual block parameters, defined in `Block` subsections of `System` sections.

The following example shows the `BlockDefaults` section for a model.

```
BlockDefaults {
  Orientation          "right"
  ForegroundColor      "black"
  BackgroundColor      "white"
  DropShadow           off
  NamePlacement        "normal"
  FontName             "Arial"
  FontSize             10
  FontWeight           "normal"
  FontAngle            "normal"
  ShowName             on
}
```

BlockParameterDefaults Section

The `BlockParameterDefaults` section appears after the `BlockDefaults` section. This section defines the default values for block-specific parameters using `Block` subsections. Each `Block` subsection defines the default parameter-value pairs for a particular type of block in the model. These values can be overridden by individual block parameters, defined in `Block` subsections of `System` sections.

The following example shows part of the `BlockParameterDefaults` section for a model.

```
BlockParameterDefaults {
  Block {
    BlockType          Constant
  }
  Block {
    BlockType          Display
    Format             "short"
    Decimation         "10"
```

```
    Floating           off
    SampleTime         "-1"
  }
  ...
}
```

AnnotationDefaults Section

The `AnnotationDefaults` section appears after the `BlockParameterDefaults` section. This section defines the default parameters for all annotations in the model (see `Simulink.Annotation`).

The following example shows the `AnnotationDefaults` section for a model.

```
AnnotationDefaults {
  HorizontalAlignment     "center"
  VerticalAlignment       "middle"
  ForegroundColor     "black"
  BackgroundColor     "white"
  DropShadow          off
  FontName            "Courier New"
  FontSize            10
  FontWeight          "normal"
  FontAngle           "normal"
}
```

LineDefaults Section

The `LineDefaults` section appears after the `AnnotationDefaults` section. This section defines the default parameters for all lines in the model.

The following example shows the `LineDefaults` section for a model.

```
LineDefaults {
  FontName            "Courier New"
  FontSize            9
  FontWeight          "normal"
  FontAngle           "normal"
}
```

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.

The following example shows parts of the `System` section for a model.

```
System {
  Name                 "my_model"
  Location             [480, 85, 1206, 386]
  Open                 on
  ModelBrowserVisibility  off
  ModelBrowserWidth       200
  ScreenColor          "white"
  PaperOrientation     "landscape"
  ...
  Block {
    BlockType            Constant
    Name                 "Constant"
    Position             [65, 100, 95, 130]
    Value                "2"
    ...
  }
  ...
  Line {
    SrcBlock           "Gain"
    SrcPort            1
    Points             [25, 0]
      Branch {
        Points         [0, 70]
        DstBlock       "Scope"
        DstPort        1
      }
      Branch {
        Points         [20, 0]
        DstBlock       "Display"
        DstPort        1
      }
```

```
  }
  ...




  Annotation {
    Name              "This model generates..."
    Position          [149, 234]
    UseDisplayTextAsClickCallback  off
  }
}
```

10

Model Advisor Checks

Simulink Checks

In this section...

In this section...

Simulink Check Overview

Use the Simulink Model Advisor checks to configure your model for simulation.

See Also

- Consulting Model Advisor in the Simulink documentation.
- Real-Time Workshop Model Advisor Check Reference in the Real-Time Workshop documentation.
- Simulink Verification and Validation Model Advisor Check Reference in the Simulink Verification and Validation documentation.

Check model, local libraries, and referenced models for known upgrade issues

Uses the `slupdate` command analysis mode to check for common upgrade issues.

Description

Check blocks, settings, and references in the model for compatibility issues resulting from using a new version of Simulink software.

Results and Recommended Actions

Condition	Recommended Action
Referenced models recommended for update.	Run Simulink update tool, `slupdate` on the listed models.
Check library update status.	Verify that indicated libraries are valid.
Check update status for the Level 2 API S-functions.	Consider replacing Level 1 S-functions with Level 2.
Blocks have configuration sets or ports with undesired settings.	Run Simulink update tool, `slupdate`, in update mode.

See Also

- `slupdate` in the Simulink documentation.
- *Writing S-Functions* in the Simulink documentation.

Identify unconnected lines, input ports, and output ports

Check for unconnected lines or ports.

Description

This check lists unconnected lines or ports. These can have difficulty propagating signal attributes such as data type, sample time, and dimensions.

Note Ports connected to ground/terminator blocks will pass this test.

Results and Recommended Actions

Condition	Recommended Action
Lines, input ports, or output ports are unconnected.	Ensure all signals are connected. Double-click the list of unconnected items to locate failure.

Tips

Use the `PortConnectivity` command to obtain an array of structures describing block input or output ports.

See Also

"Common Block Parameters" on page 8-87 in the Simulink documentation for information on the `PortConnectivity` command.

Check root model Inport block specifications

Check that root model Inport blocks fully define dimensions, sample time, and data type.

Description

Using root model Inport blocks that do not fully define dimensions, sample time, or data type can lead to undesired simulation results. Simulink software back-propagates dimensions, sample times and data types from downstream blocks unless you explicitly assign them values.

Results and Recommended Actions

Condition	Recommended Action
Root-level Inport blocks have undefined attributes.	Fully define the attributes of all root-level Inport blocks.

See Also

- “Working with Data Types” in the Simulink documentation.
- “Determining Output Signal Dimensions” in the Simulink documentation.
- “How to Specify the Sample Time” in the Simulink documentation.

Check optimization settings

Check for optimizations that can lead to nonoptimal code generation and simulation.

Description

This check reviews the status of optimizations that can improve code efficiency and simulation time.

Results and Recommended Actions

Condition	Recommended Action
The specified optimizations are off.	Select the following optimization check boxes in the Optimization pane of the Configuration Parameters dialog box: • **"Block reduction"** • **"Conditional input branch execution"** • **"Inline parameters"** • **"Implement logic signals as Boolean data (vs. double)"** • **"Enable local block outputs"** • **"Reuse block outputs"** • **"Eliminate superfluous local variables (Expression folding)"** • **"Use memset to initialize floats and doubles to 0.0"** • **"Remove code from floating-point to integer conversions that wraps out-of-range values"** • **"Inline invariant signals"** • **"Remove root level I/O zero initialization"** • **"Remove internal data zero initialization"** • **"Use bitsets for storing state configuration"**

Condition	Recommended Action
	• **"Use bitsets for storing Boolean data"**
"Application lifespan (days)" is set as infinite. This could lead to expensive 64-bit counter usage.	Choose appropriate stop time if this is not intended.
The specified diagnostics, which can increase the time it takes to simulate your model, are set to `warning` or `error`.	Select `none` for: • **Diagnostics > Solver > Solver data inconsistency** • **Diagnostics > Data Validity > Array bounds exceeded** • **Diagnostics > Data Validity > Simulation range checking**
The specified Real-Time Workshop Embedded Coder parameters are off.	If you have a Real-Time Workshop Embedded Coder license, and you are using an ERT-based system target file, select the following check boxes: • **Real-Time Workshop > Interface > "Single output/update function"** • **Real-Time Workshop > General > "Ignore test point signals"** • **Optimization > "Pass reusable subsystem outputs as"**

Tips

If the system contains Model blocks and the referenced model is in Accelerator mode, simulating the model requires generating and compiling code.

See Also

- "Optimization Pane" in the Simulink documentation.
- "Optimizing Generated Code" in the Real-Time Workshop documentation.
- "Preparing Models for Code Generation" in the Real-Time Workshop Embedded Coder documentation.

Check for parameter tunability information ignored for referenced models

Checks if parameter tunability information is included in the Model Parameter Configuration dialog box.

Description

Simulink software ignores tunability information specified in the Model Parameter Configuration dialog box. This check identifies those models containing parameter tunability information that Simulink software will ignore if the model is referenced by other models.

Results and Recommended Actions

Condition	Recommended Action
Model contains ignored parameter tunability information.	Click the links to convert to equivalent Simulink parameter objects in the MATLAB workspace.

See Also

"Parameter Considerations" in the Simulink documentation.

Check for implicit signal resolution

Identify models that attempt to resolve named signals and states to `Simulink.Signal` objects.

Description

Requiring Simulink software to resolve all named signals and states is inefficient and slows incremental code generation and model reference. This check identifies those signals and states for which you may turn off implicit signal resolution and enforce resolution.

Results and Recommended Actions

Condition	Recommended Action
Not all signals and states are resolved.	Turn off implicit signal resolution and enforce resolution for each signal and state that successfully resolves.

See Also

"Resolving Signal Objects for Output Data" in the Simulink documentation.

Check for optimal bus virtuality

Identify virtual buses that could be made nonvirtual. Making these buses nonvirtual improves generated code efficiency.

Description

This check identifies blocks incorporating virtual buses that cross a model boundary. Changing these to nonvirtual improves generated code efficiency.

Results and Recommended Actions

Condition	Recommended Action
Blocks that specify a virtual bus crossing a model boundary.	Change the highlighted bus to nonvirtual.

See Also

- "Working with Signals" in the Simulink documentation.
- "Virtual and Nonvirtual Buses" in the Simulink documentation.

Check for Discrete-Time Integrator blocks with initial condition uncertainty

Identify Discrete-Time Integrator blocks with state ports and initial condition ports that are fed by neither an Initial Condition nor a Constant block.

Description

Discrete-Time Integrator blocks with state port and initial condition ports might not be properly initialized unless they are fed from an Initial Condition or Constant block. This is more likely to happen when Discrete-Time Integrator blocks are used to model second-order or higher-order dynamic systems.

Results and Recommended Actions

Condition	Recommended Action
Discrete-Time Integrator blocks are not initialized during the model initialization phase.	Add a Constant or Initial Condition block to feed the external Initial Condition port.

See Also

- IC block
- Discrete-Time Integrator block
- Enumerated Constant block

Identify disabled library links

Search model for disabled library links.

Description

Disabled library links can cause unexpected simulation results. All disabled links should be resolved before a model is saved.

Note This check may overlap with "Check model, local libraries, and referenced models for known upgrade issues" on page 10-5.

Results and Recommended Actions

Condition	Recommended Action
Library links are disabled.	Use `Restore Link` from the `Link Options` setting in the context menu.

Tips

- Use the Model Browser to find library links.
- To enable a broken link, right-click a block in your model to display the context menu. Choose `Link Options` and click `Restore Link`.

See Also

"The Model Browser" in the Simulink documentation.

Identify parameterized library links

Search model for parameterized library links.

Description

Parameterized library links that are unintentional can result in unexpected parameter settings in your model. This can result in improper model operation.

Results and Recommended Actions

Condition	Recommended Action
Parameterized links are listed.	Verify that all parameterized links are intended.

Tips

- Right-click a block in your model to display the context menu. Choose `Link Options` and click `Go To Library Block` to see the original block from the library.
- To parameterize a library link, choose `Look Under Mask`, from the context menu and select the parameter.

See Also

"Working with Block Masks" in the Simulink documentation.

Identify unresolved library links

Search the model for unresolved library links, where the specified library block cannot be found.

Description

Check for unresolved library links. Models do not simulate while there are unresolved library links.

Results and Recommended Actions

Condition	Recommended Action
Library links are unresolved.	Locate missing library block or an alternative.

See Also

"Fixing Unresolved Library Links"

Check for proper usage of Data Store Memory blocks

Look for modeling issues related to Data Store Memory blocks.

Description

Checks shadowing of data stores of higher scope, strong typing, multitasking data integrity, and runtime diagnostics.

Results and Recommended Actions

Condition	Recommended Action
Data Store blocks have duplicate names.	Give each Data Store block in the model a unique name.
Data Store blocks are not strongly typed.	Specify a data type other than `Inherit: auto`.
Multiple tasks access the same data store.	Change the model to avoid multitask data stores.
Reading and writing occur out of order.	Restructure and prioritize so enforce correct order.
Multiple writes occur within a single time step.	Change the model to write data only once per step.

See Also

- “Working with Data Stores”
- “Storage Classes for Data Store Memory Blocks”
- Data Store Memory
- Data Store Read
- Data Store Write
- “Duplicate data store names”
- “Multitask data store”
- “Detect read before write”

- “Detect write after read”
- “Detect write after write”

Check for proper bus usage

Identify Mux blocks used as a bus creator and any bus signal that is treated as a vector.

Description

Models should not contain bus signals that Simulink software implicitly converts to vectors. Instead, either insert a Bus to Vector conversion block between the bus signal and the block input port that it feeds, or use the `Simulink.BlockDiagram.addBusToVector` command.

Note You should always run this check before running Check consistency of initialization parameters for Outport and Merge blocks.

Results and Recommended Actions

Condition	Recommended Action
Identify signals used as vectors.	In the Configuration Parameters dialog box, set Mux blocks used to create bus signals to `error`.
Model uses buses properly.	In the `Configuration Parameters` dialog set `Bus signal treated as vector` to `error`.
Bus signals are implicitly converted to vectors.	Use `Simulink.BlockDiagram.addBusToVector` or insert a Bus to Vector block.

Tips

The Bus to Vector conversion block is located in the Simulink/Signal Attributes library.

See Also

- Bus to Vector block
- "Mux blocks used to create bus signals"

- “Bus signal treated as vector”
- “Check consistency of initialization parameters for Outport and Merge blocks” on page 10-24
- in the Simulink documentation.

Check for potentially delayed function-call subsystem return values

Identify function-call return values that might be delayed because Simulink software inserted an implicit Signal Conversion block.

Description

To ensure that signals reside in contiguous memory, Simulink software can automatically insert an implicit Signal Conversion block in front of function-call initiator block input ports. This can result in a one-step delay in returning signal values from calling function-call subsystems. The delay can be avoided by ensuring the signal originates from a signal block within the function-call system. Or, if the delay is acceptable, insert a Unit Delay block in front of the affected input ports.

Results and Recommended Actions

Condition	Recommended Action
The listed block input ports could have an implicit Signal Conversion block.	Decide if a one-step delay in returning signal values is acceptable for the listed signals. • If the delay is not acceptable, rework your model so that the input signal originates from within the calling subsystem. • If the delay is acceptable, insert a Unit Delay block in front of each listed input port.

See Also

Signal Conversion block

Unit Delay block

Identify block output signals with continuous sample time and non-floating point data type

Find continuous sample time, non-floating-point output signals.

Description

Non-floating-point signals cannot properly represent continuous variables.

Results and Recommended Actions

Condition	Recommended Action
Signals with continuous sample times have a non-floating-point data type.	On the identified signals, either change the sample time to be discrete or fixed-in-minor-step ([0 1]).

See Also

"Working with Sample Times" in the Simulink documentation.

Check for proper Merge block usage

Analyze Merge blocks in the same tree as a group, and determine the possibility for them to execute at the same time step.

Description

Blocks that directly drive the same tree of Merge blocks should have mutually exclusive execution in each time step. This check identifies those blocks that drive the same tree of Merge blocks, and so are likely to execute at the same time step.

Input Parameters

Maximum analysis time (seconds)
Provide a maximum analysis time to execute the check.

Results and Recommended Actions

Condition	Recommended Action
Merge blocks can be interconnected to form a tree structure.	Rework your model so that no blocks drive the same tree of Merge blocks.

See Also

- Merge block
- “Check consistency of initialization parameters for Outport and Merge blocks” on page 10-24

Check consistency of initialization parameters for Outport and Merge blocks

Identify Outport and Merge blocks with parameter settings that can lead to unexpected initialization behavior, and migrate your model to simplified initialization mode.

Description

In R2008b or later, you can choose simplified initialization mode for conditionally executed subsystems, Merge blocks, subsystem elapsed time, and Discrete-Time Integrator blocks. Simplified initialization mode improves the consistency of simulation results, especially for models that do not specify initial conditions for conditionally executed subsystem output ports, and for models that have conditionally executed subsystem output ports connected to S-functions.

This Model Advisor check identifies settings in your model that can cause problems when using simplified initialization mode. There are two types of messages in this check, *errors* and *warnings*. Error messages identify issues that you need to address manually before the model can be migrated to simplified initialization mode. Warning messages identify issues or changes in behavior that may occur after migration.

Note Before running this check, run the check: **Check for proper bus usage** to ensure that Merge blocks are used correctly, and enable strict Merge run-time diagnostics.

After running this check, select the **Explore Result** button to access the blocks listed in each message, with the exception of library-linked blocks.

Once you have addressed all issues identified by the check, select **Proceed** to migrate your model to the simplified initialization mode.

Note Because it is difficult to undo these changes, save a backup copy of your model and libraries before migrating to simplified initialization mode.

Results and Recommended Actions

Condition	Recommended Action
Error: Strict Bus Mode Must Be Enabled	**1** Run **Check for proper bus usage** in the Model Advisor to ensure that Mux blocks are used correctly in the model. **2** In the model window, select **Simulation > Configuration Parameters > Diagnostics > Connectivity**. **3** Set **Mux blocks used to create bus signals** to `error`. **4** Set **Bus signal treated as vector** to `error`.
Error: Strict Merge Run-Time Diagnostics Must Be Enabled	In the model window: **1** Select **Simulation > Configuration Parameters > Diagnostics > Data Validity**. **2** Set **Detect multiple driving blocks executing at the same time step** to `error`. **3** Ensure that the model simulates without errors before running this check again.
Error: Referenced Models Not Yet Migrated	Migrate the model referenced by the Model block to simplified initialization mode, then migrate the top model.

Condition	Recommended Action
Error: Single-Input Merge Blocks	Replace both the Mux block used to produce the input signal and the Merge block with one multi-input Merge block. Single-input Merge blocks are not supported in simplified initialization mode.
Error: Root Merge Blocks With Unspecified Initial Output	Specify an explicit value for the **Initial output** parameter of the *root Merge block*. A root Merge block is any Merge block with an output port that does not connect to another Merge block.
Error: Merge Blocks With Non-zero Input Port Offsets	Deselect the **Allow unequal port widths** parameter of the Merge block. **Note** Consider using Merge blocks only for signal elements that require true merging. Other elements can be combined with merged elements using the Concatenate block.
Error: Merge Blocks With Unconnected Inputs or Inputs from Non-Conditionally Executed Subsystem	Set the **Number of inputs** parameter of the Merge block to the correct number of inputs, so that each input is connected to a signal. Ensure that the Merge block inputs are each driven by a conditionally executed subsystem. Merge blocks cannot be driven directly by an Iterator Subsystem or any block that is not a conditionally executed subsystem.

Condition	Recommended Action
Error: Merge Blocks With Inputs That Have Been Combined Or Reordered Outside of Conditionally Execute Subsystems	Ensure that any combination or reordering of Merge block input signals (using Mux, Bus Creator, or Selector blocks, for example) takes place within a conditionally executed subsystem.
Error: Merge Blocks With Inconsistent Input Sample Times	Ensure that each input signal has the same **Sample time**.
Error: Merge Blocks With Multiple Input Ports Driven By Same Source	Ensure that the Merge block does not have multiple input signals that are driven by the same conditionally executed subsystem or conditionally executed Model block.
Error: Outport Blocks With Conflicting Buffer Requirements	The Outport block has function-call trigger or function-call data dependency signal passing through it, along with regular signals. Some of the regular data signals require an explicit signal buffer to ensure correct initialization of the corresponding subsystem's output signal. However, buffering function-call related signals will lead to a function-call data dependency violation. Consider modifying the model to pass function-call related signals through a separate Outport. For examples of function-call data dependency violations, see the demo model `sl_subsys_semantics`.

Condition	Recommended Action
Error: Outports Requiring Explicit Bus Copy	An explicit copy of the bus signal driving the Outport block is needed to ensure correct initialization of the corresponding subsystem's output signal. Insert a Signal Conversion block before the Outport block, then set the **Output** parameter of the Signal Conversion block to `Bus copy`.
Error: Outport Blocks Being Merged But Can Reset When Disabled	Set the **Output when disabled** parameter of the Outport block to `held`. For more information, see the Outport block documentation.
Error: Outport Blocks With Unspecified Initial Output	Specify the **Initial output** parameter for the Outport block, unless you want the block to obtain its initial output value from its input signal. If you want the Outport block to obtain its initial output value from its input signal, ensure that all of its sources are valid initial output value sources. Outport blocks can inherit initial output value from Constant, Initial Condition, Merge (with initial output), or conditionally executed subsystem blocks.

Condition	Recommended Action
	Note If you are working with a conditionally executed subsystem Outport block that is driven directly by another conditionally executed subsystem Outport block, you should resolve this error message for the inner Outport block first, then run this Model Advisor check again to see if the problem is resolved for the outer Outport block.
Error: Outport Blocks With Automatic Rate Transitions	Simulink has inserted Rate Transition blocks on the input signals of the Outport block. If such rate transitions are needed, specify the **Initial output** parameter for each Outport block. Otherwise, perform the following procedure: **1** Select **Simulation > Configuration Parameters > Solver**. **2** Deselect the option **Automatically handle rate transition for data transfer**. **3** Run this Model Advisor check again.

Condition	Recommended Action
Error: Outport Blocks Require Explicit Initial Output	Specify the **Initial output** parameter for the Outport block. The Outport block cannot obtain initial output values from its input signals due to Simulink internal signal storage handling.
Error: Outport Blocks Reset When Disabled With Unspecified Initial Output	Specify the **Initial output** parameter of the Outport block. You must specify the initial output value for blocks that are configured to reset when disabled.
Error: Outport Blocks Passing Through Function-call Data Dependency Signal Must Not Have Initial Output Value	You cannot specify an Initial output value for the Outport block because function-call data dependency signals are passing through it. To set the initial output value: **1** Set the **Initial output** parameter of the Outport block to `[]`. **2** Provide the initial value at the source of the data dependency signal rather than at the Outport block.

Condition	Recommended Action
Error: Blocks Requiring Elapsed Timer Will Not Be Allowed In Iterator Subsystem	Within an Iterator subsystem hierarchy, do not use blocks that require a service that maintains the time that has elapsed between two consecutive executions.Since Iterator subsystem can be executed multiple times at a given time step, the concept of elapsed time is not well-defined between two such executions. Using these blocks inside an Iterator subsystem can cause unexpected behavior.
Warning: Outport Blocks Initial Output Value Can No Longer Be Specified By Signal Objects	Ensure that the following behavior is acceptable. The **Initial output** parameter of the Outport block cannot be specified by signal objects. You can initialize the input or output signals for an Outport block using signal objects, but the initialization results may be overwritten by that from the Outport block.
Warning: Outport Blocks Being Merged But Is Unconnected Or Connected To Ground Block	Ensure that the following behavior is acceptable. The Outport block is driving a Merge block, but its inputs are either unconnected or connected to Ground blocks. In classic initialization mode, unconnected or grounded outports do not update the merge signal even when their parent conditionally executed subsystems are executing. In simplified initialization mode, however, these outports will update the merge signal with zero value when their parent conditionally executed subsystems are executing.

Condition	Recommended Action
Warning: Merge Blocks Initial Output Value Can No Longer Be Specified By Signal Objects	Ensure that the following behavior is acceptable. The **Initial output** parameter of the Merge block cannot be specified by signal objects. You can initialize the input or output signals for a Merge block using signal objects, but the initialization results may be overwritten by that from the Merge block.
Warning: Outport Blocks Will Obtain Initial Output Value from Input Signal When Migrated	Ensure that the following behavior is acceptable. The **Initial output** parameter of the Outport block is not specified. Simplified initialization mode will assume that the initial output value for the Outport block is derived from the input signal, which may result in different initialization behavior.
Warning: Outer Outport Blocks With Explicit Initial Output	Ensure that the following behavior is acceptable. The **Initial output** and **Output when disabled** parameters of the Outport block are currently required to match those of their source Outport blocks. This constraint is not required in simplified initialization mode. However, this flexibility means that nested Outport blocks with explicit initial output values cannot share signal storage. You can regain this signal storage efficiency by setting **Initial output** to `[ ]` (empty matrix), and **Output**

Condition	Recommended Action
	when disabled to `held`, so that the Outports derive their initial output value, and can therefore share signal storage with their input signals.
Warning: Conditionally Executed Subsystems Propagating Execution Context Across Output Boundary	Ensure that the following behavior is acceptable. **Propagate execution context across subsystem boundary** is selected for the subsystem. Propagating execution context across an output boundary can cause unpredictable initialization behavior, and therefore is not supported in simplified initialization mode. Execution context will still be propagated across input boundaries.
Warning: Initialization Behavior Changes for Discrete-Time Integrators	Ensure that the following behavior is acceptable. The initialization behavior of the Discrete-Time Integrator block has been changed to be more consistent and robust in simplified initialization mode. These changes include: • Elimination of the **Use initial condition as initial and reset value for** parameter • Application of the **Initial condition** parameter only to the integrator output. • Simplified enable/disable behavior See the Discrete-Time Integrator block documentation for more

Condition	Recommended Action
	information on simplified Discrete-Time Integrator initialization.
Warning: Blocks Will Not Be Allowed To Read Input From Conditionally Executed Subsystems During Initialization	Ensure that the following behavior is acceptable. Some blocks, such as the Discrete-Time Integrator block, currently read their inputs from conditionally executed subsystems during initialization. This is done as an optimization. This optimization is not allowed in simplified initialization mode because the output of a conditionally executed subsystem at the first time step after initialization may be different than the initial value declared in the corresponding Outport block. In particular, this occurs if the subsystem is active at the first time step.
Error: Library Blocks With Instances That Cannot Be Migrated	Examine the error messages for each block to determine a proper corrective action.
Warning: Library Blocks With Instances That Have Warning Messages	Examine the warning messages for each block before migrating.
Message: Outport Blocks Will Use Dialog Initial Output Value After Migration	No action required. The Outport block will maintain its current settings and use its specified **Initial output** value.

Condition	Recommended Action
Message: Outport Blocks Will Use Initial Output Value From Input Signal After Migration	No action required. The Outport block currently specifies an **Initial output** of `[]` (empty matrix), and **Output when disabled** as `held`. This means that each outport does not perform initialization, but implicitly relies on source blocks to initialize its input signal. After migration, the parameter **Source of initial output value** will be set to `Input signal` to reflect this behavior.
Message: Outport Blocks To Use SimEvents Initialization Semantics	No action required. The Outport block will continue to use an **Initial output** of `[]` (empty matrix), and **Output when disabled** as `held`, because their parent conditionally executed subsystems are connected to SimEvents blocks.

See Also

- Consulting Model Advisor
- Simulink Checks
- “Check for proper bus usage” on page 10-19
- Merge block
- Discrete-Time Integrator block
- Outport block
- “Mux blocks used to create bus signals”
- “Bus signal treated as vector”

- "Detect multiple driving blocks executing at the same time step"
- "Underspecified initialization detection"

Check sample times of Data Store blocks

Identify Data Store blocks whose sample times cause modeling errors.

Description

Multitasking Data Store blocks often indicate a modeling problem using Data Store blocks.

Results and Recommended Actions

Condition	Recommended Action
Data Store blocks in your model have continuous or zero-order-hold sample times.	Consider making the listed blocks discrete or replacing them with Memory or Goto and From blocks.
The compile time diagnostic **Multitask data store** is set to `none`.	Consider setting **Multitask data store** to `warning` or `error` in the **Configuration Parameters > Diagnostics > Data Validity** pane.

See Also

- "Working with Data Stores".
- Data Store Memory
- Data Store Read
- Data Store Write

Check for non-continuous signals driving derivative ports

Identify noncontinuous signals that drive derivative ports.

Description

Noncontinuous signals that drive derivative ports cause the solver to reset every time the signal changes value, which slows down simulation.

Results and Recommended Actions

Condition	Recommended Action
There are noncontinuous signals in the model driving derivative ports.	• Make the specified signals continuous. • Replace the continuous blocks receiving these signals with discrete state versions of the blocks.

See Also

"How Simulink Works" in the Simulink documentation.

Runtime diagnostics for Data Store blocks

Verify that read and write order checking is on if there are Data Store blocks in the model.

Description

Results are unpredictable if read and write order checking is off.

Results and Recommended Actions

Condition	Recommended Action
Detect read before write checking is disabled.	Consider enabling **Detect read before write** in the **Configuration Parameters > Diagnostics > Data Validity** pane.
Detect write after read checking is disabled.	Consider enabling **Detect write after read** in the **Configuration Parameters > Diagnostics > Data Validity** pane.
Detect write after write checking is disabled.	Consider enabling **Detect write after write** in the **Configuration Parameters > Diagnostics > Data Validity** pane.

Tips

The runtime diagnostics might slow down simulations considerably. Set them to `Disable all` once you have verified that Simulink does not generate any warnings or errors during simulation.

See Also

- "Working with Data Stores"
- Data Store Memory
- Data Store Read
- Data Store Write

Runtime diagnostics for S-functions

Check array bounds and solver consistency if S-Function blocks are in the model.

Description

Validates whether S-Function blocks adhere to the ODE solver consistency rules that Simulink applies to its built-in blocks.

Results and Recommended Actions

Condition	Recommended Action
Solver data inconsistency is set to `none`.	Set **Solver data inconsistency** in the **Configuration Parameters > Diagnostics** pane to `warning` or `error`.
Array bounds exceeded is set to `none`.	Set **Array bounds exceeded** in the **Configuration Parameters > Diagnostics > Data Validity** pane to `warning` or `error`

See Also

Writing S-Functions in the Simulink documentation

Simulink Limits

The following table documents some limits on the size and complexity of Simulink models.

Model Feature	Limit
Maximum number of levels in a block diagram	1024
Maximum number of branches in a line	1024
Maximum length of a parameter name	256
Maximum length of a parameter string value	32768
Maximum value of a model window coordinate	32768
Maximum number of bytes of logged simulation data	2^31-1 bytes on 32-bit systems, 2^48-1 bytes on 64-bit systems
Maximum length of integer and fixed-point data types	128 bits

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z