

# Simulink®

**Simulation and Model-Based Design**

■ Modeling

■ Simulation

■ Implementation

## How to Contact The MathWorks:



[www.mathworks.com](http://www.mathworks.com)  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab)

Web  
Newsgroup



[support@mathworks.com](mailto:support@mathworks.com)  
[suggest@mathworks.com](mailto:suggest@mathworks.com)  
[bugs@mathworks.com](mailto:bugs@mathworks.com)  
[doc@mathworks.com](mailto:doc@mathworks.com)  
[service@mathworks.com](mailto:service@mathworks.com)  
[info@mathworks.com](mailto:info@mathworks.com)

Technical support  
Product enhancement suggestions  
Bug reports  
Documentation error reports  
Order status, license renewals, passcodes  
Sales, pricing, and general information



508-647-7000

Phone



508-647-7001

Fax



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

Mail

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

### *Simulink Reference*

© COPYRIGHT 2002 - 2004 by The MathWorks, Inc.

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

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

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

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

Printing History:	July 2002	Online only	New 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)

## Simulink Blocks

**1**

**Blocks — Categorical List . . . . . 1-4**

**Blocks — Alphabetical List . . . . . 1-12**

## Linearization and Trimming Commands

**2**

**Linearization and Trimming Commands . . . . . 2-2**

## Model Construction Commands

**3**

**Model Construction Commands . . . . . 3-6**

## Simulation Commands

**4**

**Simulation Commands . . . . . 4-3**

## Mask Icon Drawing Commands

**5**

**Mask Icon Drawing Commands . . . . . 5-5**

	<b>Simulink Debugger Commands</b>	
<b>6</b>		
	Simulink Debugger Commands .....	6-5

	<b>Data Type Functions</b>	
<b>7</b>		
	Data Type Functions .....	7-2

	<b>Data Object Classes</b>	
<b>8</b>		
	Classes — Alphabetical List .....	8-3

	<b>Model and Block Parameters</b>	
<b>9</b>		

	<b>Model File Format</b>	
<b>10</b>		

	<b>Embedded MATLAB Function Block Reference</b>	
<b>11</b>		



# Simulink Blocks

---

## What Each Block Reference Page Contains

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

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

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

```
showblockdatatypetable
```

## Blocks — Categorical List

“Commonly Used”  
“Continuous”  
“Discontinuities”  
“Discrete”  
“Logic and Bit Operations”  
“Lookup Tables”  
“Math Operations”  
“Model Verification”  
“Model-Wide Utilities”  
“Ports & Subsystems”  
“Signal Attributes”  
“Signal Routing”  
“Sinks”  
“Sources”  
“User-Defined Functions”  
“Additional Math & Discrete”

### Commonly Used

Bus Creator  
Bus Selector  
Constant  
Data Type Conversion  
Demux  
Discrete-Time Integrator  
Gain  
Ground  
Inport  
Integrator  
Logical Operator  
Mux  
Outport  
Product  
Relational Operator  
Saturation  
Scope, Floating Scope, Signal Viewer Scope  
Subsystem, Atomic Subsystem, CodeReuse Subsystem  
Sum

Switch  
Terminator  
Unit Delay

## **Continuous**

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

## **Discontinuities**

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

## **Discrete**

Difference  
Discrete Derivative  
Discrete Filter  
Discrete State-Space  
Discrete Transfer Fcn  
Discrete Zero-Pole  
Discrete-Time Integrator  
First-Order Hold  
Integer Delay

Memory  
Tapped Delay  
Transfer Fcn First Order  
Transfer Fcn Lead or Lag  
Transfer Fcn Real Zero  
Unit Delay  
Weighted Moving Average  
Zero-Order Hold

## **Logic and Bit Operations**

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

## **Lookup Tables**

Cosine  
Direct Lookup Table (n-D)  
Interpolation (n-D) Using PreLookup  
Lookup Table  
Lookup Table (2-D)  
Lookup Table (n-D)  
Lookup Table Dynamic

Prelookup Index Search  
Sine

## **Math Operations**

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

## **Model Verification**

Assertion  
Check Discrete Gradient  
Check Dynamic Gap

Check Dynamic Lower Bound  
Check Dynamic Range  
Check Dynamic Upper Bound  
Check Input Resolution  
Check Static Gap  
Check Static Lower Bound  
Check Static Range  
Check Static Upper Bound

## **Model-Wide Utilities**

DocBlock  
Model Info  
Time-Based Linearization  
Trigger-Based Linearization

## **Ports & Subsystems**

Configurable Subsystem  
Enable  
Enabled and Triggered Subsystem  
Enabled Subsystem  
For Iterator Subsystem  
Function-Call Generator  
Function-Call Subsystem  
If  
If Action Subsystem  
Inport  
Model  
Outport  
Subtract  
Switch Case  
Switch Case Action Subsystem  
Trigger  
Triggered Subsystem  
While Iterator Subsystem



## Signal Attributes

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

## Signal Routing

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

## Sinks

- Display
- Outport
- Scope, Floating Scope, Signal Viewer Scope
- Stop Simulation

Terminator  
To File  
To Workspace  
XY Graph

## **Sources**

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

## **User-Defined Functions**

Embedded MATLAB Function  
Fcn  
MATLAB Fcn  
M-File S-Function  
S-Function  
S-Function Builder

## **Additional Math & Discrete**

### **Additional Discrete**

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

### **Additional Math**

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

## Blocks – Alphabetical List

Abs  
Action Port  
Add  
Algebraic Constraint  
Assertion  
Assignment  
Backlash  
Band-Limited White Noise  
Bias  
Bit Clear  
Bit Set  
Bitwise Operator  
Bus Creator  
Bus Selector  
Check Discrete Gradient  
Check Dynamic Gap  
Check Dynamic Lower Bound  
Check Dynamic Range  
Check Dynamic Upper Bound  
Check Input Resolution  
Check Static Gap  
Check Static Lower Bound  
Check Static Range  
Check Static Upper Bound  
Chirp Signal  
Clock  
Combinatorial Logic  
Compare To Constant  
Compare To Zero  
Complex to Magnitude-Angle  
Complex to Real-Imag  
Configurable Subsystem  
Constant  
Cosine  
Coulomb and Viscous Friction  
Counter Free-Running  
Counter Limited  
Data Store Memory

Data Store Read  
Data Store Write  
Data Type Conversion  
Data Type Conversion Inherited  
Data Type Duplicate  
Data Type Propagation  
Data Type Scaling Strip  
Dead Zone  
Dead Zone Dynamic  
Decrement Real World  
Decrement Stored Integer  
Decrement Time To Zero  
Decrement To Zero  
Demux  
Derivative  
Detect Change  
Detect Decrease  
Detect Fall Negative  
Detect Fall Nonpositive  
Detect Increase  
Detect Rise Nonnegative  
Detect Rise Positive  
Difference  
Digital Clock  
Direct Lookup Table (n-D)  
Discrete Derivative  
Discrete Filter  
Discrete State-Space  
Discrete-Time Integrator  
Discrete Transfer Fcn  
Discrete Zero-Pole  
Display  
Divide  
DocBlock  
Dot Product  
Embedded MATLAB Function  
Enable  
Enabled and Triggered Subsystem  
Enabled Subsystem  
Environment Controller

Extract Bits  
Fcn  
First-Order Hold  
Fixed-Point State-Space  
For Iterator  
For Iterator Subsystem  
From  
From File  
From Workspace  
Function-Call Generator  
Function-Call Subsystem  
Gain  
Goto  
Goto Tag Visibility  
Ground  
Hit Crossing  
IC  
If  
If Action Subsystem  
Increment Real World  
Increment Stored Integer  
Index Vector  
Inport  
Integer Delay  
Integrator  
Interpolation (n-D) Using PreLookup  
Interval Test  
Interval Test Dynamic  
Logical Operator  
Lookup Table  
Lookup Table (2-D)  
Lookup Table (n-D)  
Lookup Table Dynamic  
Magnitude-Angle to Complex  
Manual Switch  
Math Function  
MATLAB Fcn  
Matrix Concatenation  
Memory  
Merge

M-File S-Function  
MinMax  
MinMax Running Resettable  
Model  
Model Info  
Multiport Switch  
Mux  
Outport  
Polynomial  
Prelookup Index Search  
Product  
Product of Elements  
Probe  
Pulse Generator  
Quantizer  
Ramp  
Random Number  
Rate Limiter  
Rate Limiter Dynamic  
Rate Transition  
Real-Imag to Complex  
Relational Operator  
Relay  
Repeating Sequence  
Repeating Sequence Interpolated  
Repeating Sequence Stair  
Reshape  
Rounding Function  
Saturation  
Saturation Dynamic  
Scope, Floating Scope, Signal Viewer Scope  
Selector  
S-Function  
S-Function Builder  
Shift Arithmetic  
Sign  
Signal Builder  
Signal Conversion  
Signal Generator  
Signal Specification

Sine  
Sine Wave  
Sine Wave Function  
Slider Gain  
State-Space  
Step  
Stop Simulation  
Subsystem, Atomic Subsystem, CodeReuse Subsystem  
Subtract  
Sum  
Sum of Elements  
Switch  
Switch Case  
Switch Case Action Subsystem  
Tapped Delay  
Terminator  
Time-Based Linearization  
To File  
To Workspace  
Transfer Fcn  
Transfer Fcn Direct Form II  
Transfer Fcn Direct Form II Time Varying  
Transfer Fcn First Order  
Transfer Fcn Lead or Lag  
Transfer Fcn Real Zero  
Transport Delay  
Trigger  
Trigger-Based Linearization  
Triggered Subsystem  
Trigonometric Function  
Unary Minus  
Uniform Random Number  
Unit Delay  
Unit Delay Enabled  
Unit Delay Enabled External IC  
Unit Delay Enabled Resettable  
Unit Delay Enabled Resettable External IC  
Unit Delay External IC  
Unit Delay Resettable  
Unit Delay Resettable External IC



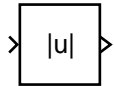
Unit Delay With Preview Enabled  
Unit Delay With Preview Enabled Resettable  
Unit Delay With Preview Enabled Resettable External RV  
Unit Delay With Preview Resettable  
Unit Delay With Preview Resettable External RV  
Variable Transport Delay  
Weighted Moving Average  
Weighted Sample Time  
Weighted Sample Time Math  
While Iterator  
While Iterator Subsystem  
Width  
Wrap To Zero  
XY Graph  
Zero-Order Hold  
Zero-Pole



**Purpose** Output the absolute value of the input

**Library** Math Operations

**Description** The Abs block outputs the absolute value of the input.



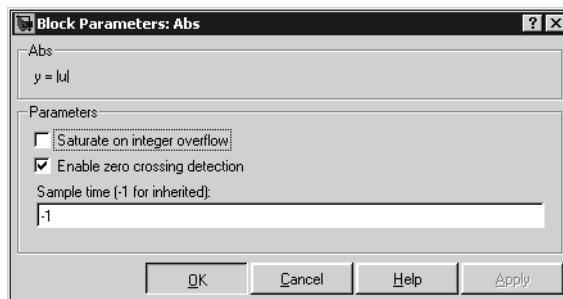
For signed data types, the absolute value of the most negative value is problematic since it is not representable by the data type. In this case, the behavior of the block is controlled by the **Saturate on integer overflow** check box. If checked, the absolute value of the data type saturates to the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

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

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

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

## Parameters and Dialog Box



**Saturate on integer overflow**

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

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

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

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

**Enable zero crossing detection**

Select to enable zero crossing detection. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

**Sample time (-1 for inherited)**

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

**Characteristics**

Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Dimensionalized	Yes
Zero Crossing	No, unless you select <b>Enable zero crossing detection</b>

## Purpose

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

## Library

Ports & Subsystems

## Description

A rectangular block with a thin black border. Inside the block, the word "Action" is written in a simple, black, sans-serif font, centered horizontally and vertically.

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

Use Action Port blocks to create Action subsystems as follows:

- 1 Place a subsystem in the system containing the If or Switch Case block.  
You can use an ordinary subsystem or an atomic subsystem. In either case, the resulting Action subsystem is atomic.
- 2 Add an Action Port to the new subsystem.  
This adds an input port named Action to the subsystem, which is now an Action subsystem.

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

- 1 Create an Action subsystem for each output port configured for an If or Switch Case block.
- 2 Connect each output port (if, else, or elseif ports for the If block; case or default ports for the Switch Case block) to the Action port on an Action subsystem.  
When the connection is made, the icon for the subsystem and the Action Port block it contains are changed to the name of the output port for the If or Switch Case block (i.e., if{ }, else{ }, elseif{ }, case{ }, or default{ }).
- 3 Open the new subsystem and add the diagram that you want to execute in response to the condition this subsystem covers.

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

# Action Port

---

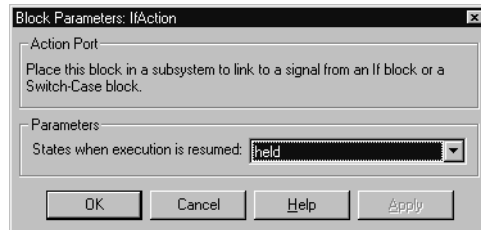
**Note** All blocks in an Action subsystem driven by an If or Switch Case block must run at the same rate as the driving block.

---

## Data Type Support

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

## Parameters and Dialog Box

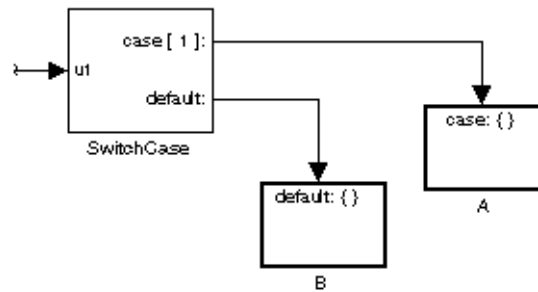


### States when execution is resumed

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

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

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



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

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

<b>Characteristics</b>	Sample Time	Inherited from driving If or Switch Case block.
------------------------	-------------	---

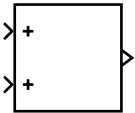
# Add

---

**Purpose** Add or subtract inputs

**Library** Math Operations

**Description** The Add block is an implementation of the Sum block. See Sum on page 1-471 for more information.

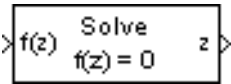




**Purpose** Constrain the input signal to zero

**Library** Math Operations

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

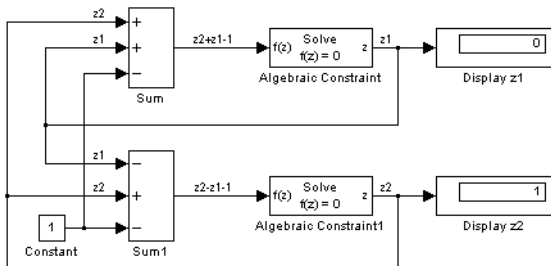


By default, the **Initial guess** parameter is zero. You can improve the efficiency of the algebraic loop solver by providing an **Initial guess** for the algebraic state  $z$  that is close to the solution value.

For example, the following model solves these equations.

$$\begin{aligned} z2 + z1 &= 1 \\ z2 - z1 &= 1 \end{aligned}$$

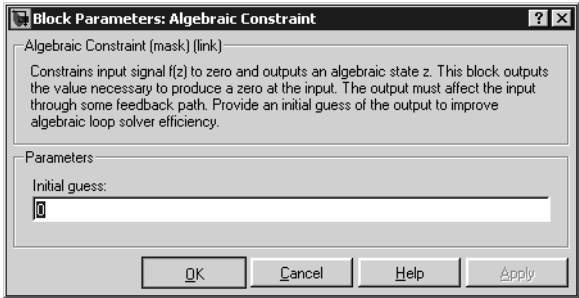
The solution is  $z2 = 1, z1 = 0$ , as the Display blocks show.



**Data Type Support** The Algebraic Constraint block accepts and outputs real values of type double.

# Algebraic Constraint

## Parameters and Dialog Box



### Initial guess

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

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

**Purpose** Check whether a signal is nonzero

**Library** Model Verification

## Description



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

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

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

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

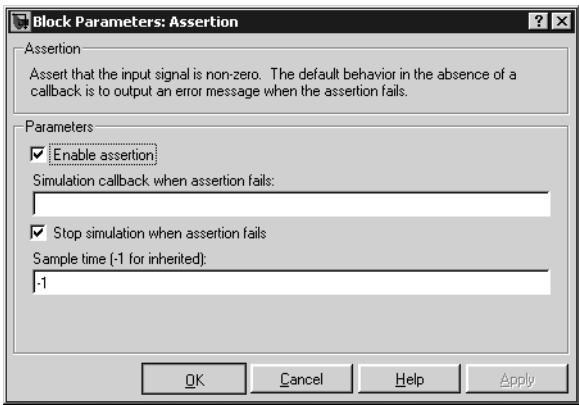
## Data Type Support

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

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

# Assertion

## Parameters and Dialog Box



### Enable Assertion

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

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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

### Sample time (-1 for inherited)

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

### Characteristics

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

Dimensionalized	Yes
Zero Crossing	No

# Assignment

---

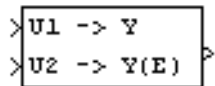
## Purpose

Assign values to specified elements of a signal

## Library

Math Operations

## Description



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

---

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

---

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

### Assigning Values to a Vector Signal

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

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

- -1 (internal source only)

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

- Index of a single element specified as a nonnegative integer

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

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

- A set of indices specified as a vector

Assigns U2 to a specified set of elements of Y.

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

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

## Assigning Values to a Matrix Signal

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

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

- -1 (internal source only)

Specifies all rows or columns of Y.

# Assignment

---

- Single row or column index value

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

- Vector of row or column indices

Specifies a set of rows or columns of Y.

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

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

When determining the dimensions of U2, count a scalar index as a vector of size 1 and -1 as equivalent to a vector of indices of the same width as the row or dimension size of Y. For example, suppose your row and column index sources are a scalar and a two-element vector, respectively. Then U2 must be a 1-by-2 matrix.

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

---

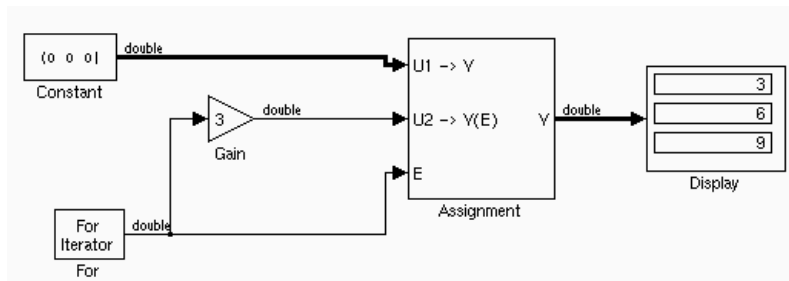
**Note** An Assignment block whose **Input type** is Matrix accepts only matrix signals at its U1 port and only a matrix signal or a one-element vector signal at its U2 port. Simulink displays an error dialog box if you update or simulate a model that violates this constraint.

---



## Iterated Assignment

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



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

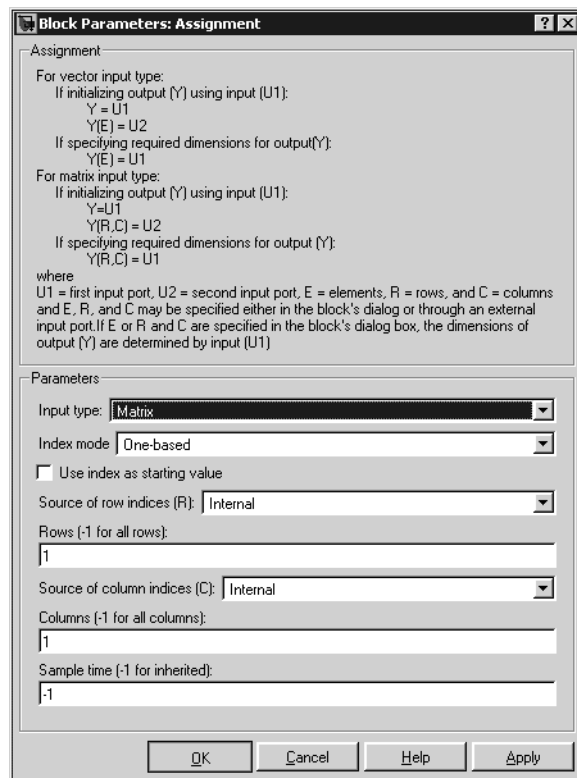
## Data Type Support

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

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

# Assignment

## Parameters and Dialog Box



### Input Type

You can select either Vector or Matrix input. If you select Vector, the **Source of element indices** field appears. If you select Matrix, the **Source of row indices** and **Source of column indices** fields appear.

### Index mode

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

### Use index as starting value

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

### Source of element indices

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

## Elements

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

## Source of row indices

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

## Rows

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

## Source of column indices

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

## Columns

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

## Output (Y)

This control appears only if the source of assignment indices is external or, in the case of matrix assignment, the source of either the row or column indices, or both, is external. The options are Initialize using input (U1) (the default) or Specify required dimensions. The first option causes the

Assignment block to display an initialization port labeled U1 and to use the signal at this port to initialize the output signal (Y) before assigning it values from the data port (U2) as specified by the external indices signal (E). The second option does not initialize Y before assigning values from the block's data input port (labeled U1 in this case) to it. This option requires that the block's U1 and E inputs assign values to every element of Y. Further, it requires that you specify the dimensions of the output signal (see next control).

**Output dimensions**

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

**Diagnostic if not all required dimensions are populated**

This control appears only if you specify the Specify required dimensions option of the **Output (Y)** control. It specifies the diagnostic action that Simulink should take if the block's data (U1) and external indices (E) inputs do not assign a value to every element of the block's output (Y). The options are to display an error message and halt the simulation (Error), display a warning message (Warning) and continue the simulation, or continue the simulation (None). If you choose Warning or None, the values of the unassigned elements of the output are undefined.

**Sample time (-1 for inherited)**

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

**Characteristics**

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

## Purpose

Model the behavior of a system with play

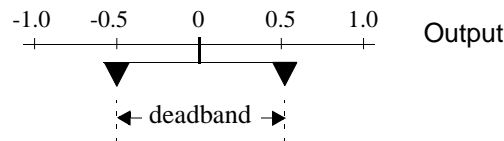
## Library

Discontinuities

## Description



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



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

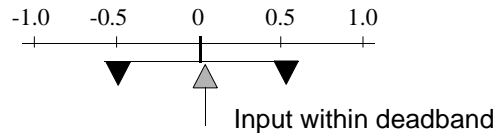
- Disengaged – In this mode, the input does not drive the output and the output remains constant.
- Engaged in a positive direction – In this mode, the input is increasing (has a positive slope) and the output is equal to the input *minus* half the deadband width.
- Engaged in a negative direction –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

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

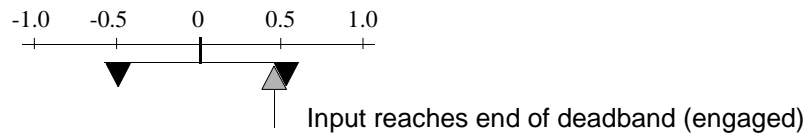
For example, the Backlash block can be used to model the meshing of two gears. The input and output are both shafts with a gear on one end, and the output shaft is driven by the input shaft. Extra space between the gear teeth introduces *play*. The width of this spacing is the **Deadband width** parameter. If the system is disengaged initially, the output (the position of the driven gear) is defined by the **Initial output** parameter.

# Backlash

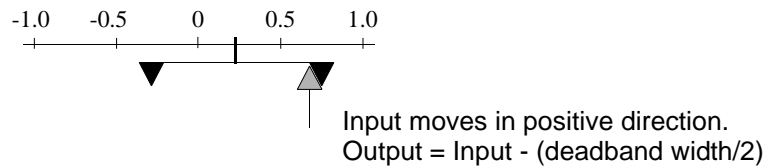
The following figures illustrate the block's operation when the initial input is within the deadband. The first figure shows the relationship between the input and the output while the system is in disengaged mode (and the default parameter values are not changed).



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.

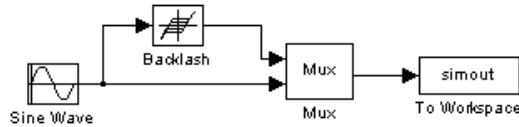


If the input reverses its direction, it disengages from the output. The output remains constant until the input either reaches the opposite end of the deadband or reverses its direction again and engages at the same end of the deadband. Now, as before, movement in the input causes equal movement in the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

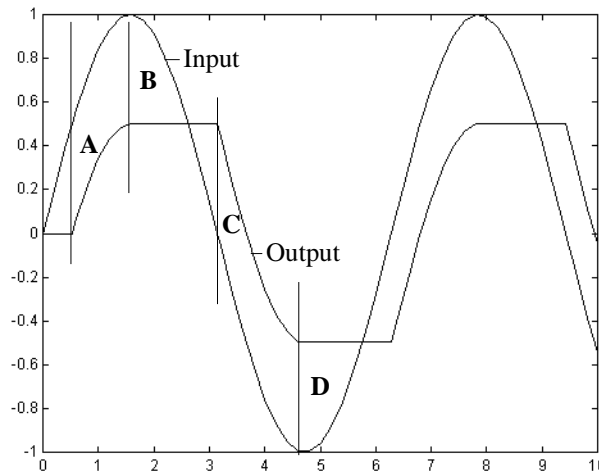
For example, if the deadband width is 2 and the initial output is 5, the output,  $y$ , at the start of the simulation is as follows:

- 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.



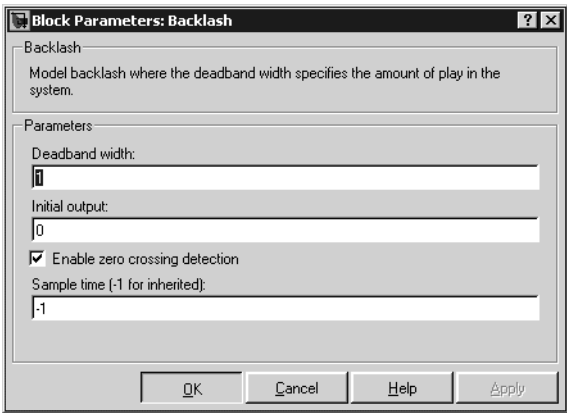
- A** Input engages in positive direction. Change in input causes equal change in output.
- B** Input disengages. Change in input does not affect output.
- C** Input engages in negative direction. Change in input causes equal change in output.
- D** Input disengages. Change in input does not affect output.

## Data Type Support

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

# Backlash

## 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.

### Enable zero crossing detection

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

### Sample time (-1 for inherited)

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

## Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in <b>Sample time</b> parameter.
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	Yes, if you select <b>Enable zero crossing detection</b> .



**Purpose** Introduce white noise into a continuous system

**Library** Sources

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



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

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

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

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

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

## The Algorithm Used in the Block Implementation

To produce the correct intensity of this noise, the covariance of the noise is scaled to reflect the implicit conversion from a continuous PSD to a discrete noise covariance. The appropriate scale factor is  $1/tc$ , where  $tc$  is the correlation time of the noise. This scaling ensures that the response of a continuous system to the approximate white noise has the same covariance as the system would have to true white noise. Because of this scaling, the covariance of the signal from the Band-Limited White Noise block is not the same as the **Noise power** (intensity) dialog box parameter. This parameter is actually the height of the PSD of the white noise. While the covariance of true

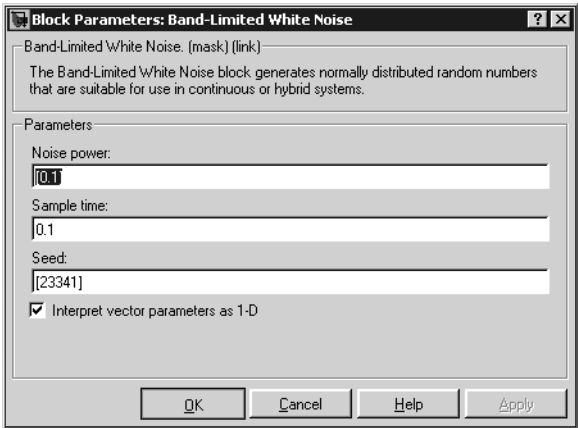
# Band-Limited White Noise

white noise is infinite, the approximation used in this block has the property that the covariance of the block output is the **Noise Power** divided by  $tc$ .

## Data Type Support

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

## Parameters and Dialog Box



### Noise power

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

### Sample time

The correlation time of the noise. The default value is 0.1. See “Specifying Sample Time” in the online documentation for more information.

### Seed

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

### Interpret vector parameters as 1-D

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

## Characteristics

Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes, of <b>Noise power</b> and <b>Seed</b> parameters and output

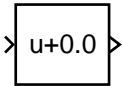
Dimensionalized	Yes
Zero Crossing	No

# Bias

**Purpose** Add a bias to the input

**Library** Math Operations

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



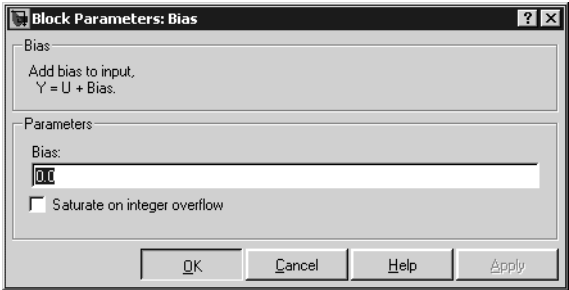
$$Y = U + Bias$$

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

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

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

**Parameters and Dialog Box**



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

**Saturate on integer overflow** Select to have overflows saturate.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from the driving block.
	Scalar Expansion	Yes
	States	0

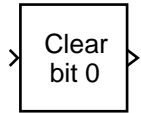
Dimensionalized	Yes
Zero Crossing	No

# Bit Clear

**Purpose** Set the specified bit of the stored integer to zero

**Library** Logic and Bit Operations

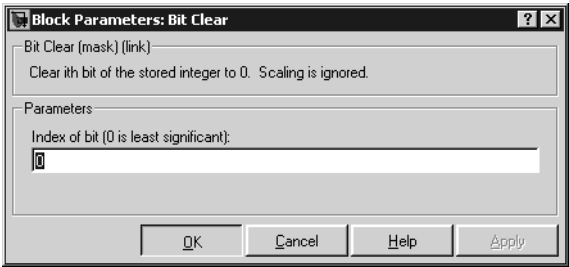
**Description** The Bit Clear block sets the specified bit, given by its index, of the stored integer to zero. Scaling is ignored.



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

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

**Parameters and Dialog Box**



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

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

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

**See Also** Bit Set

**Purpose** Set the specified bit of the stored integer to one

**Library** Logic and Bit Operations

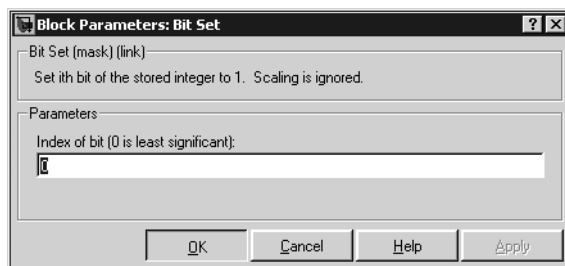
**Description** The Bit Set block sets the specified bit of the stored integer to one. Scaling is ignored.



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

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

## Parameters and Dialog Box



### Index of bit

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

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

**Characteristics**

Direct Feedthrough	Yes
Scalar Expansion	Yes

**See Also** Bit Clear

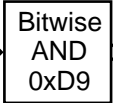
# Bitwise Operator

**Purpose** Perform the specified bitwise operation on the inputs

**Library** Logic and Bit Operations

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

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



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

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

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

- The NOT operator accepts only one input, which can be a scalar or a vector. If the input is a vector, the output is a vector of the same size containing the bitwise logical complements of the input vector elements.
- For a single vector input, the block applies the operation (except the NOT operator) to all elements of the vector. If a bit mask is not specified, then the output is a scalar. If a bit mask is specified, then the output is a vector.



- For two or more inputs, the block performs the operation between all of the inputs. If the inputs are vectors, the operation is performed between corresponding elements of the vectors to produce a vector output.

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

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

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

---

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

---

The **Treat mask as** parameter list controls how the mask is treated. The possible values are Real World Value and Stored Integer. In terms of the general encoding scheme described in the “Scaling” section of the Simulink Fixed Point documentation, Real World Value treats the mask as  $V = SQ + B$  where  $S$  is the slope and  $B$  is the bias. Stored Integer treats the mask as a stored integer,  $Q$ .

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

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

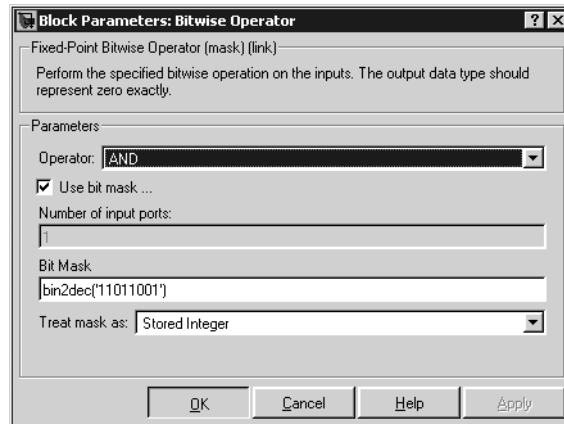
# Bitwise Operator

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.

## 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

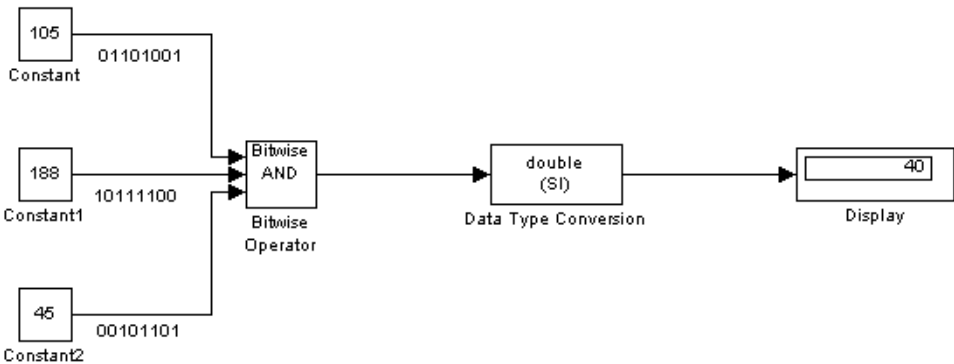
The bit mask to associate with a single input. The **Bit Mask** parameter is converted from a double to the input data type offline using round-to-nearest and saturation.

### Treat mask as

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

## Examples

To help you understand the Bitwise Operator block logic operations, consider the fixed-point model shown below.



The Constant blocks are configured to output an 8-bit unsigned integer (uint(8)). The results for all logic operations are shown below.

Operation	Binary Value	Decimal Value
AND	00101000	40
OR	11111101	253
NAND	11010111	215
NOR	00000010	2
XOR	11111000	248
NOT	N/A	N/A

## Characteristics

Direct Feedthrough	No
Scalar Expansion	Yes, of inputs

# Bus Creator

**Purpose** Create a signal bus

**Library** Signal Routing

**Description**



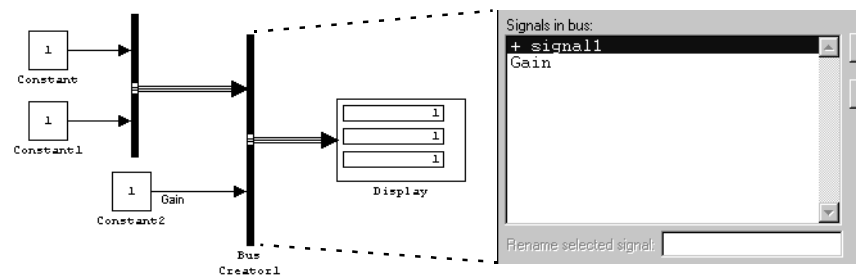
The Bus Creator block combines a set of signals into a bus, i.e., a group of signals represented by a single line in a block diagram. The Bus Creator block, when used in conjunction with the Bus Selector block, allows you to reduce the number of lines required to route signals from one part of a diagram to another. This makes your diagram easier to understand.

To bundle a group of signals with a Bus Creator block, set the block’s **Number of inputs ports** 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. 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.

**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 1-37) 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 inherit 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 on the block's parameter dialog box. The block's parameter dialog box displays the names of the signals currently connected to its inputs or a generated name (for example, `signal2`) for an anonymous input. You can now use the parameter dialog box to change the required names of the block's inputs. To change the required signal name, select the signal in the **Signals in bus** list. The selected signal's name appears in the **Rename selected signal** field. Edit the name in the field and select the parameter dialog box's **Apply** button to apply your edits or the **OK** button to apply the edits and close the dialog box.

## Browsing Bus Signals

The **Signals in bus** list on a Bus Creator block's parameter dialog displays a list of the signals entering the block. A plus sign (+) sign next to a signal indicates that the signal is itself a bus. You can display its contents by clicking the plus sign. If the expanded input includes bus signals, plus signs appear next to the names of those bus signals. You can expand them as well. In this way, you can view all signals entering the block, including those entering via buses. To find the source of any signal entering the block, select the signal in the **Signals in bus** list and click the adjacent **Find** button. Simulink opens the subsystem containing the signal source, if necessary, and highlights the source's icon.

---

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

---

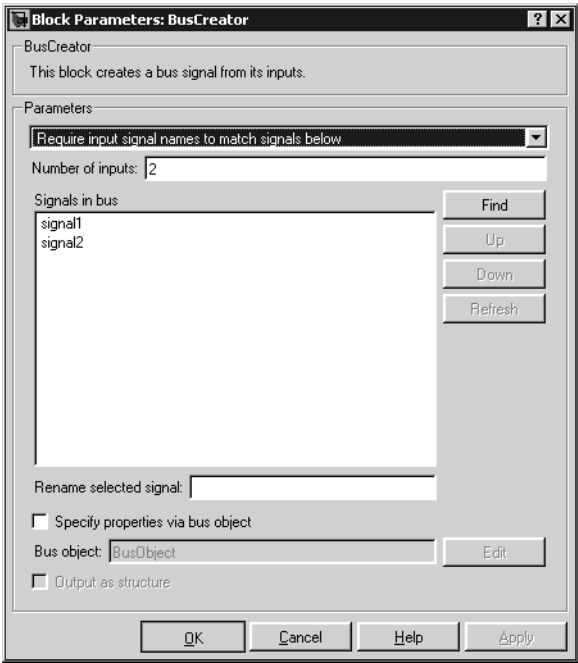
# Bus Creator

## Data Type Support

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

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

## Parameters and Dialog Box



### Signal naming options

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

### Number of inputs

Specifies the number of input ports on this block.

### Signals in bus

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

plus sign to display the subsidiary bus signals. Click the **Refresh** button to update the list after editing the name of an input signal. Click the **Find** button to highlight the source of the currently selected signal.

## **Rename selected signal**

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

## **Specify properties via bus object**

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

## **Bus object**

This option is enabled only if you select the **Specify properties via bus object option**. It specifies the name of bus object used to define the structure of the bus created by this block. At the beginning of a simulation or when you update the model’s diagram, Simulink checks whether the signals connected to this Bus Creator block have the specified structure. If not, Simulink displays an error message.

## **Output as structure**

This option is enabled only if you select the **Specify properties via bus object option**. If this option is selected, this block outputs a nonvirtual bus; otherwise, it outputs a virtual bus (see “Virtual Versus Nonvirtual Buses” in the online Simulink documentation). 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.

# Bus Selector

**Purpose** Select signals from an incoming bus

**Library** Signal Routing

**Description** The Bus Selector block accepts input from a Bus Creator block or another Bus Selector block. This block has one input port. The number of output ports depends on the state of the **Muxed output** check box. If you select **Muxed output**, the signals are combined at the output port and there is only one output port; otherwise, there is one output port for each selected signal.

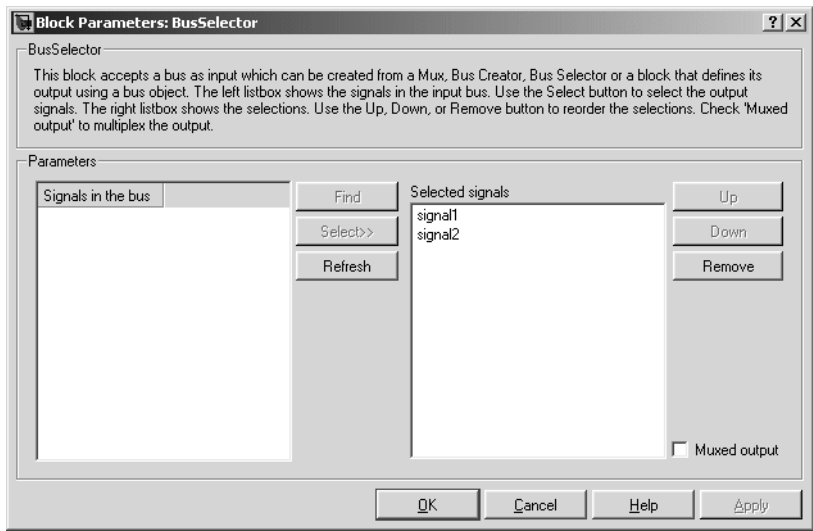


**Note** Simulink hides the name of a Bus Selector block when you copy it from the Simulink library to a model.

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

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

## Parameters and Dialog Box





## Signals in the bus

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

## Selected signals

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

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

The signal label at the output port is automatically set by the block except when you select the **Muxed output** check box. If you try to change this label, you get an error message stating that you cannot change the signal label of a line connected to the output of a Bus Selector block.

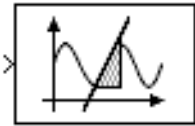
# Check Discrete Gradient

---

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

**Library** Model Verification

## Description



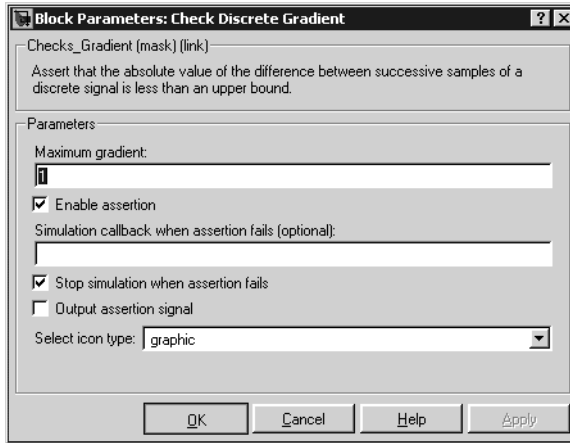
The Check Discrete Gradient block checks each signal element at its input to determine whether the absolute value of the difference between successive samples of the element is less than an upper bound. The block's parameter dialog box allows you to specify the value of the upper bound (1 by default).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

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

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

**Data Type Support** The Check Discrete Gradient block accepts single, double, int8, int16, and int32 input signals of any dimensions.

## Parameters and Dialog Box



### Maximum gradient

Upper bound on the gradient of the discrete input signal.

### Enable Assertion

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

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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

### Output Assertion Signal

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

# Check Discrete Gradient

---

selected the Implement logic signals as boolean data option on the **Simulation and code generation** optimization pane of Simulink's **Configuration Parameters** dialog box. Otherwise the data type of the output signal is double.

## Select Icon Type

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

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

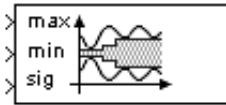
## Purpose

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

## Library

Model Verification

## Description



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

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

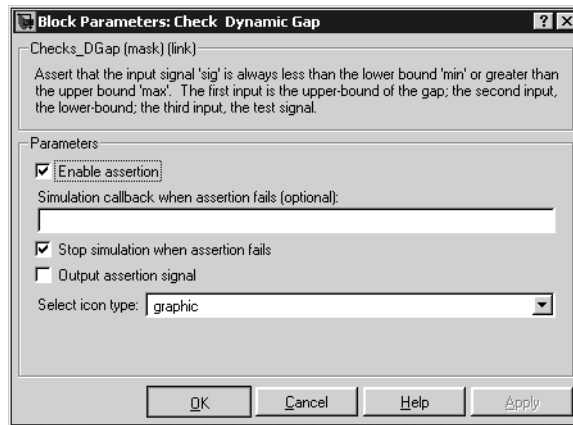
## Data Type Support

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

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

# Check Dynamic Gap

## Parameters and Dialog Box



### Enable Assertion

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

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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

### Output Assertion Signal

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

## Select Icon Type

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

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Check Dynamic Lower Bound

---

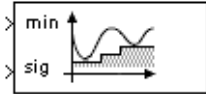
## Purpose

Check that one signal is always less than another signal

## Library

Model Verification

## Description



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

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

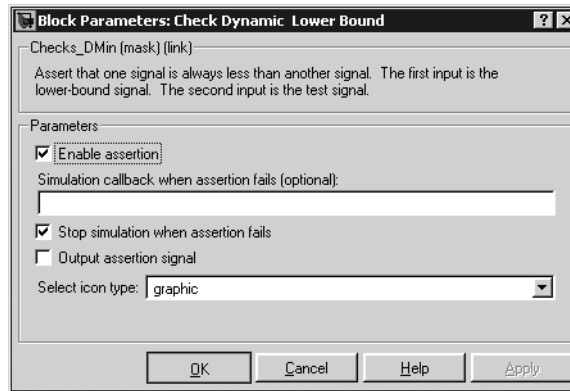
## Data Type Support

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

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



## Parameters and Dialog Box



### Enable Assertion

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

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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

### Output Assertion Signal

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

# Check Dynamic Lower Bound

---

## Select Icon Type

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

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

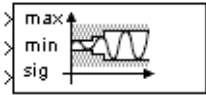
## Purpose

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

## Library

Model Verification

## Description



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

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

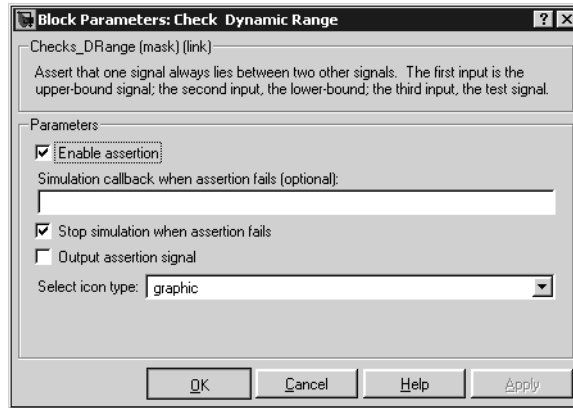
## Data Type Support

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

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

# Check Dynamic Range

## Parameters and Dialog Box



### Enable Assertion

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

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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

### Output Assertion Signal

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

## Select Icon Type

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

## Characteristics

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

# Check Dynamic Upper Bound

---

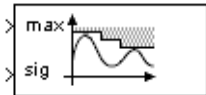
## Purpose

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

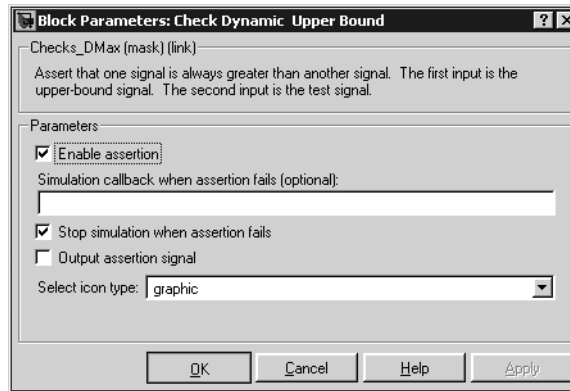
The Check Dynamic Upper Bound block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied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

## Data Type Support

The Check Dynamic Upper Bound block accepts input signals of any dimensions and of any data type supported by Simulink. The test and the reference signals must have the same dimensions and data type. If the inputs are nonscalar, the block 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, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.

## Parameters and Dialog Box



### Enable Assertion

Unchecking this option 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 on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all model verification blocks, including Check Dynamic Upper Bound blocks, in a model regardless of the setting of this option.

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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

### Output Assertion Signal

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

# Check Dynamic Upper Bound

---

## Select Icon Type

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

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

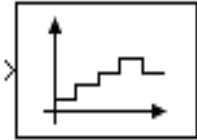


**Purpose**

Check that the input signal has a 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” on page 1-58). 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.

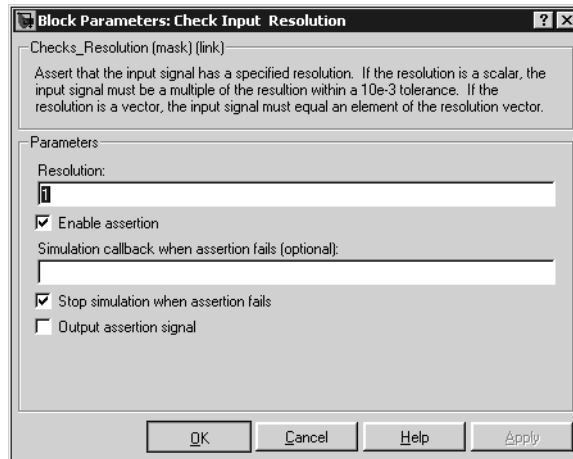
**Data Type Support**

The Check Input Resolution block accepts input signals of any dimensions and of any data type supported by Simulink. 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, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.

# Check Input Resolution

## Parameters and Dialog Box



### Resolution

Resolution that the input signal must have.

### Enable Assertion

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

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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

### Output Assertion Signal

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

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

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Check Static Gap

---

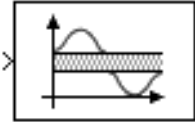
## Purpose

Check that a gap exists in a 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.

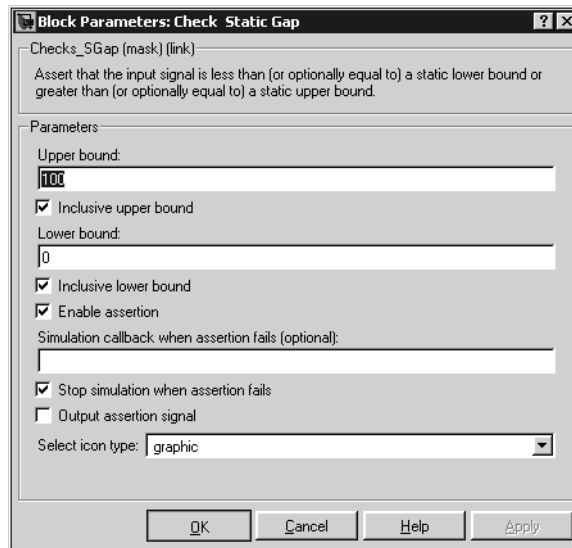
The Check Static Gap block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that a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

## Data Type Support

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

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

## Parameters and Dialog Box



### Upper bound

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

### Inclusive upper bound

If checked, this option 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

If checked, this option specifies that the gap includes the lower bound.

### Enable Assertion

Unchecking this option 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 on the **Data Integrity** diagnostics pane of the **Configuration Parameters** dialog box allows you to enable or disable all model verification blocks, including Check Static Gap blocks, in a model regardless of the setting of this option.

### Simulation callback when assertion fails

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

# Check Static Gap

---

## Stop simulation when assertion fails

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

## Output Assertion Signal

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

## Select Icon Type

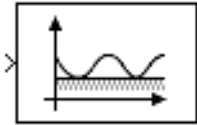
Type of icon used to display this block in a block diagram: either graphic or text. The graphic option displays a graphical representation of the assertion condition on the icon. The text option displays a mathematical expression that represents the assertion condition. If the icon is too small to display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

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

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

**Library** Model Verification

## Description



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

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

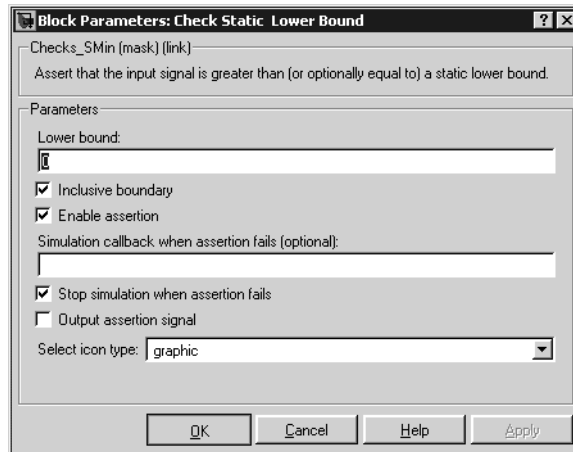
## Data Type Support

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

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

# Check Static Lower Bound

## Parameters and Dialog Box



### Lower bound

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

### Inclusive boundary

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

### Enable Assertion

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

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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



## Output Assertion Signal

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

## Select Icon Type

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

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Check Static Range

---

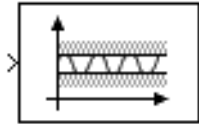
## Purpose

Check that a signal falls inside a fixed range of amplitudes

## Library

Model Verification

## Description



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

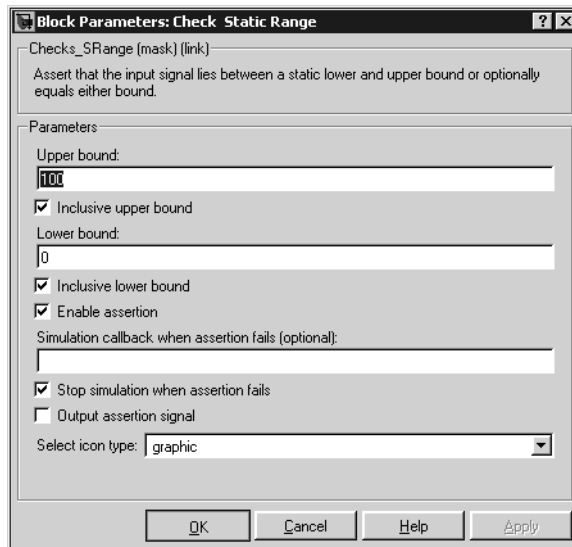
The Check Static Range block and its companion blocks in the Model Verification library are intended to facilitate creation of self-validating models. For example, you can use model verification blocks to test that signals do not exceed specified limits during simulation. When you are satisfied that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

## Data Type Support

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

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

## Parameters and Dialog Box



### Upper bound

Upper bound of the range of valid input signal amplitudes.

### Inclusive upper bound

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

### Lower bound

Lower bound of the range of valid input signal amplitudes.

### Inclusive lower bound

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

### Enable Assertion

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

# Check Static Range

## Simulation callback when assertion fails

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

## Stop simulation when assertion fails

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

## Output Assertion Signal

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

## Select Icon Type

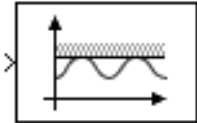
Type of icon used to display this block in a block diagram: either graphic or text. The graphic option displays a graphical representation of the assertion condition on the icon. The text option displays a mathematical expression that represents the assertion condition. If the icon is too small to display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

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

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

**Library** Model Verification

## Description



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

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

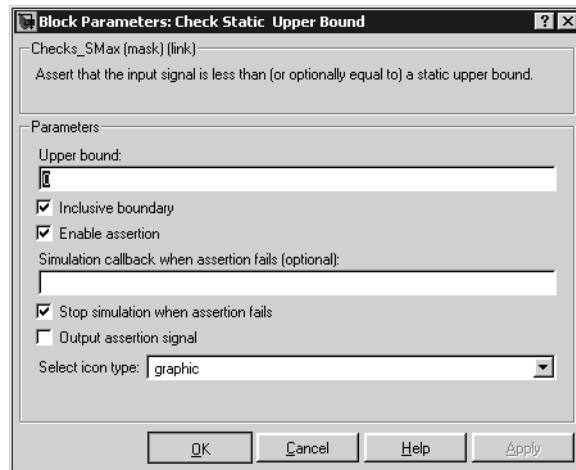
## Data Type Support

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

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

# Check Static Upper Bound

## Parameters and Dialog Box



### Upper bound

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

### Inclusive boundary

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

### Enable Assertion

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

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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

## Output Assertion Signal

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

## Select Icon Type

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

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Chirp Signal

**Purpose** Generate a sine wave with increasing frequency

**Library** Sources

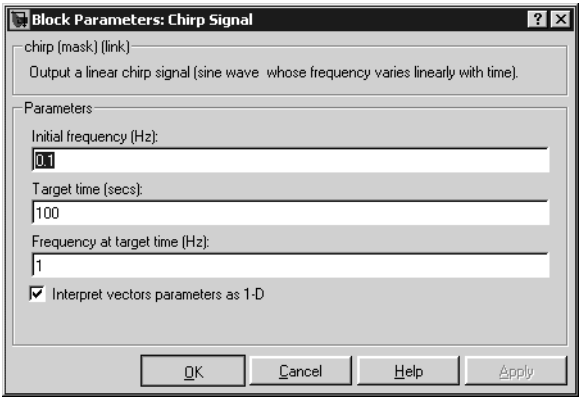
**Description** The Chirp Signal block generates a sine wave whose frequency increases at a linear rate with time. You can use this block for spectral analysis of nonlinear systems. The block generates a scalar or vector output.



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

**Data Type Support** The Chirp Signal block outputs a real-valued signal of type double.

## Parameters and Dialog Box



**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.

<b>Characteristics</b>	Sample Time	Continuous
	Scalar Expansion	Yes, of parameters
	Dimensionalized	Yes
	Zero Crossing	No

# Clock

**Purpose** Display and provide the simulation time

**Library** Sources

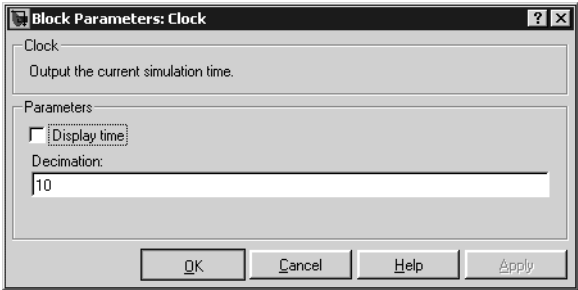
**Description** The Clock block outputs the current simulation time at each simulation step. This block is useful for other blocks that need the simulation time.



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

**Data Type Support** The Clock block outputs a real-valued signal of type double.

**Parameters and Dialog Box**



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

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

<b>Characteristics</b>	Sample Time	Continuous
	Scalar Expansion	N/A
	Dimensionalized	No
	Zero Crossing	No

**Purpose** Implement a truth table

**Library** Logic and Bit Operations

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



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

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

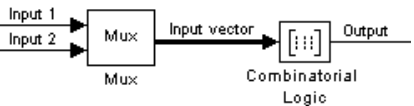
$$\text{number of rows} = 2^{\text{(number of inputs)}}$$

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

$$\text{row index} = 1 + u(m) \cdot 2^0 + u(m-1) \cdot 2^1 + \dots + u(1) \cdot 2^{m-1}$$

### 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

# Combinatorial Logic

column with the same name. The combination of these values determines the row of the Output column of the table that is passed as block output.

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

$$(21*1 + 1)$$

The output value is 0.

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

## Example of Circuit

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

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

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

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

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

## Data Type Support

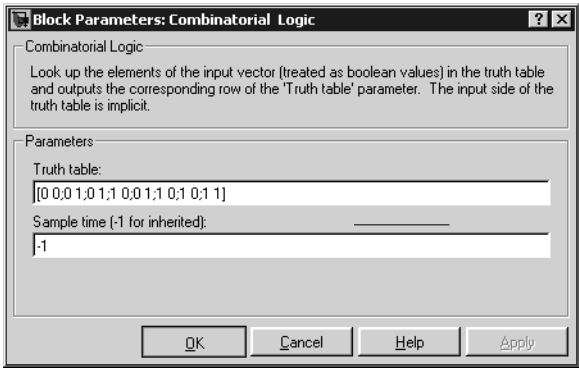
The type of signals accepted by a Combinatorial Logic block depends on whether you selected the Boolean logic signals option (see “Enabling Strict Boolean Type Checking” in the Using Simulink documentation). If this option is enabled, the block accepts real signals of type `Boolean` or `double`. The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

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

If Boolean compatibility mode is disabled, the Combinatorial Logic block accepts only signals of type `Boolean`. The block outputs `double` if the truth table contains non-Boolean values of type `double`. Otherwise, the output is `Boolean`.

# Combinatorial Logic

## Parameters and Dialog Box



### Truth table

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

### Sample time (-1 for inherited)

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

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

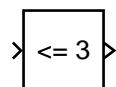
## Purpose

Determine how a signal compares to the specified constant

## Library

Logic and Bit Operations

## Description



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

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

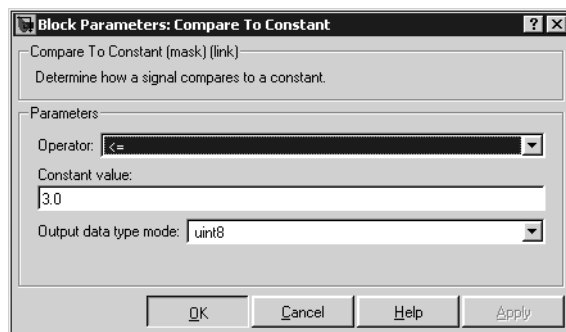
The output is 0 if the comparison is false, and 1 if it is true.

## Data Type Support

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

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

## Parameters and Dialog Box



# Compare To Constant

---

**Operator**

Specify how the input is compared to the constant value, as discussed in “Description” on page 1-79.

**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.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

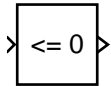
<b>See Also</b>	Compare To Zero
-----------------	-----------------



**Purpose** Determine how a signal compares to zero

**Library** Logic and Bit Operations

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



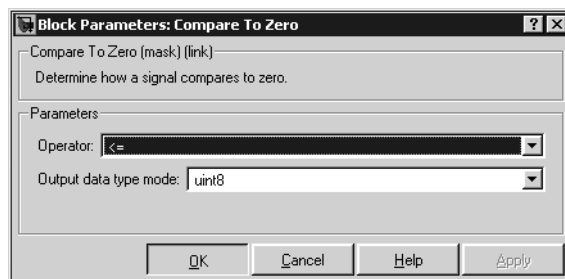
- == — Determine whether the input is equal to zero.
- ~= — Determine whether the input is not equal to zero.
- < — Determine whether the input is less than zero.
- <= — Determine whether the input is less than or equal to zero.
- > — Determine whether the input is greater than zero.
- >= — Determine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

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

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

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

## Parameters and Dialog Box



### Operator

Specify how the input is compared to zero, as discussed in “Description” on page 1-81.

# Compare To Zero

---

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

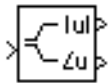
<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

<b>See Also</b>	Compare To Constant
-----------------	---------------------

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

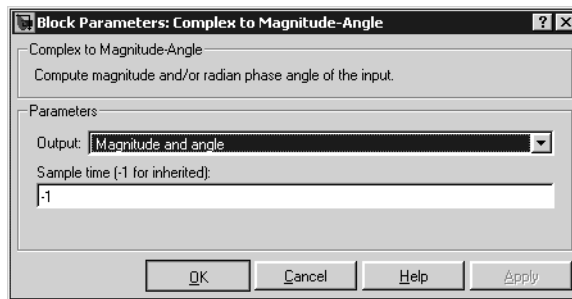
**Library** Math Operations

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



**Data Type Support** See the preceding description.

## Parameters and Dialog Box



## Output

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

## Sample time (-1 for inherited)

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

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block

# Complex to Magnitude-Angle

---

Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

**Purpose**

Output the real and imaginary parts of a complex input signal

**Library**

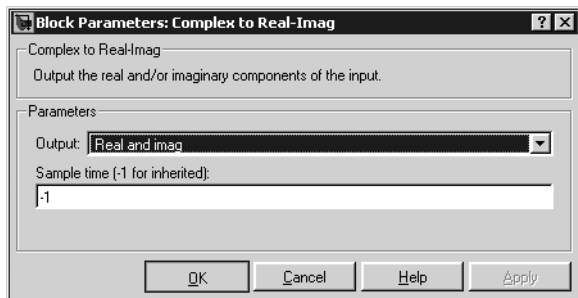
Math Operations

**Description**

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

**Data Type Support**

See the preceding description. For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.

**Parameters and Dialog Box****Output**

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

**Sample time (-1 for inherited)**

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

# Complex to Real-Imag

---

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

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

**Library** Ports & Subsystems

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



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

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

## Creating a Master Configurable Subsystem

To create a master configurable subsystem:

- 1 Create a library of blocks representing the various configurations of the configurable subsystem.
- 2 Save the library.
- 3 Create an instance of the Configurable Subsystem block in the library.  
To do this, drag a copy of the Configurable Subsystem block from the Simulink Signals and Systems library into the library you created in the preceding step.
- 4 Display the Configurable Subsystem block's dialog by double-clicking it. The dialog displays a list of the other blocks in the library.
- 5 Select the blocks that represent the various configurations of the configurable subsystems you are creating.

# Configurable Subsystem

---

- 6 Select **Block Choice** from the subsystem's context menu.  
The context menu displays a submenu listing the blocks that the subsystem can represent.
- 7 Select the block that you want the subsystem to represent by default.
- 8 Close the dialog.
- 9 Save the library.

---

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

---

## 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.

## Mapping I/O Ports

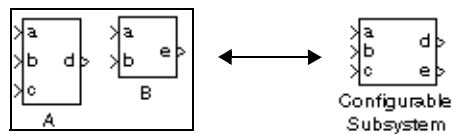
A configurable subsystem displays a set of input and output ports corresponding to input and output ports in the selected library. Simulink uses the following rules to map library ports to Configurable Subsystem block ports:

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



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

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



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

---

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

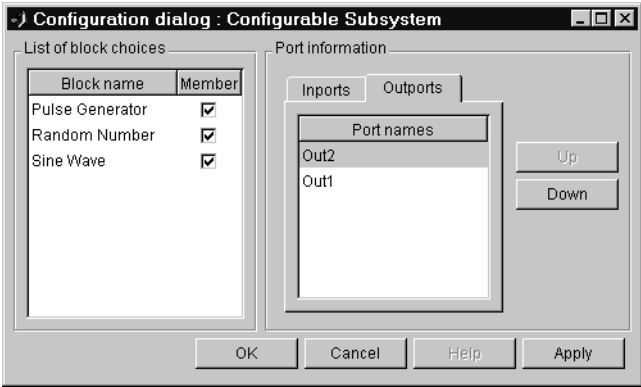
---

## Data Type Support

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

# Configurable Subsystem

## 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.

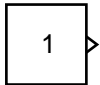
### Characteristics

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 a constant value

**Library** Sources

## Description



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

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

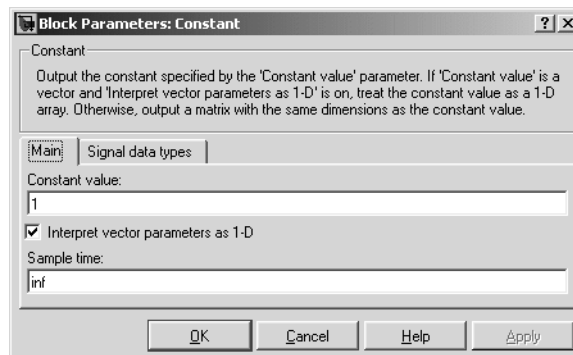
## Data Type Support

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

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

## Parameters and Dialog Box

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



## Constant value

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

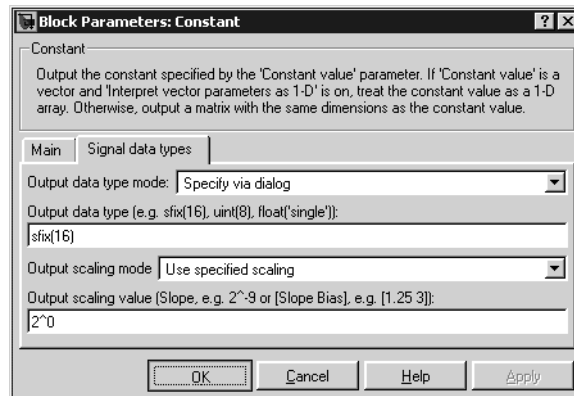
## Interpret vector parameters as 1-D

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

## Sample time

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

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



## Output data type mode

Specify how the data type of the output is designated. The data type can be inherited through backpropagation, or can be designated in the **Constant value** parameter, for example `int8(29)`. You can also choose a built-in

data type from the list. If you choose `Specify via dialog`, the following parameters become visible.

# Constant

---

### Output data type

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

### Output scaling mode

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

### Output scaling value

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

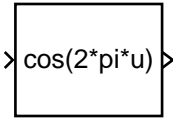
### Characteristics

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

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

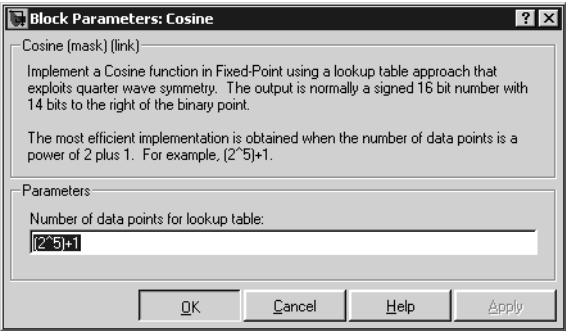
**Library** Lookup Tables

**Description** The Cosine block implements a cosine function using a lookup table that exploits quarter wave symmetry. The output is normally a signed 16 bit number with 14 bits to the right of the binary point.



**Data Type Support** The Cosine block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



**Number of data points for lookup table**  
The number of data points in the lookup table

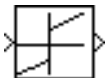
**Characteristics** Direct Feedthrough Yes

# Coulomb and Viscous Friction

**Purpose** Model discontinuity at zero, with linear gain elsewhere

**Library** Discontinuities

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



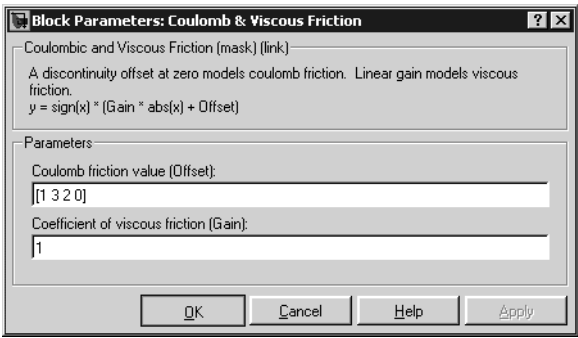
$$y = \text{sign}(u) * (\text{Gain} * \text{abs}(u) + \text{Offset})$$

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

The block accepts one input and generates one output.

**Data Type Support** The Coulomb and Viscous Friction block accepts and outputs real signals of type double.

## Parameters and Dialog Box



### Coulomb friction value

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


### Coefficient of viscous friction

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



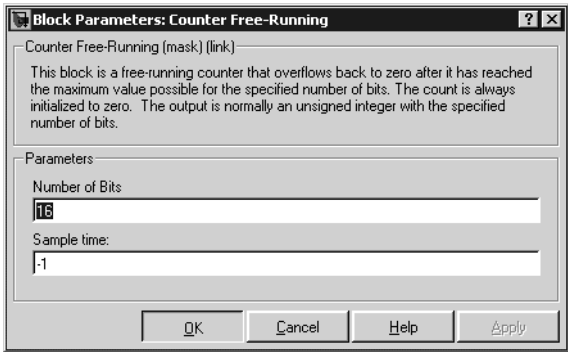
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	Yes, at the point where the static friction is overcome

# Counter Free-Running

<b>Purpose</b>	Count up and overflow back to zero after the maximum value possible is reached for the specified number of bits
<b>Library</b>	Sources
<b>Description</b>	<div></div> <p>The Counter Free-Running block counts up until the maximum possible value, <math>2^{N_{\text{bits}}} - 1</math>, is reached, where <math>N_{\text{bits}}</math> is the number of bits. Then the counter overflows to zero, and restarts counting up. The counter is always initialized to zero.</p> <p>You can specify the number of bits with the <b>Number of Bits</b> parameter.</p> <p>You can specify the sample time with the <b>Sample time</b> parameter.</p> <p>The output is an unsigned integer. If you select the global doubles override, the Counter Free-Running block does not wrap back to zero.</p>

<b>Data Type Support</b>	The Counter Free-Running block accepts signals of any data type supported by Simulink, including fixed-point data types.
--------------------------	--

## Parameters and Dialog Box



**Number of Bits**  
Specified number of bits.

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

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	No
<b>See Also</b>	Counter Limited	

# Counter Limited

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

**Library** Sources

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



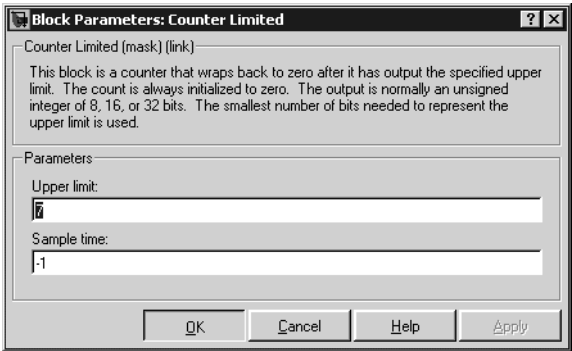
You can specify the upper limit with the **Upper limit** parameter.

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

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

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

## Parameters and Dialog Box



**Upper limit** Upper limit.

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

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	No

<b>See Also</b>	Counter Free-Running
-----------------	----------------------

# Data Store Memory

---

**Purpose** Define a data store

**Library** Signal Routing

## Description



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

Each data store must be defined by a Data Store Memory block. The location of the Data Store Memory block that defines a data store determines the Data Store Read and Data Store Write blocks that can access the data store:

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

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

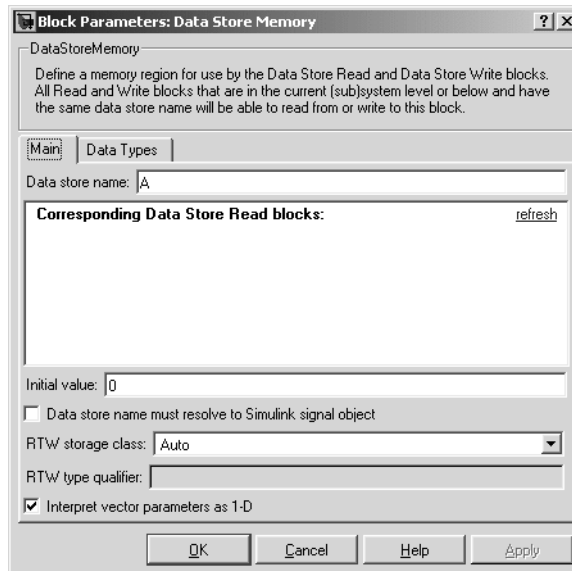
## Data Type Support

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

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

# Data Store Memory

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

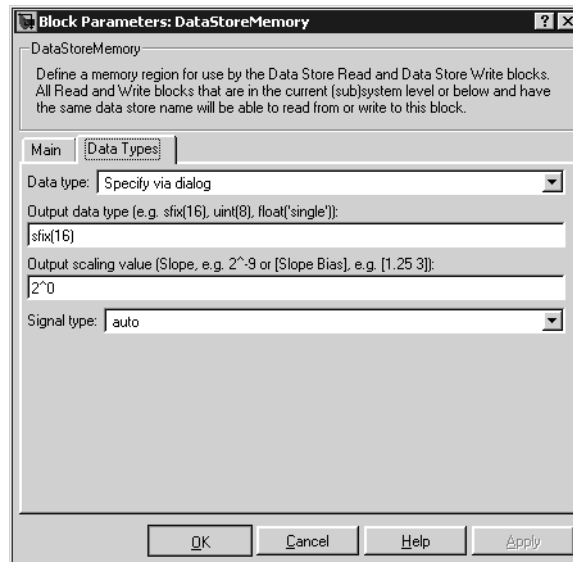
- **Data store name must resolve to Simulink signal object**
- **RTW storage class**
- **RTW type qualifier**

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

## Interpret vector parameters as 1-D

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

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



## Data type

Specify the data type of the values stored in the data store. You can choose a data type from the drop-down list, or you can choose **Specify via dialog** to specify the data type and scaling through the **Output data type** and **Output scaling value** parameters.



## Output data type

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

## Output scaling value

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

## Signal type

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

<b>Characteristics</b>	Sample Time	N/A
	Dimensionalized	Yes
<b>See Also</b>	Data Store Read, Data Store Write	

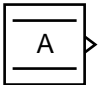
# Data Store Read

---

**Purpose** Read data from a data store

**Library** Signal Routing

## Description



The Data Store Read block copies data from the named data store to its output. The data is initialized by a Data Store Memory block and possibly written by a Data Store Write block.

The data store from which the data is read is determined by the location of the Data Store Memory block that defines the data store. For more information, see “Data Store Memory” on page 1-102.

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

---

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

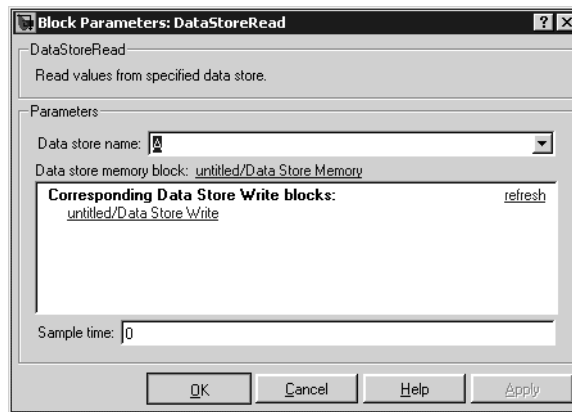
---

## Data Type Support

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

For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using 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 compiles the model containing this block, Simulink searches the model upwards from this block's level for a Data Store Memory block having the specified data store name. Simulink also searches the MATLAB workspace for a Simulink.Signal object have the specified name. If it finds the data store block and the signal object, it checks the block's properties (data type, dimensions, complexity, etc.) against those specified by the signal object. If they disagree, Simulink halts the compilation and displays an error. If Simulink finds the signal object, but not the data store block, Simulink creates a hidden Data Store Memory block at the model's root level having the properties specified by the signal object and an initial value of 0. If Simulink finds the data store but not the signal object, it continues updating the model.

### Data store memory block

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

# Data Store Read

---

## Data store write blocks

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

## Sample time

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

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Dimensionalized	Yes

<b>See Also</b>	Data Store Memory, Data Store Write
-----------------	-------------------------------------

**Purpose** Write data to a data store

**Library** Signal Routing

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



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

The data store to which this block writes is determined by the location of the Data Store Memory block that defines the data store. For more information, see “Data Store Memory” on page 1-102. The size of the data store is set by the Data Store Memory block that defines and initializes the data store. Each Data Store Write block that writes to that data store must write the same amount of data.

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

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

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

## Parameters and Dialog Box



# Data Store Write

---

## Data store name

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

When Simulink compiles the model containing this block, Simulink searches the model upwards from this block's level for a Data Store Memory block having the specified data store name. Simulink also searches the MATLAB workspace for a Simulink.Signal object have the specified name. If it finds the data store block and the signal object, it checks the block's properties (data type, dimensions, complexity, etc.) against those specified by the signal object. If they disagree, Simulink halts the compilation and displays an error. If Simulink finds the signal object, but not the data store block, Simulink creates a hidden Data Store Memory block at the model's root level having the properties specified by the signal object and an initial value of 0. If Simulink finds the data store but not the signal object, it continues updating the model.

## Data store memory block

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

## Data store read blocks

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

## Sample time

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

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Dimensionalized	Yes

### See Also

Data Store Memory, Data Store Read

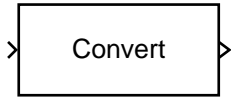
# Data Type Conversion

---

**Purpose** Convert an input signal to a specified data type

**Library** Signal Attributes

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



---

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

---

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

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

## Working with Fixed-Point Values Greater than 32 Bits

The MATLAB built-in integer data types are limited to 32 bits. If you want to output fixed-point numbers that range between 33 and 53 bits without loss of precision or range, you should break the number into pieces using the Gain



block, and then output the pieces using the Data Type Conversion block to store the value inside a double.

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

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

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

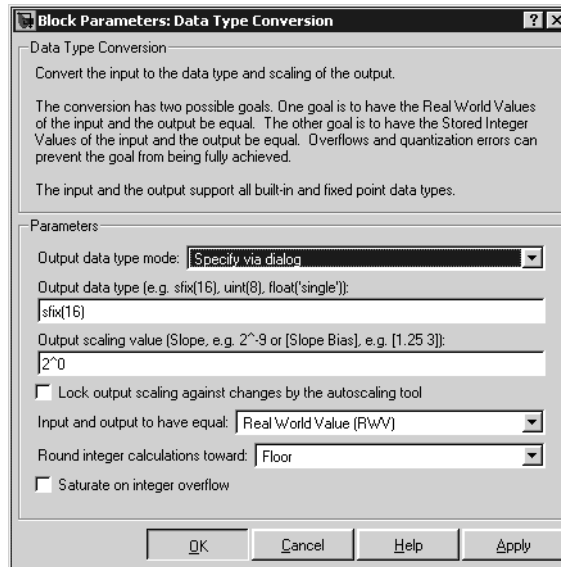
## Data Type Support

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

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

# Data Type Conversion

## Parameters and Dialog Box



### Output data type mode

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

### Output data type

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

### Output scaling value

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

### Lock output scaling against changes by the autoscaling tool

Select to lock scaling of outputs. This parameter is only visible if you select **Specify via dialog** for the **Output data type mode** parameter.

## 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.

## Round integer calculations toward

Select the rounding mode for fixed-point operations.

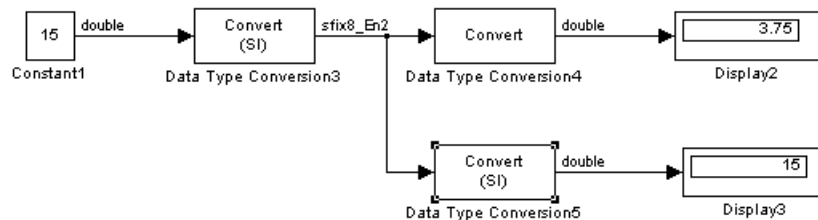
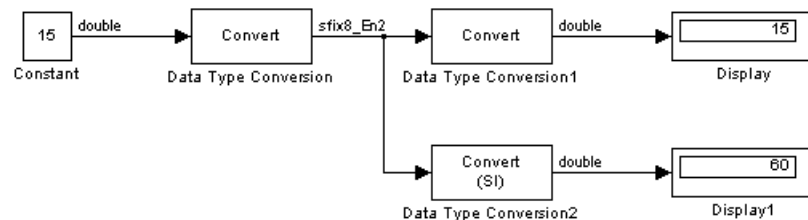
## Saturate on integer overflow

Select to have overflows saturate.

## Examples

### Example 1 — Real World Values Versus Stored Integers

This example uses the Data Type Conversion block to help you understand 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

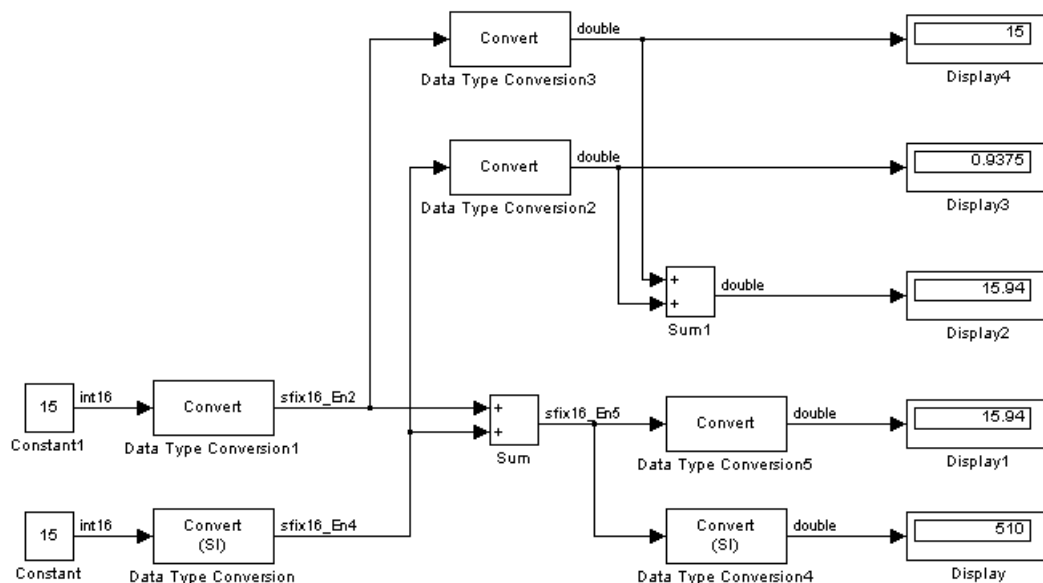
# Data Type Conversion

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
	Zero Crossing	No

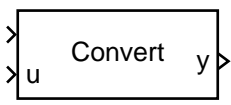
See Also      Data Type Conversion Inherited

# 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 (top) input is used as the reference signal and the second (bottom) input is converted to the reference type by inheriting the data type and scaling information. Either input is scalar expanded such that the output has the same width as the widest input.



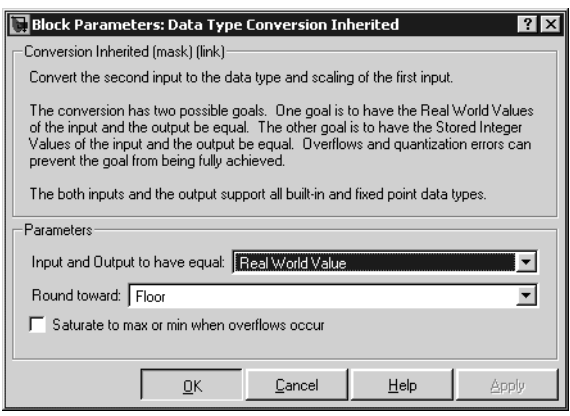
Inheriting the data type and scaling provides these advantages:

- 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 Support block handles any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using 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” on page

1-112 in the Data Type Conversion block reference page for more information about these choices.

**Round toward**

Select the rounding mode for fixed-point operations.

**Saturate to max or min when overflows occur**

Select to have overflows saturate.

**Characteristics**    Direct Feedthrough    Yes

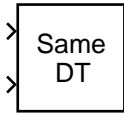
**See Also**            Data Type Conversion

# Data Type Duplicate

**Purpose** Force all inputs to the 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.



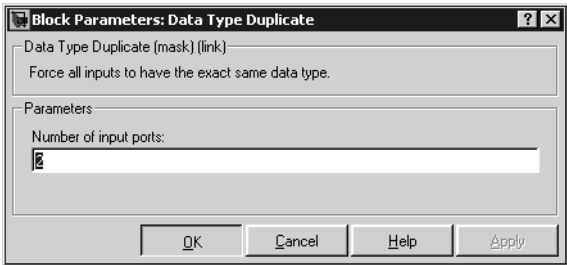
You can use the Data Type Duplicate block to check for consistency of data types among blocks. If all signals do not have the same data type, the block returns an error message.

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 backpropagation.

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** The Data Type Duplicate block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



**Number of input ports** Number of input ports.



Characteristics	Scalar Expansion	Yes
	States	0

# Data Type Propagation

---

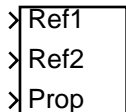
## Purpose

Set the data type and scaling of the propagated signal based on information from the 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 **Specify data type and scaling** parameter configured to 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 “Inheriting Data Type Information” of the Simulink Fixed Point documentation.

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 “Inheriting Scaling Information” in the Simulink Fixed Point documentation.

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 visa 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.

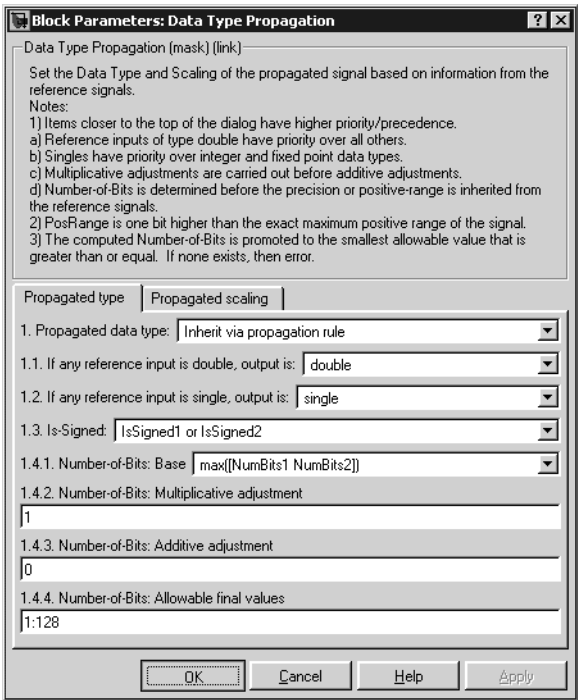
# Data Type Propagation

## Data Type Support

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

## 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 visa versa.

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

# Data Type Propagation

## 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 documentation for more information about the base data type.

This parameter is only visible 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 only visible 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 may overflow. To prevent overflow, you can

associate guard bits with the propagated data type. To associate four guard bits, you specify the value 4.

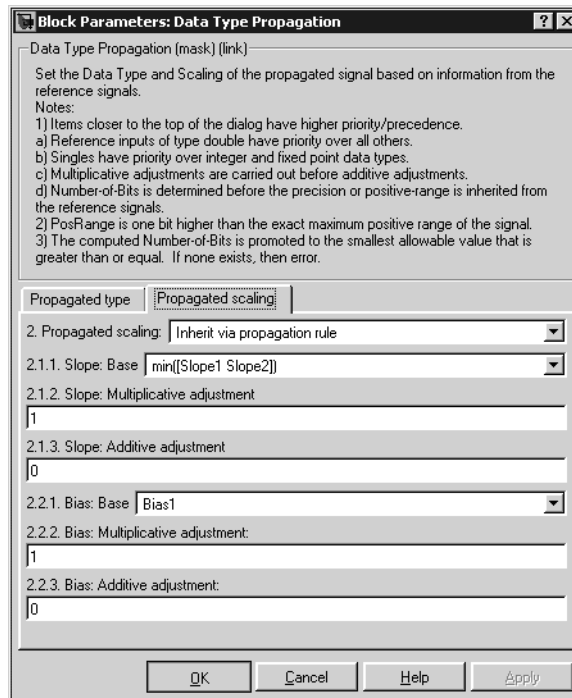
This parameter is only visible if *Inherit via propagation rule* is selected for the **Propagated data type** parameter list.

## Number-of-bits: Allowable final values

parameter allows you to 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 only visible 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:



# Data Type Propagation

## Propagated scaling

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

## 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 only visible 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.
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.



Parameter Value	Description
<code>min([PosRange1 PosRange2])</code>	The range of Prop is given by the minimum range of the reference signals.
<code>PosRange1*PosRange2</code>	The range of Prop is given by the product of the reference signal ranges.
<code>PosRange1/PosRange2</code>	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 only visible 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 only visible 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 only visible if Inherit via propagation rule is selected for the **Propagated scaling** parameter list.

# Data Type Propagation

## Bias: Base

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

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

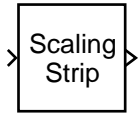
<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

# Data Type Scaling Strip

**Purpose** Remove scaling and map to a built in integer

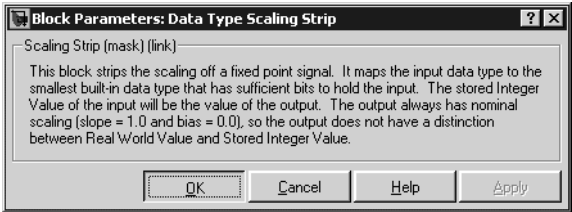
**Library** Signal Attributes

**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 data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



**Characteristics** Direct Feedthrough Yes  
Scalar Expansion Yes

**Purpose** Provide a region of zero output

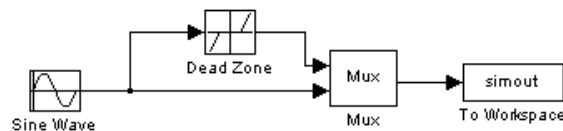
**Library** Discontinuities

**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:

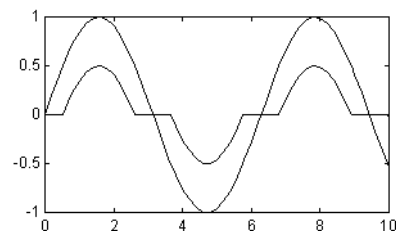


- If the input is within the dead zone (greater than the lower limit and less than the upper limit), the output is zero.
- If the input is greater than or equal to the upper limit, the output is the input minus the upper limit.
- If the input is less than or equal to the lower limit, the output is the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

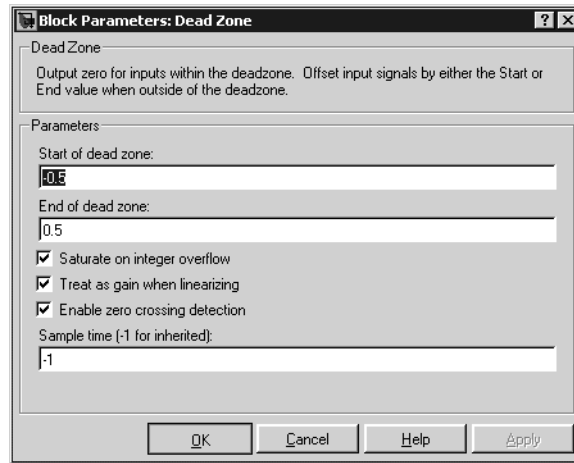
The Dead Zone block accepts and outputs a real signal of any data type supported by Simulink, except Boolean. The Dead Zone block supports fixed-point data types.

# Dead Zone

---

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

### Treat as gain when linearizing

The linearization commands in Simulink 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 to detect when the limits are reached. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

**Sample time (-1 for inherited)**

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

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of parameters
	Dimensionalized	Yes

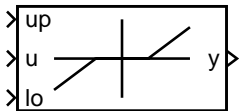
<b>See Also</b>	Dead Zone Dynamic
-----------------	-------------------

# Dead Zone Dynamic

**Purpose** Set inputs within the bounds to zero

**Library** Discontinuities

**Description**



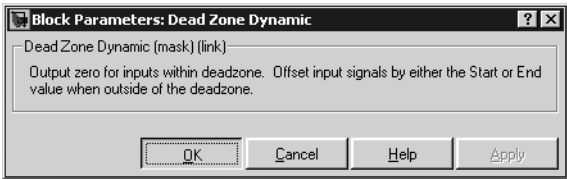
The Dead Zone Dynamic block dynamically bounds the range of the input signal, providing a region of zero output. The bounds change according to the upper and lower limit input signals where

- 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 data type supported by Simulink, including fixed-point data types.

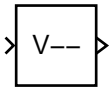

**Parameters and Dialog Box**




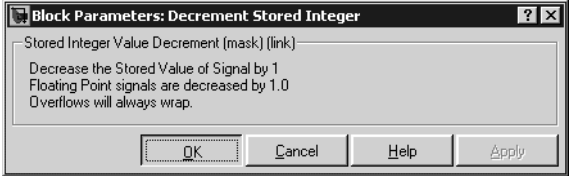
<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

**See Also** Dead Zone



<b>Purpose</b>	Decrease the real world value of the signal by one				
<b>Library</b>	Additional Math & Discrete / Additional Math: Increment - Decrement				
<b>Description</b>	The Decrement Real World block decreases the real world value of the signal by one. Overflows always wrap.				
					
<b>Data Type Support</b>	The Decrement Real World block accepts signals of any data type supported by Simulink, including fixed-point data types.				
<b>Parameters and Dialog Box</b>					
<b>Characteristics</b>	<table><tr><td>Direct Feedthrough</td><td>Yes</td></tr><tr><td>Scalar Expansion</td><td>No</td></tr></table>	Direct Feedthrough	Yes	Scalar Expansion	No
Direct Feedthrough	Yes				
Scalar Expansion	No				
<b>See Also</b>	Decrement Stored Integer, Decrement Time To Zero, Decrement To Zero, Increment Real World				

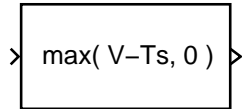
# Decrement Stored Integer

<b>Purpose</b>	Decrease the stored integer value of a signal by one				
<b>Library</b>	Additional Math & Discrete / Additional Math: Increment - Decrement				
<b>Description</b>	<div></div> <p>The Decrement Stored Integer block decreases the stored integer value of a signal by one.</p> <p>Floating-point signals are also decreased by one, and overflows always wrap.</p>				
<b>Data Type Support</b>	The Decrement Stored Integer block accepts signals of any data type supported by Simulink, including fixed-point data types.				
<b>Parameters and Dialog Box</b>	<div></div>				
<b>Characteristics</b>	<table><tr><td>Direct Feedthrough</td><td>Yes</td></tr><tr><td>Scalar Expansion</td><td>No</td></tr></table>	Direct Feedthrough	Yes	Scalar Expansion	No
Direct Feedthrough	Yes				
Scalar Expansion	No				
<b>See Also</b>	Decrement Real World, Decrement Time To Zero, Decrement To Zero, Increment Stored Integer				

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

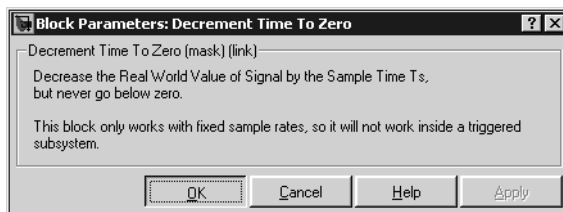
**Library** Additional Math & Discrete / Additional Math: Increment - Decrement

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



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

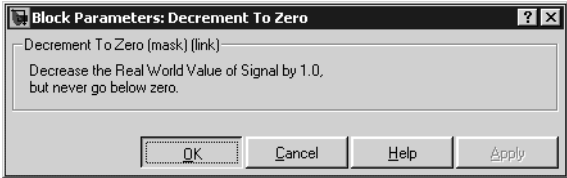
## Parameters and Dialog Box



<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	No

**See Also** Decrement Real World, Decrement Stored Integer, Decrement To Zero

# Decrement To Zero

Purpose	Decreases the real-world value of a signal by one, but only to zero.
Library	Additional Math & Discrete / Additional Math: Increment - Decrement
Description	<div>The Decrement To Zero block decreases the real-world value of the signal by one. The output will never go below zero.</div> <div><div><div>max( V--, 0 )</div></div></div>
Data Type Support	The Decrement To Zero block accepts signals of any data type supported by Simulink, including fixed-point data types.
Parameters and Dialog Box	<div></div>
Characteristics	<div>Direct Feedthrough    Yes</div> <div>Scalar Expansion      No</div>
See Also	Decrement Real World, Decrement Stored Integer, Decrement Time To Zero

**Purpose** Extract and output the elements of a bus or vector signal

**Library** Signal Routing

## Description



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

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

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

### Specifying the Number of Outputs in Vector Mode

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

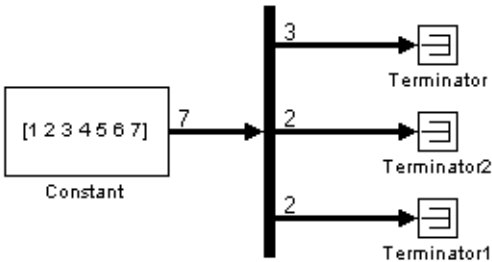
# Demux

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	
$p < n$ $n \bmod p = 0$	$p$ 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.
$p < n$ $n \bmod p = m$	$m$ 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.
$[p_1 \ p_2 \ \dots \ p_m]$ $p_1+p_2+\dots+p_m=n$ $p_i > 0$	$m$ vector signals having widths $p_1, p_2, \dots, p_m$	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
$[p_1 \ p_2 \ \dots \ p_m]$ $p_1 + p_2 + \dots + p_m = n$ some or all $p_i = -1$	m vector signals	If $p_i$ 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.
$[p_1 \ p_2 \ \dots \ p_m]$ $p_1 + p_2 + \dots + p_m \neq n$ $p_i = > 0$	Error	

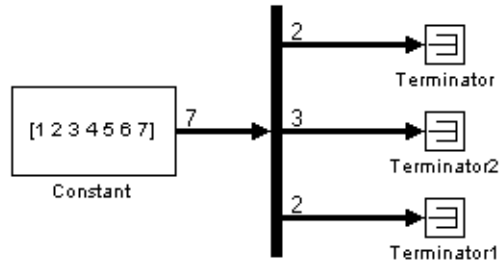
Note that you can specify the number of outputs as fewer than the number of input elements, in which case the block distributes the elements as evenly as possible over the outputs as illustrated in the following example.



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

If a vector expression comprises positive values and -1 values, the block assigns as many elements as needed to the ports with positive values and distributes the remain elements as evenly as possible over the ports with -1 values. For example, suppose that the block input is seven elements wide and you specify

the output as  $[-1, 3 \ -1]$ . In this case, the block outputs two elements on the first port, three elements on the second, and two elements on the third.

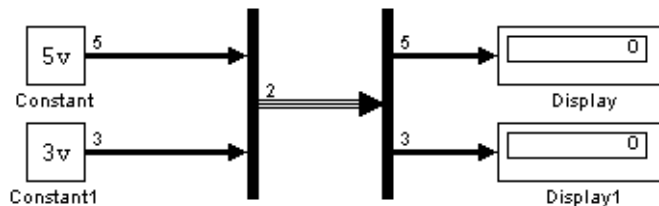


## Specifying the Number of Outputs in Bus Selection Mode

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

- Scalar specifying the number of output ports

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



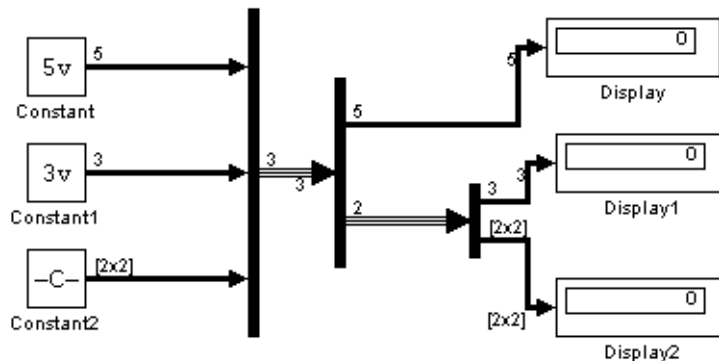
- Vector each of whose elements specifies the number of signals to output on the corresponding port

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

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



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



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

---

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

---

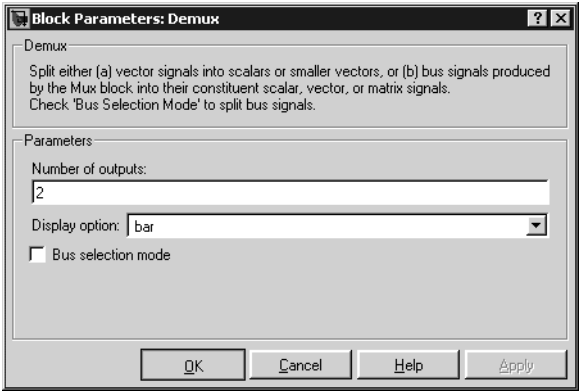
## Data Type Support

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

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

# Demux

## Parameters and Dialog Box



### Number of outputs

The number and dimensions of outputs.

### Display option

Options for displaying the Demux block. The options are

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

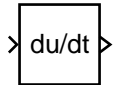
### Bus selection mode

Enable bus selection mode.

**Purpose** Output the time derivative of the input

**Library** Continuous

**Description** The Derivative block approximates the derivative of its input by computing



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

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

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

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

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

and taking the  $z$ -transform

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

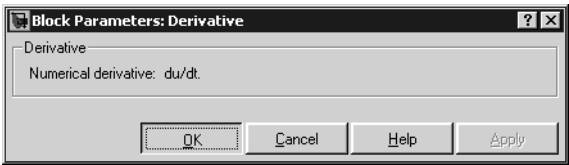
Using `linmod` to linearize a model that contains a Derivative block can be troublesome. For information about how to avoid the problem, see “Linearizing Models” in the Using Simulink documentation.

**Data Type Support** The Derivative block accepts and outputs a real signal of type double.

# Derivative

---

## Parameters and Dialog Box



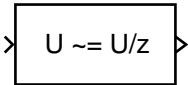
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Continuous
	Scalar Expansion	N/A
	States	2*[1+(number of input elements)]
	Dimensionalized	Yes
	Zero Crossing	No

**See Also**      Discrete Derivative

**Purpose** Detect a change in a signal’s value

**Library** Logic and Bit Operations

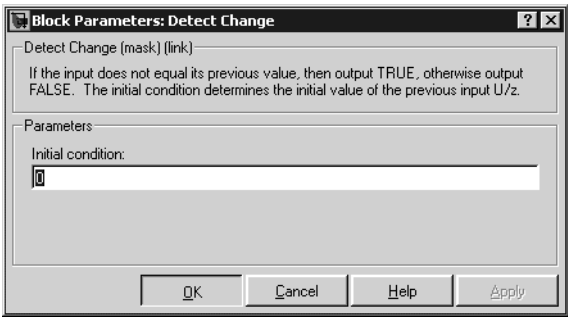
**Description** The Detect Change block determines if an input does not equal its previous value where



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

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

**Parameters and Dialog Box**



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

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

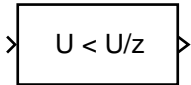
**See Also** Detect Decrease, Detect Fall Negative, Detect Fall Nonpositive, Detect Increase, Detect Rise Nonnegative, Detect Rise Positive

# Detect Decrease

**Purpose** Detect a decrease in a signal's value

**Library** Logic and Bit Operations

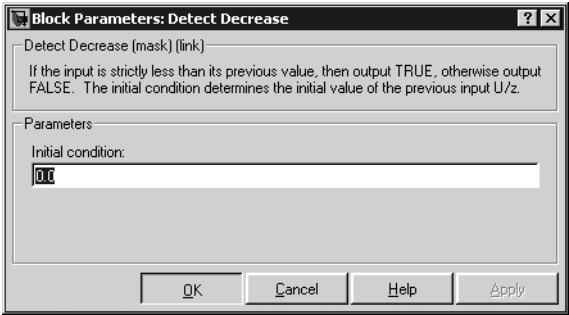
**Description** The Detect Decrease block determines if an input is strictly less than its previous value where



- The output is true (not 0), 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, including fixed-point data types.

**Parameters and Dialog Box**



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

**Characteristics**

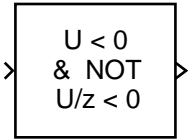
Direct Feedthrough	Yes
Scalar Expansion	Yes

**See Also** Detect Change, Detect Fall Negative, Detect Fall Nonpositive, Detect Increase, Detect Rise Nonnegative, Detect Rise Positive

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

**Library** Logic and Bit Operations

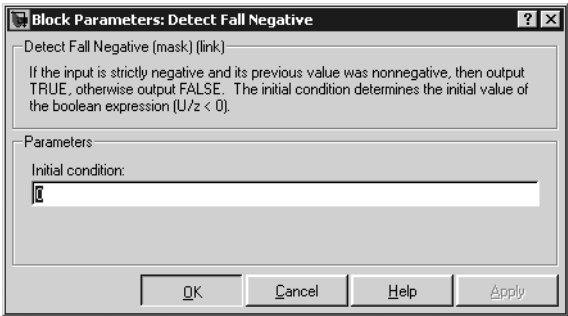
**Description** The Detect Fall Negative block determines if the input is less than zero, and its previous value was greater than or equal to zero where



- The output is true (not 0), 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 data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



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

**Characteristics**

Direct Feedthrough	Yes
Scalar Expansion	Yes

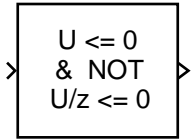
**See Also** Detect Change, Detect Decrease, Detect Fall Nonpositive, Detect Increase, Detect Rise Nonnegative, Detect Rise Positive

# Detect Fall Nonpositive

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

**Library** Logic and Bit Operations

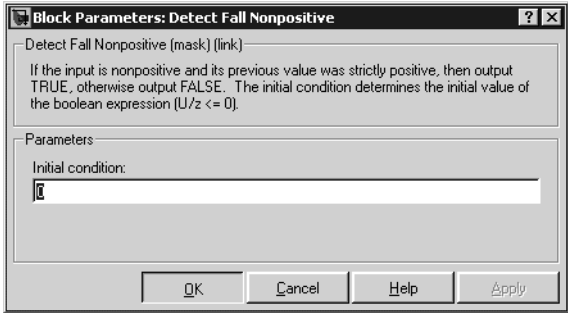
**Description** The Detect Fall Nonpositive block determines if the input is less than or equal to zero, and its previous value was positive where



- The output is true (not 0), 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 data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



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

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

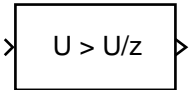
**See Also** Detect Change, Detect Decrease, Detect Fall Negative, Detect Increase, Detect Rise Nonnegative, Detect Rise Positive



**Purpose** Detect an increase in a signal's value

**Library** Logic and Bit Operations

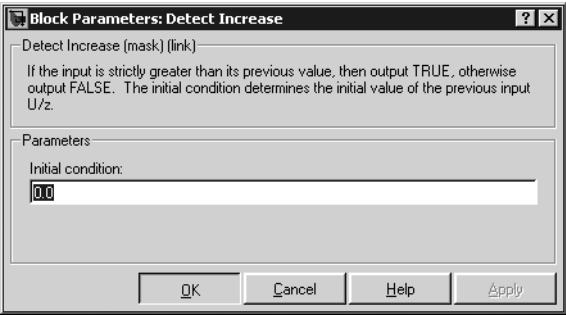
**Description** The Detect Increase block determines if an input is strictly greater than its previous value where



- The output is true (not 0), 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, including fixed-point data types.

**Parameters and Dialog Box**



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

**Characteristics**

Direct Feedthrough	Yes
Scalar Expansion	Yes

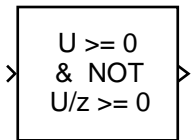
**See Also** Detect Change, Detect Decrease, Detect Fall Negative, Detect Fall Nonpositive, Detect Rise Nonnegative, Detect Rise Positive

# Detect Rise Nonnegative

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

**Library** Logic and Bit Operations

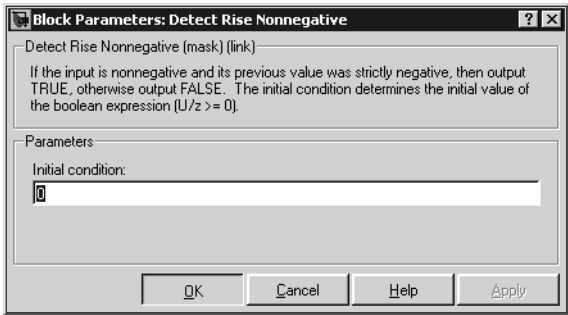
**Description** The Detect Rise Nonnegative block determines if the input is greater than or equal to zero, and its previous value was less than zero where



- The output is true (not 0), 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 data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box



**Initial condition** Set the initial condition of the Boolean expression  $U/z \geq 0$ .

**Characteristics**

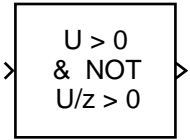
Direct Feedthrough	Yes
Scalar Expansion	Yes

**See Also** Detect Change, Detect Decrease, Detect Fall Negative, Detect Fall Nonpositive, Detect Increase, Detect Rise Positive

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

**Library** Logic and Bit Operations

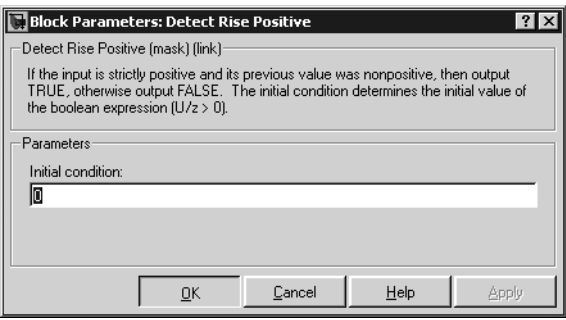
**Description** The Detect Rise Positive block determines if the input is strictly positive, and its previous value was nonpositive where



- The output is true (not 0), 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 data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



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

**Characteristics**

Direct Feedthrough	Yes
Scalar Expansion	Yes

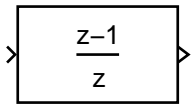
**See Also** Detect Change, Detect Decrease, Detect Fall Negative, Detect Fall Nonpositive, Detect Increase, Detect Rise Nonnegative

# Difference

**Purpose** Calculate the change in a signal over one time step

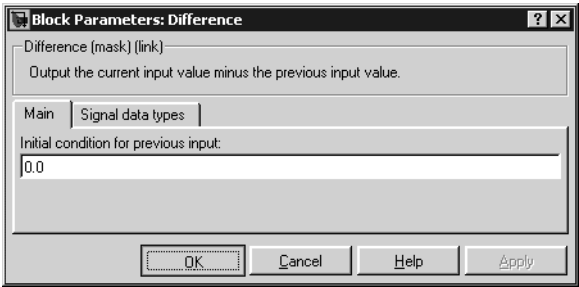
**Library** Discrete

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



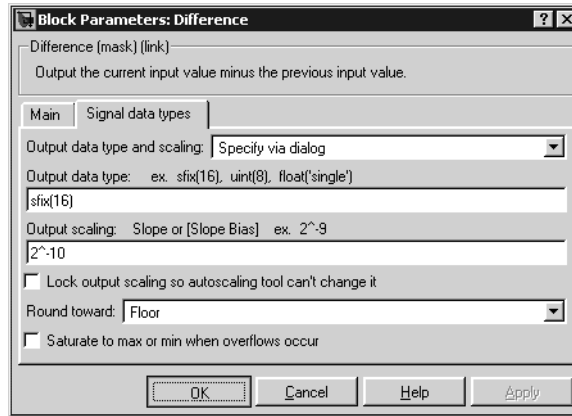
**Data Type Support** The Difference block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box** The **Main** pane of the Difference block dialog appears as follows:



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

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



## Output data type and scaling

Specify the output data type and scaling via the dialog box, or inherit the data type and scaling from an internal rule or by backpropagation.

## Output data type

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

## Output scaling

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

## Lock output scaling so autoscaling tool can't change it

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

## Round toward

Rounding mode for the fixed-point output.

## Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate.

## Characteristics

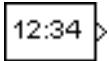
Direct Feedthrough	Yes
Scalar Expansion	Yes, of inputs and gain

# Digital Clock

**Purpose** Output simulation time at the specified sampling interval

**Library** Sources

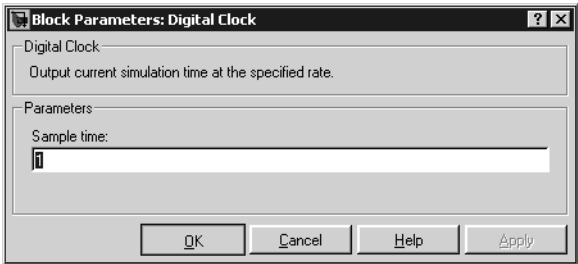
**Description** The Digital Clock block outputs the simulation time only at the specified sampling interval. At other times, the output is held at the previous value.



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 online documentation for more information.

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	No
	Dimensionalized	No
	Zero Crossing	No

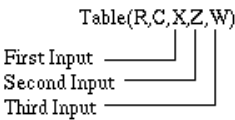
**Purpose** Index into an N-dimensional table to retrieve a scalar, vector, or 2-D matrix

**Library** Lookup Tables

**Description** The Direct Lookup Table (n-D) block uses its block inputs as zero-based indices into an n-D table. The number of inputs varies with the shape of the output desired. The output can be a scalar, a vector, 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.



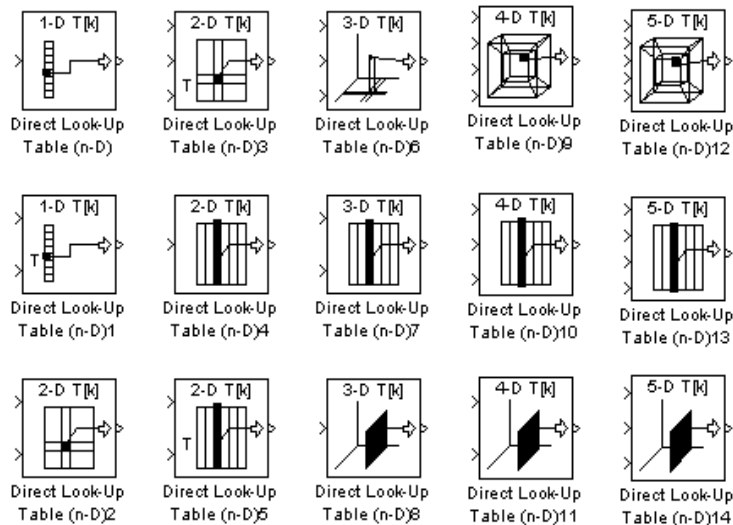
You define a set of output values as the **Table data** parameter. You specify what the output shape is: a scalar, a vector, or a 2-D matrix. The first input specifies the zero-based index to the first dimension higher than the number of dimensions in the output, the second input specifies the index to the next table dimension, and so on, as shown by this figure:



The figure shows a 5-D table with an output shape set to “2-D Matrix”; the output is a 2-D Matrix with R rows and C columns.

# Direct Lookup Table (n-D)

This figure shows the set of all the different icons that the Direct Lookup Table block shows (depending on the options you choose in the block's dialog box).



With dimensions higher than 4, the icon matches the 4-D icons, but shows the exact number of dimensions in the top text, e.g., “8-D T[k].” The top row of icons is used when the block output is made from one or more single-element lookups on the table. The blocks labeled “n-D Direct Table Lookup5,” 6, 8, and 12 are configured to extract a column from the table, and the two blocks ending in 7 and 9 are extracting a plane from the table. Blocks in the figure ending in 10, 11, and 12 are configured to have the table be an input instead of a parameter.

## Example

In this example, the block parameters are defined as

```
Invalid input value: "Clip and Warn"  
Output shape:       "Vector"  
Table data:         int16(a)
```

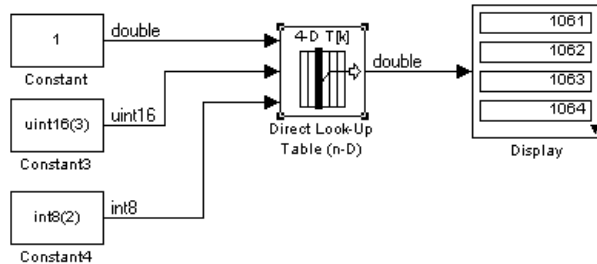
where *a* is a 4-D array of linearly increasing numbers calculated using MATLAB.

```
a = ones(20,4,5,7); L = prod(size(a));
```



```
a(1:L) = [1:L]';
```

The figure shows the block outputting a vector of the 20 values in the second column of the fourth element of the third dimension from the third element of the fourth dimension.



Note that the block uses zero-based indexing. The output values in this example can be calculated manually in MATLAB (which uses 1-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
```

# Direct Lookup Table (n-D)

1076  
1077  
1078  
1079  
1080

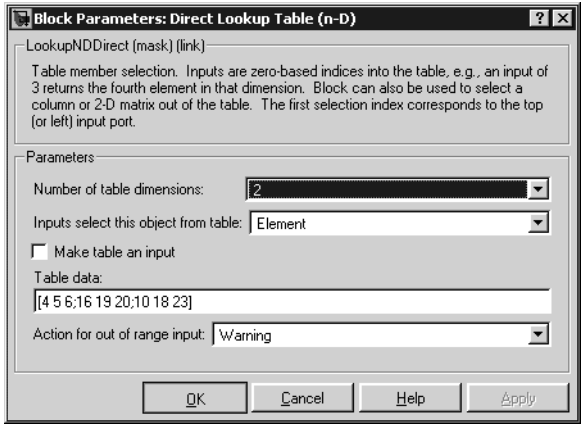
## Data Type Support

The Direct Lookup Table (n-D) block accepts mixed-type signals of data type supported by Simulink. For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.

The output type can differ from the input type and can be any of the types listed for input; the output type is inherited from the data type of the **Table data** parameter.

In the case that the table comes into the block on an input port, the output port type is inherited from the table input port. Inputs for indexing must be real; 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 determines the number of independent variables for the table and hence the number of inputs to the block. The options are 1, 2, 3, or more dimensions. If you choose more, the dialog box displays an edit field, **Explicit number of table dimensions**, that allows you to enter a number of dimensions.

## Explicit number of table dimensions

This field appears if you select more as the value of the **Number of table dimensions**. Enter the number of table dimensions in this field.

## Inputs select this object from table

Specify whether the output data is a single element, an n-D column, or a 2-D matrix. The number of ports changes for each selection:

Element — # of ports = # of dimensions

Column — # of ports = # of dimensions - 1

2-D matrix — # of ports = # of dimensions - 2

This numbering agrees with MATLAB indexing. For example, if you have a 4-D table of data, to access a single element you must specify four indices, as in `array(1,2,3,4)`. To specify a column, you need three indices, as in `array(:,2,3,4)`. Finally, to specify a 2-D matrix, you only need two indices, as in `array(:, :, 3, 4)`.

## Make table an input

Selecting this box forces the Direct Lookup Table (n-D) block to ignore the Table Data parameter. Instead, a new port appears with “T” next to it. Use this port to input table data.

## Table data

The table of output values. The matrix size must match the dimensions defined by the **N breakpoint set** parameter or by the **Explicit number of dimensions** parameter when the number of dimensions exceeds four. During block diagram editing, you can leave the **Table data** field empty, but for running the simulation, you must match the number of dimensions in the **Table data** to the **Number of table dimensions**. For information about how to construct multidimensional arrays in MATLAB, see [Multidimensional Arrays in the MATLAB online documentation](#). (This field appears only if **Make table an input** is not selected.)

## Action for out of range input

None, Warning, Error.

**Real-Time Workshop Note:** in the generated code, the “Clip and Warn” and “Clip Index” options cause Real-Time Workshop to generate clipping code with no code included to generate warnings. Code generated for the

# Direct Lookup Table (n-D)

---

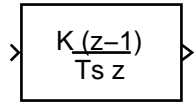
other option, “Generate Error”, has *no* clipping code or error messages at all, on the assumption that simulation during the design phase of your project should reveal model defects leading to out-of-range cases. This assumption helps the code generated by Real-Time Workshop to be highly efficient.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving blocks
	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)
	Zero Crossing	No

**Purpose** Compute a discrete time derivative

**Library** Discrete

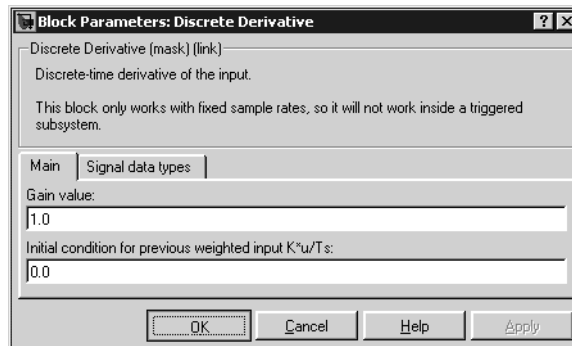
**Description** The Discrete Derivative block computes a discrete time derivative by subtracting the input value at the previous time step from the current value, and dividing by the sample time.



**Data Type Support** The Discrete Derivative block supports all Simulink data types, including fixed-point data types.

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

**Parameters and Dialog Box** The **Main** pane of the Discrete Derivative block dialog appears as follows:



## Gain value

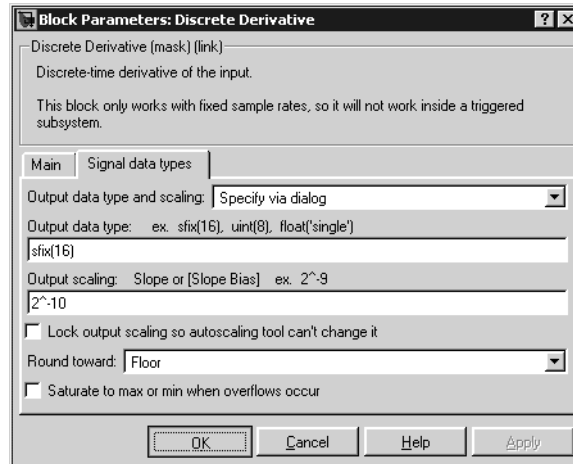
Specify the weight by which the sample time is multiplied.

## Initial condition for previous weighted input $K \cdot u / T_s$

Set the initial condition for the previous scaled input.

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

# Discrete Derivative



## Output data type and scaling

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

## Output data type

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

## Output scaling

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

## Lock output scaling so autoscaling tool can't change it

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

## Round toward

Select the rounding mode for fixed-point operations.

## Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes, of inputs and gain

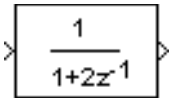
<b>See Also</b>	Derivative
-----------------	------------

# Discrete Filter

**Purpose** Implement IIR and FIR filters

**Library** Discrete

**Description** The Discrete Filter block implements Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) filters. You specify the coefficients of the numerator and denominator polynomials in ascending powers of  $z^{-1}$  as vectors using the **Numerator** and **Denominator** parameters. The order of the denominator must be greater than or equal to the order of the numerator. See Discrete Transfer Fcn on page 1-183 for more information about coefficients.

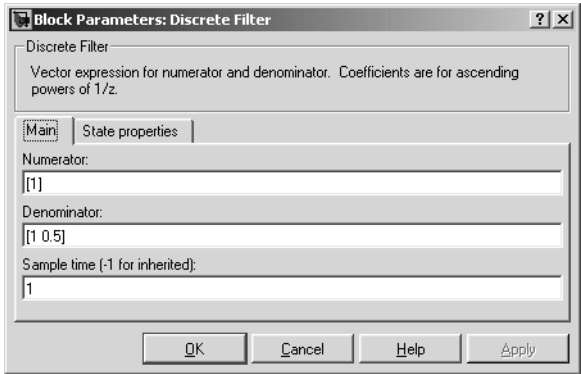


The Discrete Filter block represents the method often used by signal processing engineers, who describe digital filters using polynomials in  $z^{-1}$  (the delay operator). The Discrete Transfer Fcn block represents the method often used by control engineers, who represent a discrete system as polynomials in  $z$ . The methods are identical when the numerator and denominator are the same length. A vector of  $n$  elements describes a polynomial of degree  $n-1$ .

The block displays the numerator and denominator according to how they are specified. For a discussion of how Simulink displays the icon, see Transfer Fcn on page 1-496.

**Data Type Support** The Discrete Filter block accepts and outputs a real signal of type double.

## Parameters and Dialog Box





**Numerator**

The vector of numerator coefficients. The default is [ 1 ].

**Denominator**

The vector of denominator coefficients. The default is [ 1 2 ].

**Sample time**

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

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

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

# Discrete State-Space

**Purpose** Implement a discrete state-space system

**Library** Discrete

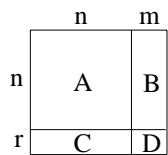
**Description** The Discrete State-Space block implements the system described by

$$\begin{aligned}y(n) &= Cx(n) + Du(n) \\ x(n+1) &= Ax(n) + Bu(n)\end{aligned}$$

$$x(n+1) = Ax(n) + Bu(n)$$
$$y(n) = Cx(n) + Du(n)$$

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

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

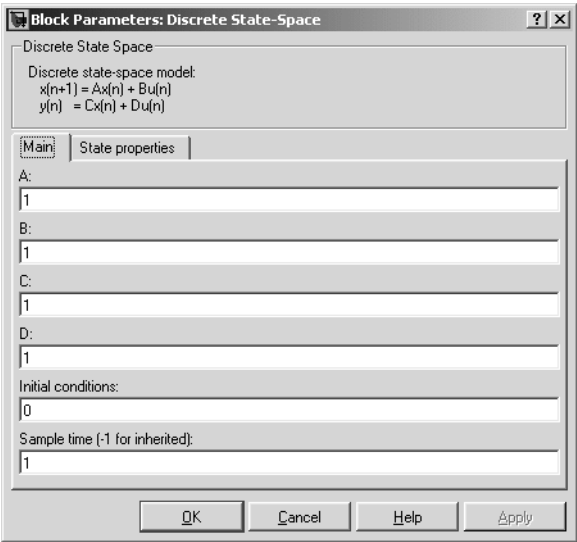


The block accepts one input and generates one output. The input vector width is determined by the number of columns in the B and D matrices. The output vector width is determined by the number of rows in the C and D matrices.

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

**Data Type Support** The Discrete State Space block accepts and outputs a real signal of type double.

Parameters and Dialog Box



A, B, C, D

The matrix coefficients, as defined in the preceding equations.

Initial conditions

The initial state vector. The default is 0.

Sample time

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

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

Characteristics

Direct Feedthrough	Only if $D \neq 0$
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes, of the initial conditions
States	Determined by the size of $A$

# Discrete State-Space

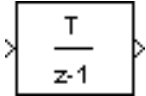
---

Dimensionalized	Yes
Zero Crossing	No

**Purpose** Perform discrete-time integration or accumulation of a signal

**Library** Discrete

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



The Discrete-Time Integrator block allows you to

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

These features are described below.

## Integration and Accumulation Methods

The block can integrate or accumulate using the Forward Euler, Backward Euler, and Trapezoidal methods. For a given step  $n$ , Simulink updates  $y(n)$  and  $x(n+1)$ .  $T$  is the sampling period (delta  $T$  in the case of triggered sampling time).  $K$  is the gain value. Values are clipped according to upper or lower limits.

---

**Note** For accumulation,  $T = 1$ .

---

- 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

$$y(n) = y(n-1) + K \cdot T \cdot u(n-1)$$

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

$$\begin{aligned} \text{Step 0:} \quad y(0) &= x(0) = \text{IC (clip if necessary)} \\ x(1) &= y(0) + K \cdot T \cdot u(0) \end{aligned}$$

$$\text{Step 1:} \quad y(1) = x(1)$$

# Discrete-Time Integrator

---

$$x(2) = x(1) + K \cdot T \cdot u(1)$$

$$\begin{aligned} \text{Step } n: \quad y(n) &= x(n) \\ x(n+1) &= x(n) + K \cdot T \cdot u(n) \quad (\text{clip if necessary}) \end{aligned}$$

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  $T \cdot z / (z-1)$ . The resulting expression for the output of the block at step  $n$  is

$$y(n) = y(n-1) + K \cdot T \cdot u(n)$$

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

$$\begin{aligned} \text{Step } 0: \quad y(0) &= x(0) = \text{IC (clipped if necessary)} \\ x(1) &= y(0) \end{aligned}$$

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

$$\begin{aligned} \text{Step } 0: \quad y(0) &= x(0) = \text{IC (clipped if necessary)} \\ x(1) &= y(0) = x(0) + K \cdot T \cdot u(0) \end{aligned}$$

$$\begin{aligned} \text{Step } 1: \quad y(1) &= x(1) + K \cdot T \cdot u(1) \\ x(2) &= y(1) \end{aligned}$$

$$\begin{aligned} \text{Step } n: \quad y(n) &= x(n) + K \cdot T \cdot u(n) \\ x(n+1) &= y(n) \end{aligned}$$

With this method, input port 1 has direct feedthrough.

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

$$T/2 \cdot (z+1) / (z-1)$$

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

$$x(n) = y(n-1) + K \cdot T/2 \cdot u(n-1)$$

The block uses the following steps to compute its output

$$\begin{aligned} \text{Step } 0: \quad y(0) &= x(0) = \text{IC (clipped if necessary)} \\ x(1) &= y(0) + K \cdot T/2 \cdot u(0) \end{aligned}$$

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

$$\begin{aligned}\text{Step 0:} \quad y(0) &= x(0) = \text{IC (clipped if necessary)} \\ x(1) &= y(0) = x(0) + K \cdot T/2 \cdot u(0)\end{aligned}$$

$$\begin{aligned}\text{Step 1:} \quad y(1) &= x(1) + K \cdot T/2 \cdot u(1) \\ x(2) &= y(1) + K \cdot T/2 \cdot u(1)\end{aligned}$$

$$\begin{aligned}\text{Step } n: \quad y(n) &= x(n) + K \cdot T/2 \cdot u(n) \\ x(n+1) &= y(n) + K \cdot T/2 \cdot u(n)\end{aligned}$$

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

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

$$\begin{aligned}\text{Step 0:} \quad y(0) &= x(0) = \text{IC (clipped if necessary)} \\ x(1) &= y(0)\end{aligned}$$

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

$$\begin{aligned}\text{Step 0:} \quad y(0) &= x(0) = \text{IC (clipped if necessary)} \\ x(1) &= y(0) = x(0) + K \cdot T/2 \cdot u(0)\end{aligned}$$

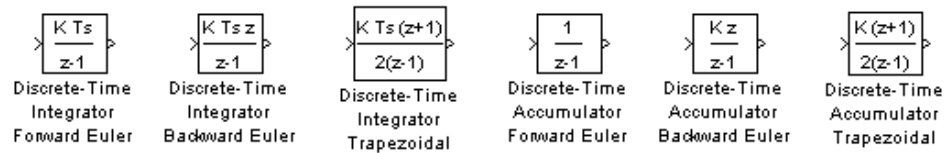
$$\begin{aligned}\text{Step 1:} \quad y(1) &= x(1) + T/2 \cdot (u(1) + u(0)) \\ x(2) &= y(1)\end{aligned}$$

$$\begin{aligned}\text{Step } n: \quad y(n) &= x(n) + T/2 \cdot (u(n) + u(n-1)) \\ x(n+1) &= y(n)\end{aligned}$$

With this method, input port 1 has direct feedthrough.

# Discrete-Time Integrator

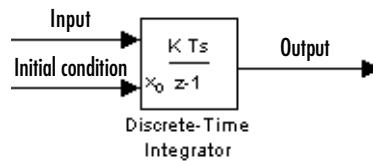
The block reflects the selected integration or accumulation method, as this figure shows.



## Defining Initial Conditions

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

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



## Using the State Port

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

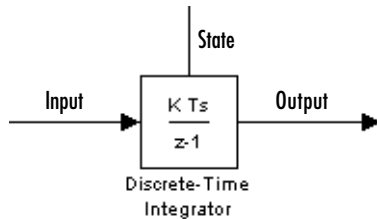
- When the output of the block is fed back into the block through the reset port or the initial condition port, causing an algebraic loop. For an example of this situation, see the bounce model.
- When you want to pass the state from one conditionally executed subsystem to another, which can cause timing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

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



these problems. You output the block state by selecting the **Show state port** check box.

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

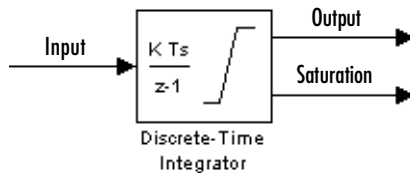


## Limiting the Integral

To prevent the output from exceeding specifiable levels, select the **Limit output** check box and enter the limits in the appropriate parameter fields. Doing so causes the block to function as a limited integrator. When the output reaches the limits, the integral action is turned off to prevent integral wind up. During a simulation, you can change the limits but you cannot change whether the output is limited. The output is determined as follows:

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

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



# Discrete-Time Integrator

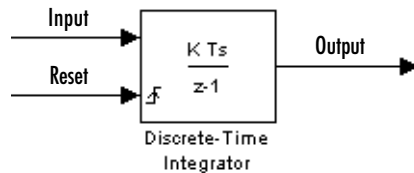
---

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.

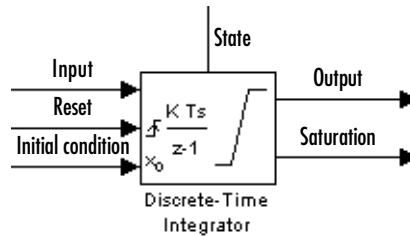


- Select rising to trigger the state reset when the reset signal has a rising edge.
- Select falling to trigger the state reset when the reset signal has a falling edge.
- Select either to trigger the reset when either a rising or falling signal occurs.
- Select level1 to trigger the reset and hold the output to the initial condition while the reset signal is nonzero.

The reset port has direct feedthrough. If the block output is fed back into this port, either directly or through a series of blocks with direct feedthrough, an algebraic loop results. 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.

## Choosing All Options

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

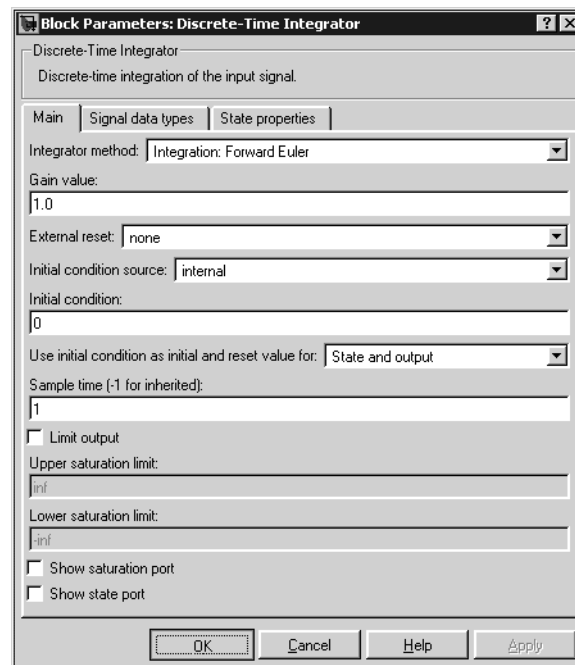


## Data Type Support

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

## Parameters and Dialog Box

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



# Discrete-Time Integrator

---

## Integrator method

Specify the integration or accumulation method.

## Gain value

Specify the weight by which the sample time is specified.

## External reset

Resets the states to their initial conditions when a trigger event occurs in the reset signal.

## Initial condition source

Gets the states' initial conditions from the **Initial condition** parameter (internal) or from an external block (external).

## Initial condition

The states' initial conditions. This parameter is only available if the **Initial condition source** parameter is set to internal.

## Use initial condition as initial and reset value for

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

$$y(0) = IC$$

$$x(0) = IC$$

or at reset

$$y(n) = IC$$

$$x(n) = IC$$

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

$$x(0) = IC$$

or at reset

$$x(n) = IC$$

## Sample time

The time interval between samples. The default is 1. See “Specifying Sample Time” in the online documentation for more information.

## Limit output

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

## Upper saturation limit

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

## Lower saturation limit

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

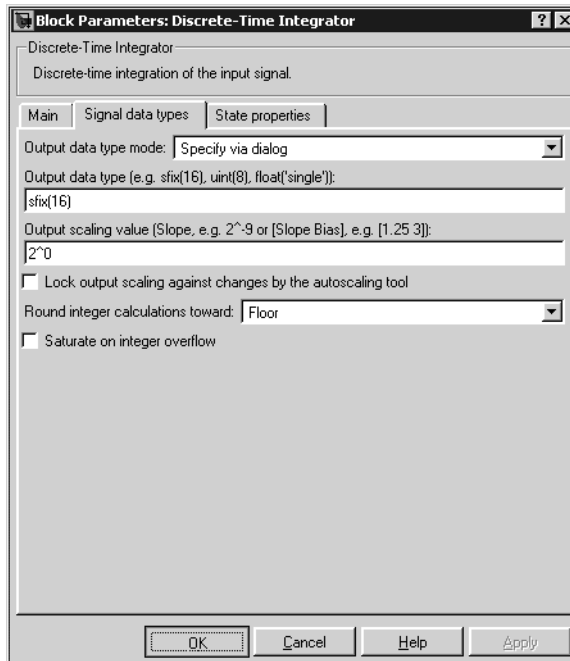
## Show saturation port

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

## Show state port

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

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



# Discrete-Time Integrator

---

## Output data type mode

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

## Output data type

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

## Output scaling value

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

## Lock output scaling against changes by the autoscaling tool

Select to lock scaling of outputs. This parameter is only visible if you select Specify via dialog for the **Output data type mode** parameter.

## Round integer calculations toward

Select the rounding mode for fixed-point operations.

## Saturate on integer overflow

Select to have overflows saturate.

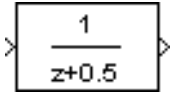
The **State properties** pane of this block pertains to code generation and has no effect on model simulation. See “Block States: Storing and Interfacing” in the Real-Time Workshop documentation for more information.

<b>Characteristics</b>	Direct Feedthrough	Yes, of the reset and external initial condition source ports
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of parameters
	States	Inherited from driving block and parameter
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Implement a discrete transfer function

**Library** Discrete

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



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

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

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

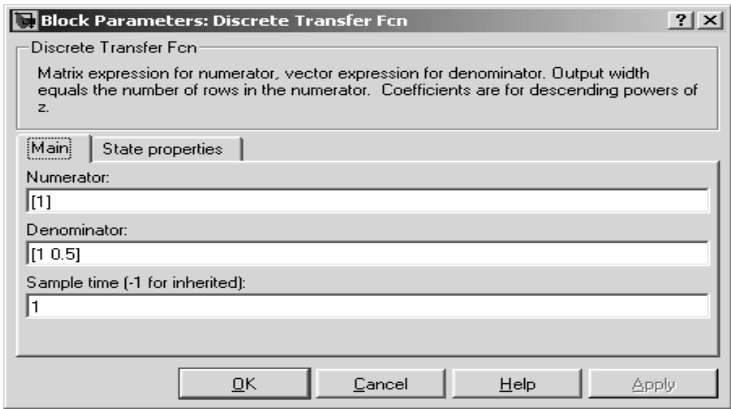
The Discrete Transfer Fcn block represents the method typically used by control engineers, representing discrete systems as polynomials in  $z$ . The Discrete Filter block represents the method typically used by signal processing engineers, who describe digital filters using polynomials in  $z^{-1}$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

The Discrete Transfer Fcn block displays the numerator and denominator within its icon depending on how they are specified. See Transfer Fcn on page 1-496 for more information.

**Data Type Support** The Discrete Transfer Function block accepts and outputs real signals of type double.

# Discrete Transfer Fcn

## Parameters and Dialog Box



### Numerator

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

### Denominator

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

### Sample time

The time interval between samples. The default is 1. See “Specifying Sample Time” in the online documentation for more information.

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

## Characteristics

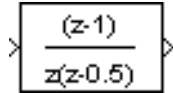
Direct Feedthrough	Only if the lengths of the <b>Numerator</b> and <b>Denominator</b> parameters are equal
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	No
States	Length of <b>Denominator</b> parameter -1
Dimensionalized	No
Zero Crossing	No



**Purpose** Implement a discrete transfer function specified in terms of poles and zeros

**Library** Discrete

**Description** The Discrete Zero-Pole block implements a discrete system with the specified zeros, poles, and gain in terms of the delay operator  $z$ . A transfer function can be expressed in factored or zero-pole-gain form, which, for a single-input, single-output system in MATLAB, is



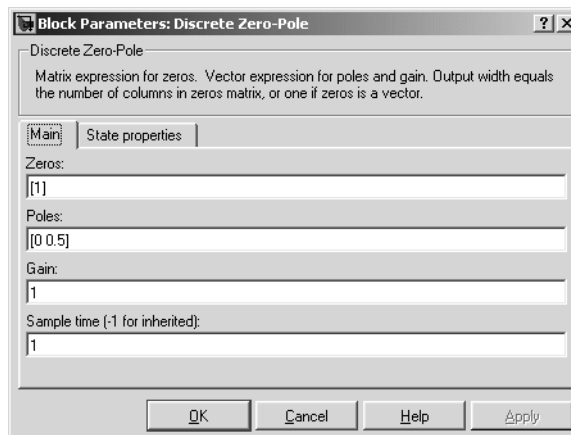
$$H(z) = K \frac{Z(z)}{P(z)} = K \frac{(z - Z_1)(z - Z_2) \dots (z - Z_m)}{(z - P_1)(z - P_2) \dots (z - P_n)}$$

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 on page 1-577 for more information.

**Data Type Support** The Discrete Zero-Pole block accepts and outputs real signals of type double.

## Parameters and Dialog Box



# Discrete Zero-Pole

---

**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 online documentation for more information.

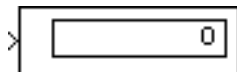
The **State properties** pane of this block pertains to code generation and has no effect on model simulation. See “Block States: Storing and Interfacing” in the Real-Time Workshop documentation for more information.

<b>Characteristics</b>	Direct Feedthrough	Yes, if the number of zeros and poles are equal
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	No
	States	Length of <b>Poles</b> vector
	Dimensionalized	No
	Zero Crossing	No

**Purpose** Show the value of the 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:

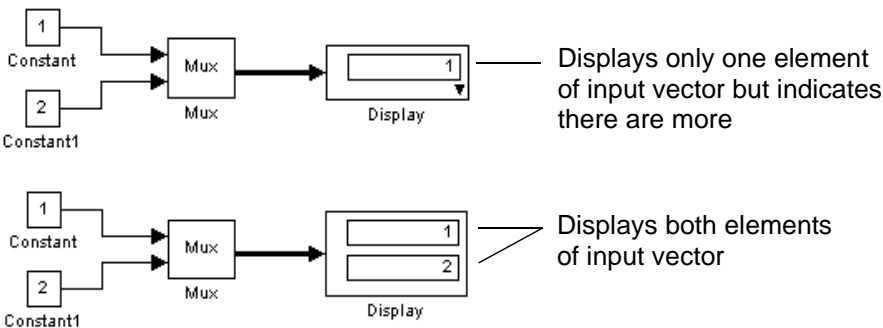
- **short** — displays a 5-digit scaled value with fixed decimal point
- **long** — displays a 15-digit scaled value with fixed decimal point
- **short\_e** — displays a 5-digit value with a floating decimal point
- **long\_e** — displays a 16-digit value with a floating decimal point
- **bank** — displays 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

The amount of data displayed and the time steps at which the data is displayed are determined by block parameters:

- 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.
- The **Sample time** parameter enables you to specify a sampling interval at which to display 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 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 top model shows the block before it is resized; notice the black triangle. The bottom model shows the resized block displaying both input elements.



## 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

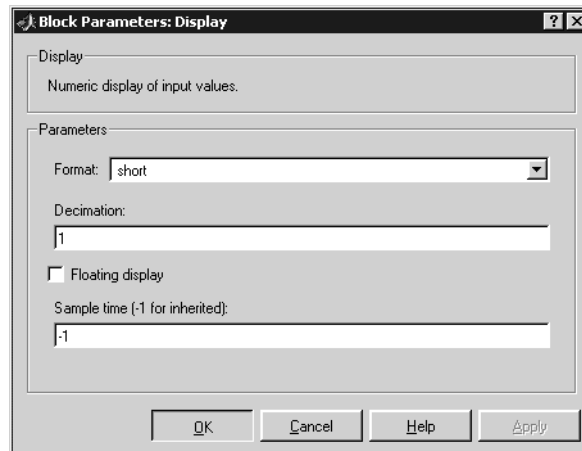
To use the block as a floating display, select the **Floating display** check box. The block's input port disappears and the block displays the value of the signal on a selected line. If you select the **Floating display** option, you must turn off the signal storage reuse feature in Simulink. See "Signal storage reuse" in the Using Simulink documentation for more information.

## Data Type Support

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

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

## Parameters and Dialog Box



### Format

Specify the format of the data displayed, as discussed in “Description” on page 1-187. 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.

### Sample time

Specify the sample time at which to display points. See “Specifying Sample Time” in the online documentation for more information.

## Characteristics

Sample Time	Specified in the <b>Sample time</b> parameter
Dimensionalized	Yes

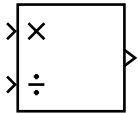
# Divide


---

**Purpose** Multiply or divide inputs

**Library** Math Operations

**Description** The Divide block is an implementation of the Product block. See Product on page 1-359 for more information.



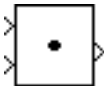
<b>Purpose</b>	Create text that documents the model and save the text with the model
<b>Library</b>	Model-Wide Utilities
<b>Description</b>	The DocBlock allows you to create and edit text that documents a model and save that text with the model. Double-clicking an instance of this block creates a temporary file containing the text associated with this block and opens the file in the text editor that you have selected in the MATLAB <b>Preferences</b> dialog box. Use the text editor to modify the text and save the file. Simulink stores the contents of the saved file in the model file.
	
<b>Data Type Support</b>	Not applicable.
<b>Parameters and Dialog Box</b>	The DocBlock does not have a parameter dialog box.
<b>Characteristics</b>	Not applicable

# Dot Product

**Purpose** Generate the dot product

**Library** Math Operations

**Description** The Dot Product block generates the dot product of its two input vectors. The scalar output,  $y$ , is equal to the MATLAB operation



$$y = \text{sum}(\text{conj}(u1) \text{ .* } u2)$$

where  $u1$  and  $u2$  represent the vector inputs. If both inputs are vectors, they must be the same length. 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

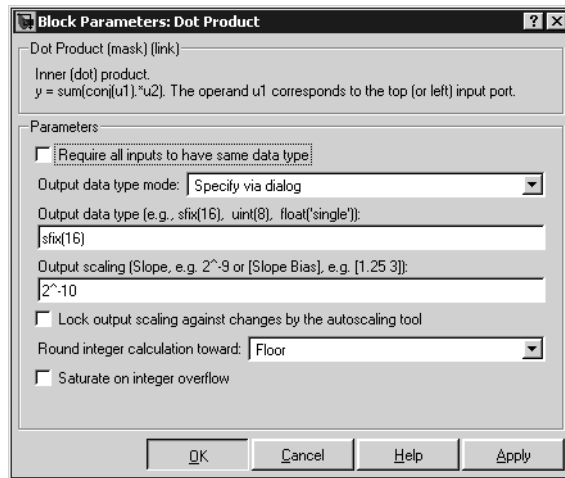
To perform element-by-element multiplication without summing, use the Product block.

**Data Type Support** The Dot Product block accepts and outputs signals of any data type supported by Simulink, including fixed-point data types.

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



## Parameters and Dialog Box



### Require all inputs to have same data type

Select to require all inputs to have the same data type.

### Output data type mode

Set the data type and scaling of the output to be the same as that of the first input, or to be inherited via an internal rule or by backpropagation.

Alternatively, choose to specify the data type and scaling of the output through the **Output data type** and **Output scaling value** parameters.

### Output data type

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

### Output scaling

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

### Lock output scaling against changes by the autoscaling tool

Select to lock scaling of outputs. This parameter is only visible if you select Specify via dialog for the **Output data type mode** parameter.

# Dot Product

---

**Round integer calculations toward**

Select the rounding mode for fixed-point operations.

**Saturate on integer overflow**

Select to have overflows saturate.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	States	0
	Dimensionalized	Yes
	Zero Crossing	No

## Purpose

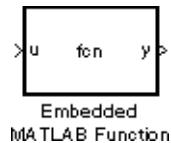
Include MATLAB code in models that generate embeddable C code.

## Library

User-Defined Functions

## Description

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



The screenshot shows the 'Embedded MATLAB Editor' window for a block named 'call\_stats\_block/Embedded MATLAB'. The window has a menu bar (File, Edit, Text, Debug, Tools, Window, Help) and a toolbar. The main text area contains the following MATLAB code:

```
1 function [mean,stdev] = stats(vals)
2
3 % calculates a statistical mean and a standard
4 % deviation for the values in vals.
5
6 len = length(vals);
7 mean = avg(vals,len);
8 stdev = sqrt(sum(((vals-avg(vals,len)).^2)/len));
9 plot(vals,'-+');
10
11 function mean = avg(array,size)
12 mean = sum(array)/size;
```

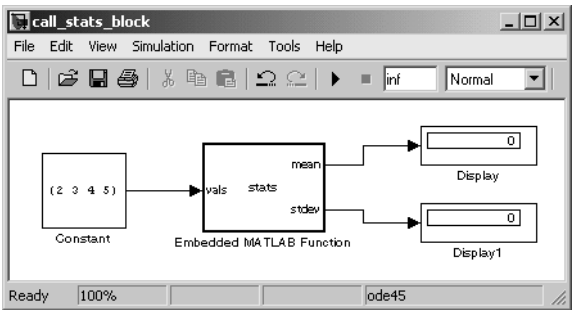
The status bar at the bottom indicates 'Ready' and 'Ln 12 Col 24'.

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

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

# Embedded MATLAB Function

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



The Embedded MATLAB Function block supports a subset of the language for which it can generate efficient embeddable code. The following table gives a high-level overview of its capabilities with links to more detailed information.

Supported MATLAB Features	Unsupported MATLAB Features
Two-Dimensional Arrays	N-Dimensional Arrays
Matrix operations(+, -, *, ....)	Matrix Deletion (X(1) = []) Logical Indexing
Complex Numbers	Sparse Matrices
Double/Single Math	Integer Math
if/switch/while/for	try-catch
Numeric Types	Cell Arrays, Structures, Java, User-Defined Classes
Subfunctions	Calling out to functions on the path (except for simulation)
persistent	global
Simulink Parameters as Inputs	Command Duality

See the “Embedded MATLAB Function Block Reference” for full details.

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 “Limitations on Indexing Operations” and “Limitations with Complex Numbers” in Simulink 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 Run-Time Function Library”. 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 Run-Time Library - by Category” for a complete list of function categories.

---

**Note** Although Embedded MATLAB attempts to produce exactly the same results as MATLAB, 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.

---

# Embedded MATLAB Function

---

To support visualization of data, Embedded MATLAB Function blocks support calls to MATLAB functions for simulation only. See “Calling MATLAB Functions” in Simulink 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 Argument and Return Variables”, “Sizing Function Argument and Return Variables”, and “Parameter Arguments in Embedded MATLAB Functions” for more detailed descriptions of variables that you use in Embedded MATLAB Functions.

## Data Type Support

The Embedded MATLAB Function block accepts inputs of any type supported by Simulink, except fixed-point types. For a discussion on the variable types supported by Embedded MATLAB functions in Simulink, refer to “Data Types Supported by Simulink” in the Simulink documentation.

Math with double and single types is supported. Integer math is not.

Simulink frames are not supported. However, you can use the Rate Transition block to convert frames into vectors.

## 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, CodeReuse Subsystem blocks for an identification of each field.

## Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	No

**Purpose** Add an enabling port to a subsystem

**Library** Ports & Subsystems

**Description** Adding an Enable block to a subsystem makes it an enabled subsystem. An enabled subsystem executes while the input received at the Enable port is greater than zero.



At the start of simulation, Simulink initializes the states of blocks inside an enabled subsystem to their initial conditions. When an enabled subsystem restarts (executes after having been disabled), the **States when enabling** parameter determines what happens to the states of blocks contained in the enabled subsystem:

- **reset** resets the states to their initial conditions (zero if not defined).
- **held** holds the states at their previous values.

You can output the enabling signal by selecting the **Show output port** check box. Selecting this option allows the system to process the enabling signal.

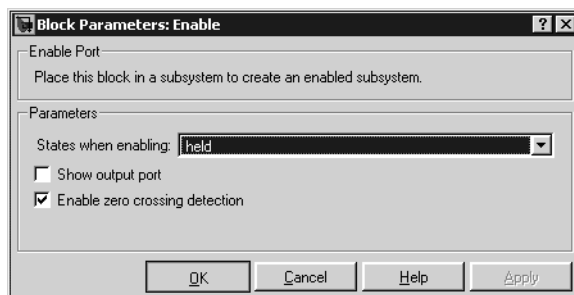
A subsystem can contain no more than one Enable block.

**Data Type Support** The data type of the input of the Enable port can be any data type supported by Simulink, including fixed-point data types.

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

See “Creating Conditionally Executed Subsystems” in the online Simulink help for more information about enabled subsystems.

## Parameters and Dialog Box



# Enable

---

**States when enabling**

Specifies how to handle internal states when the subsystem becomes reenabled.

**Show output port**

If selected, Simulink draws the Enable block output port and outputs the enabling signal.

**Enable zero crossing detection**

Select to enable zero crossing detection. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

**Characteristics**

Sample Time	Determined by the signal at the enable port
Dimensionalized	Yes

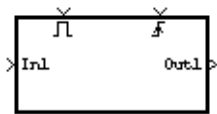


# Enabled and Triggered Subsystem

**Purpose** Represent a 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 Subsystem” in the online Simulink help.



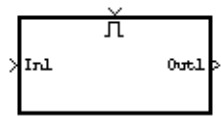
# Enabled Subsystem

---

**Purpose** Represent a subsystem whose execution is enabled by external input

**Library** Ports & Subsystems

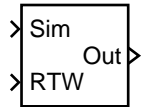
**Description** This block is a Subsystem block that is preconfigured to serve as the starting point for creating an enabled subsystem. For more information, see “Enabled Subsystems” in the Using Simulink documentation.



**Purpose** Create branches of a 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. If you enable block reduction optimization (see “Block reduction optimization” in the online Simulink documentation), Simulink eliminates the branch connected to the block's RTW port when compiling the model for simulation.

**Data Type Support** The Environment Controller block accepts signals of any numeric or data type. It outputs the type at its input.

## Parameters and Dialog Box



# Extract Bits

---

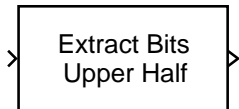
## Purpose

Output a selection of contiguous bits from the 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. The **Bits to extract** parameter defines the method by which you select the output bits.

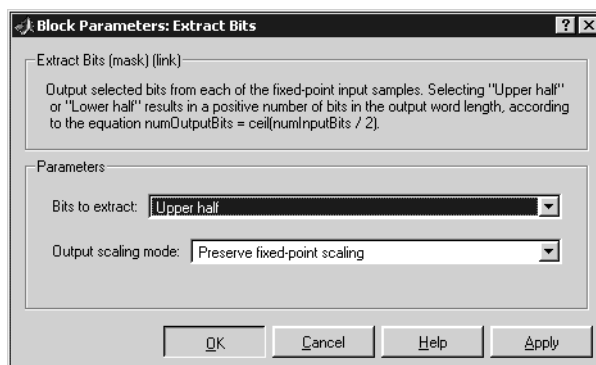
- Select `Upper half` to output the half of the input bits that contain the most significant bit. If there is an odd number of bits in the input signal, the number of output bits is given by the equation
$$\text{number of output bits} = \text{ceil}(\text{number of input bits}/2)$$
- Select `Lower half` 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 data type supported by Simulink, including fixed-point data types. Floating-point inputs are passed through the block unchanged. Boolean inputs are treated as `uint8` signals.

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

## Parameters and Dialog Box



### Bits to extract

Select the mode by which to extract bits from the input signal, as discussed in “Description” on page 1-204.

### Number of bits

(Not shown on dialog above.) Select the number of bits to output from the input signal.

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.

# Extract Bits

---

- 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.

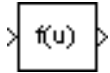
## Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited
Scalar Expansion	N/A
States	None
Dimensionalized	Inherited
Zero Crossing	N/A

**Purpose** Apply a specified expression to the input

**Library** User-Defined Functions

**Description** The Fcn block applies the specified C language style 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.
- 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).

---

**Note** The Fcn block does not support tunable parameters, i.e., it ignores changes in the values of workspace variables during simulation. The Fcn block also does not support Custom Storage Classes (see the Real-Time Workshop documentation for more information).

---

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 (:).

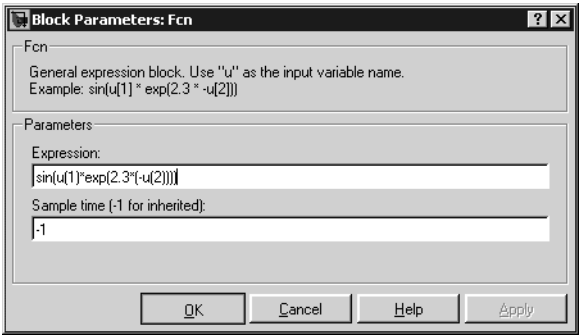
Block input can be a scalar or vector. The output is always a scalar. For vector output, consider using the Math Function block. If a block is a vector and the function operates on input elements individually (for example, the sin function), the block operates on only the first vector element.

**Note** Simulink does not allow you to change the value of the block's **Expression** parameter while running a model in accelerated mode (see “The Simulink Accelerator”). Furthermore, Simulink does not update the value of the Fcn expression to reflect changes in the workspace while running in accelerated mode.

Data Type Support

The Fcn block accepts and outputs signals of type double.

Parameters and Dialog Box





**Expression**

The C language style 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.).

**Sample time (-1 for inherited)**

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

**Characteristics**

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

# First-Order Hold

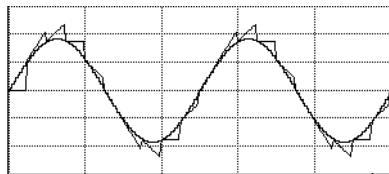
**Purpose** Implement a first-order sample-and-hold

**Library** Discrete

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



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



**Data Type Support** The First-Order Hold block accepts and outputs signals of type double.

## Parameters and Dialog Box



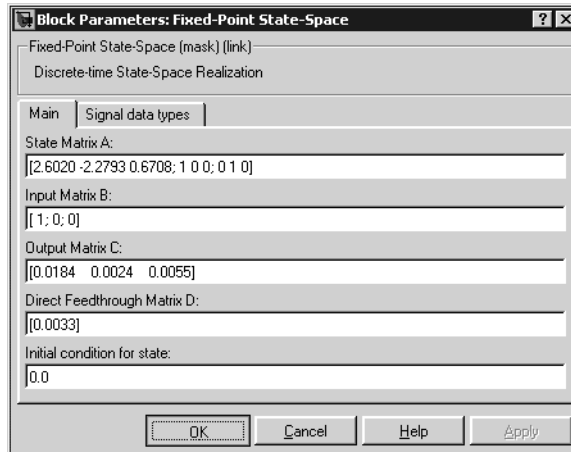
### Sample time

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

Characteristics	Direct Feedthrough	No
	Sample Time	Specified in the <b>Sample time</b> 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	<div><div><div><div>y(n)=Cx(n)+Du(n) x(n+1)=Ax(n)+Bu(n)</div></div></div></div> <p>The Fixed-Point State-Space block implements the system described by</p> $y(n) = Cx(n) + Du(n)$ $x(n+1) = Ax(n) + Bu(n)$ <p>where <math>u</math> is the input, <math>x</math> is the state, and <math>y</math> is the output. Both equations have the same data type.</p> <p>The matrices A, B, C and D have the following characteristics:</p> <ul style="list-style-type: none"><li>• A must be an n-by-n matrix, where n is the number of states.</li><li>• B must be an n-by-m matrix, where m is the number of inputs.</li><li>• C must be an r-by-n matrix, where r is the number of outputs.</li><li>• D must be an r-by-m matrix.</li></ul> <p>In addition:</p> <ul style="list-style-type: none"><li>• The state <math>x</math> must be a n-by-1 vector</li><li>• The input <math>u</math> must be a m-by-1 vector</li><li>• The output <math>y</math> must be a r-by-1 vector</li></ul> <p>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.</p>
Data Type Support	The Fixed-Point State-Space block accepts signals of any data type supported by Simulink, including fixed-point data types.
Parameters and Dialog Box	The <b>Main</b> pane of the Fixed-Point State-Space block dialog appears as follows:



## State Matrix A

Specify the matrix of states.

## Input Matrix B

Specify the column vector of inputs.

## Output Matrix C

Specify the column vector of outputs.

## Direct Feedthrough Matrix D

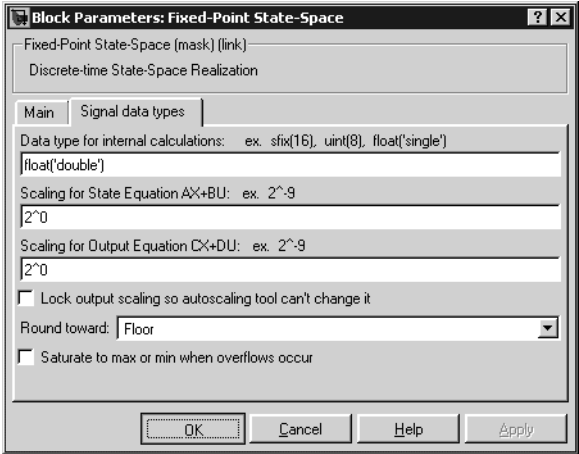
Specify the matrix for direct feedthrough.

## Initial condition for state

Specify the initial condition for the state.

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

# Fixed-Point State-Space



### Data type for internal calculations

Specify the data type for internal calculations.

### Scaling for State Equation $AX+BU$

Specify the scaling for state equations.

### Scaling for Output Equation $CX+DU$

Specify the scaling for output equations.

### Lock output scaling so autoscaling tool can't change it

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

### Round toward

Select the rounding mode for fixed-point operations.

### Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes, of initial conditions

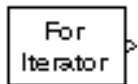
## Purpose

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

## Library

Ports & Subsystems/For Iterator Subsystem

## Description

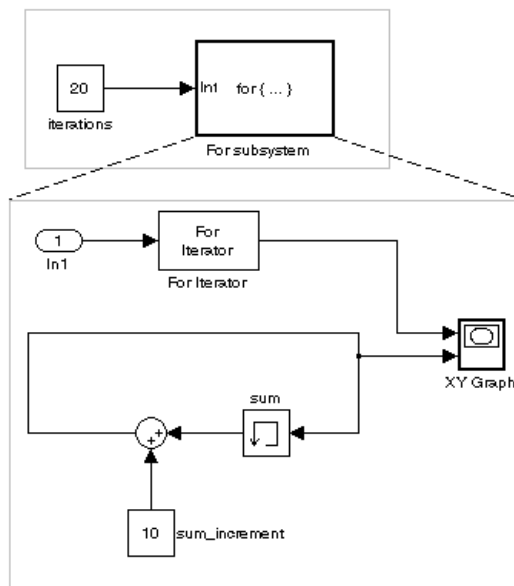


The For Iterator block, when placed in a subsystem, repeatedly executes the contents of the subsystem at the current time step until an iteration variable exceeds a specified iteration limit. You can use this block to implement the block diagram equivalent of a for loop in the C programming language.

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

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

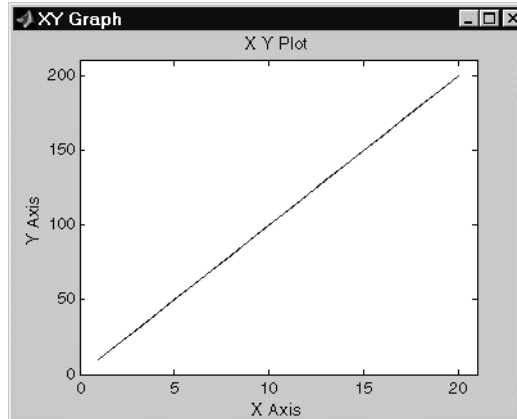
The model in the following figure uses a For Iterator block to increment an initial value of zero by 10 over 20 iterations at every time step.



# For Iterator

---

The following figure shows the result.



Points:  
(1,10)  
(2,20)  
etc.

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 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.



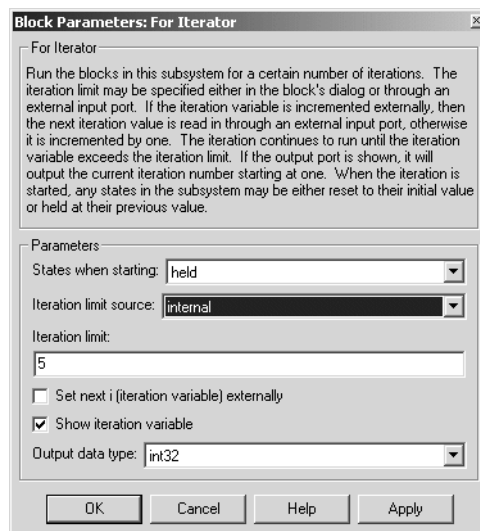
- If no output port is specified, the input port value is cast to type int32.
- If the input port value exceeds the maximum value of the output port's type, it is truncated to that maximum value.

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 **Number of iterations** field determines the number of iterations. If you set this field to external, the

# For Iterator

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 for the For Iterator block to this value. This field appears only if you selected internal for the **Source of number of iterations** field.

## 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.

## Output data type

Set the type for the iteration value output from the iteration number port to double, int32, int16, or int8.

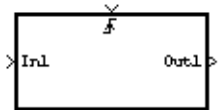
## Characteristics

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

**Purpose** Represent a subsystem that executes repeatedly during a 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 and “Control Flow Blocks” in the Using Simulink documentation.



# From

**Purpose** Accept input from a Goto block

**Library** Signal Routing

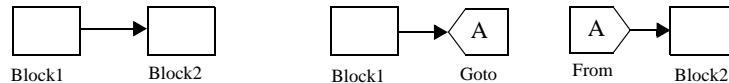
## Description



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.



Associated Goto and From blocks can appear anywhere in a model, with this exception: if either block is in a conditionally executed subsystem, the other block must be either in the same subsystem or in a subsystem below it in the model hierarchy (but not in another conditionally executed subsystem).

However, if a Goto block is connected to a state port, the signal can be sent to a From block inside another conditionally executed subsystem. For more information about conditionally executed subsystems, see “Creating Conditionally Executed Subsystems” in the Using Simulink documentation.

The visibility of a Goto block tag determines the From blocks that can receive its signal. For more information, see Goto on page 1-237 and Goto Tag Visibility on page 1-240. The block indicates the visibility of the Goto block tag:

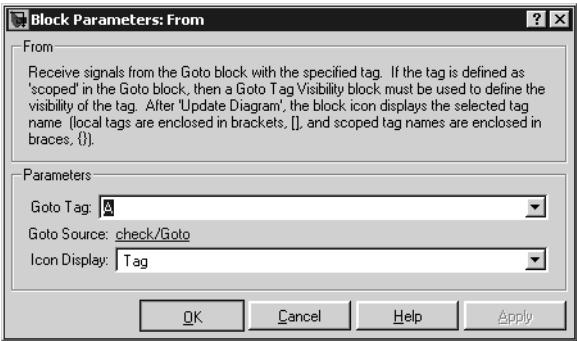
- A local tag name is enclosed in brackets ([]).
- A scoped tag name is enclosed in braces ({}).
- A global tag name appears without additional characters.

**Data Type Support**

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

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

**Parameters and Dialog Box**



**Goto tag**

The tag of the Goto block passing the signal to this From block.

**Goto source**

Path of the Goto block connected to this From block. Double-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

# From File

**Purpose** Read data from a MAT file.

**Library** Sources

**Description** The From File block outputs data read from a MAT file. Its icon displays the pathname of the file supplying the data.



**Note** The From block can read data only from MAT files. It does not support any other file format.

The MAT file must contain a matrix of two or more rows. The first row must contain monotonically increasing time points. Other rows contain data points that correspond to the time point in that column. The matrix is expected to have 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 to determine its output, but does not output the time values. This means that in a matrix containing  $m$  rows, the block outputs a vector of length  $m-1$ , consisting of data from all but the first row of the appropriate column.

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 extrapolates, using the first two or last two 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 column encountered. For example, for a matrix that has this data:

time values:      0 1 2 2

data points:     2 3 4 5

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

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

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

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

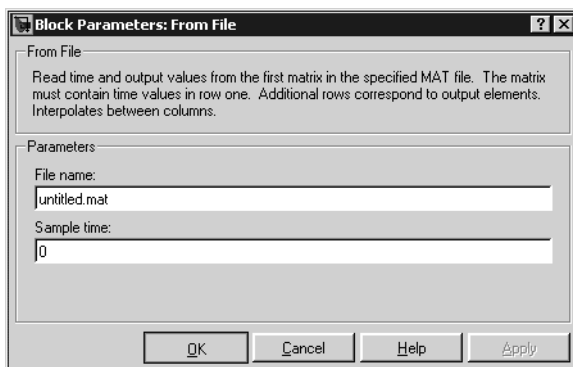
To read data written by a To Workspace block and saved to a MAT file:

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

## Data Type Support

The From File block outputs real signals of type double.

## Parameters and Dialog Box



### File name

The fully qualified pathname or file name of the MAT file that contains the data used as input. On UNIX, the pathname can start with a tilde (~)

# From File

---

character signifying your home directory. The default file name is `untitled.mat`. If you specify an unqualified file name, Simulink assumes that the MAT file resides in the MATLAB working directory. (To determine the working directory, enter `pwd` at the MATLAB command line.) If Simulink cannot find the specified file name in the working directory, it displays an error message.

## Sample time

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

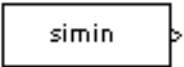
Characteristics	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	No
	Dimensionalized	1-D array only
	Zero Crossing	No



**Purpose** Read data from the workspace

**Library** Sources

**Description** The From Workspace block reads data from the MATLAB workspace. The block’s **Data** parameter specifies the workspace data via a MATLAB expression that evaluates to a matrix (2-D array) or a structure containing an array of signal values and time steps. The format of the matrix or structure is the same as that used to load input data from the workspace (see “Loading Input from the Base Workspace”). The From Workspace icon displays the expression in the **Data** parameter.



**Note** You must use the structure-with-time format to load matrix (2-D) data from the workspace. You can use either the array or the structure format to load scalar or vector (1-D) data.

The From Workspace block’s **Interpolate data** parameter determines the block’s output in the time interval for which workspace data is supplied. If you select the **Interpolate data** option, the block interpolates between data values for time steps that occur between the times for which data is supplied from the workspace. Otherwise, the block uses the most recent data value supplied from the workspace.

The block’s **Form output after final data value by** parameter determines the block’s output after the last time step for which data is available from the workspace. The following table summarizes the output block based on the options that the parameter provides.

Form Output Option	Interpolate Option	Block Output After Final Data
Extrapolate	On	Extrapolated from final data value
Extrapolate	Off	Error
SettingToZero	On	Zero
SettingToZero	Off	Zero

# From Workspace

Form Output Option	Interpolate Option	Block Output After Final Data
HoldingFinalValue	On	Final value from workspace
HoldingFinalValue	Off	Final value from workspace
CyclicRepetition	On	Error
CyclicRepetition	Off	Repeated from workspace. This option is valid only for workspace data in structure-without-time format.

If the input array contains more than one entry for the same time step, Simulink uses the signals specified by the last entry. For example, suppose the input array has this data:

```
time:      0 1 2 2
signal:    2 3 4 5
```

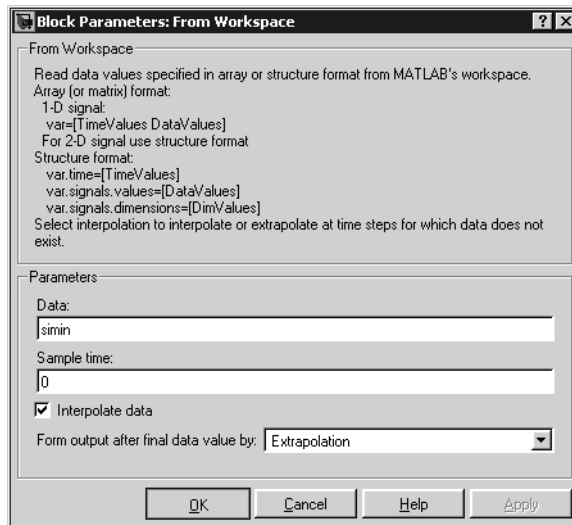
At time 2, the output is 5, the signal value for the last entry for time 2.

**Note** A From Workspace block can directly read the output of a To Workspace block (see To Workspace on page 1-492) if the output is in structure-with-time format (see “Loading Input from the Base Workspace” for a description of these formats).

## Data Type Support

The From Workspace block accepts real or complex signals of any type supported by Simulink. 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.

## 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 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 “Specifying 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.

### 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.

# From Workspace

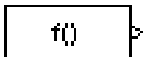
---

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Execute a function-call subsystem a specified number of times at a specified rate

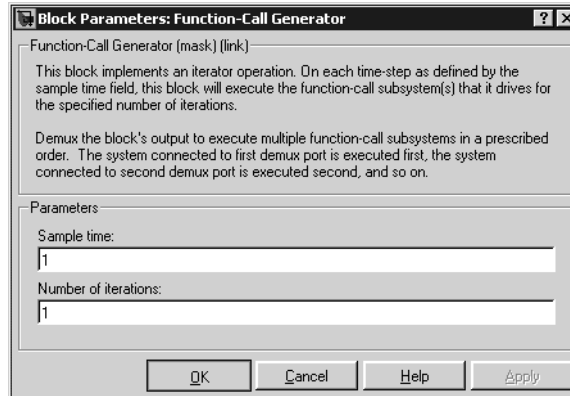
**Library** Ports & Subsystems

**Description** The Function-Call Generator block executes a function-call subsystem (for example, a Stateflow<sup>®</sup> state chart configured as a function-call system) at the rate specified by the block's **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 be controlled. Then connect the outputs of the Demux block to the systems to be controlled. 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 real signal of type double.

## Parameters and Dialog Box



### Sample time

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

### Number of iterations

Number of times to execute the block per time step.

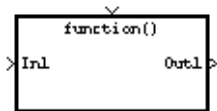
# Function-Call Generator

Characteristics	Direct Feedthrough	No
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Represent a subsystem that can be invoked as a 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 Using Simulink documentation.

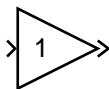


# Gain

**Purpose** Multiply the input by a 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.



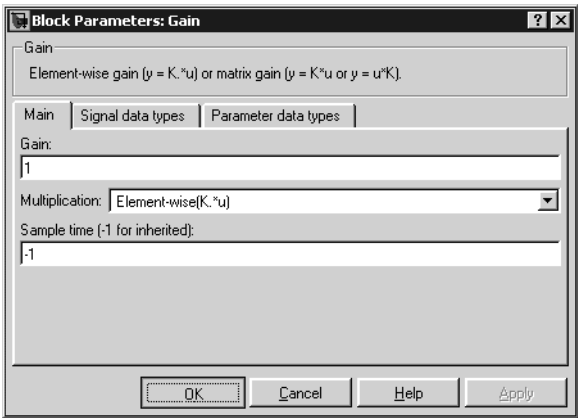
You specify the value of the gain in the **Gain** parameter. The **Multiplication** parameter lets you specify element-wise or matrix multiplication. For matrix multiplication, this parameter also lets you indicate the order of the multiplicands.

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 data type supported by Simulink except Boolean. 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, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.

**Parameters and Dialog Box** The **Main** pane of the Gain block dialog appears as follows:





## Gain

Specify the value by which to multiply the input. The gain may be a scalar, vector, or matrix. The gain may not be Boolean.

## Multiplication

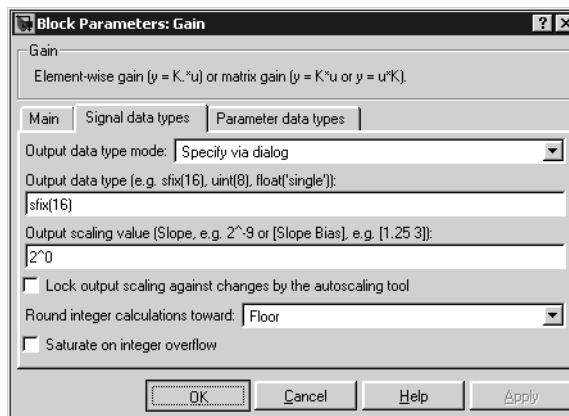
Specify the multiplication mode:

- **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. The input and the output are required to be vectors and their lengths are determined by the dimensions of the gain.

## Sample time (-1 for inherited)

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

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



## Output data type mode

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

If you select *Inherit via internal rule* for this parameter, Simulink chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the output range and maintaining the output precision of the block and with the word size of the targeted hardware implementation specified for the model. If the **Device type** parameter on the **Hardware implementation** configuration parameters pane is set to ASIC/FPGA, Simulink chooses the output data type without regard to hardware constraints. Otherwise, Simulink chooses the smallest available hardware data type capable of meeting the range and precision constraints. For example, if the block multiplies an input of type `int8` by a gain of `int16` and ASIC/FPGA is specified as the targeted hardware type, the output data type is `sfix24`. If *Unspecified* (assume 32-bit *Generic*), i.e., a generic 32-bit microprocessor, is specified as the target hardware, the output data type is `int32`. If none of the word lengths provided by the target microprocessor can accommodate the output range, Simulink displays an error message in the Simulation Diagnostics Viewer.

## Output data type

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

## Output scaling value

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

## Lock output scaling against changes by the autoscaling tool

Select to lock scaling of outputs. This parameter is only visible if you select *Specify via dialog* for the **Output data type mode** parameter.

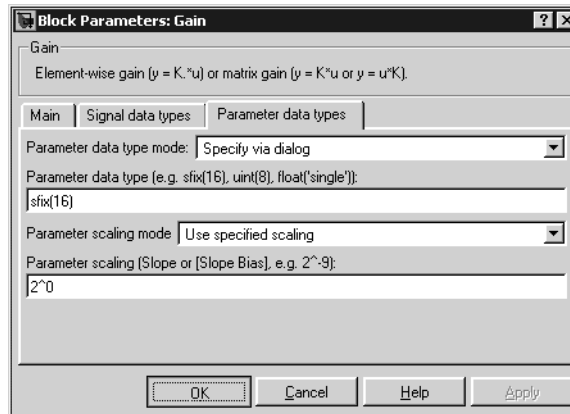
## Round integer calculations toward

Select the rounding mode for fixed-point operations.

## Saturate on integer overflow

Select to have overflows saturate.

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



## Parameter data type mode

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

## Parameter data type

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

## Parameter scaling mode

Set the mode to determine the scaling of the gain.

- **Use specified scaling**—This mode allows you to set the scaling of the gain in the **Parameter scaling** parameter.
- **Best Precision: Element-wise**—This mode sets binary points for the elements of the gain such that the precision of each element is maximized.

- **Best Precision: Row-wise**—This mode sets a common binary point within each row of the gain such that the largest element of each row has the best possible precision.
- **Best Precision: Column-wise**—This mode sets a common binary point within each column of the gain such that the largest element of each column has the best possible precision.
- **Best Precision: Matrix-wise**—This mode sets a common binary point for all the elements of the gain such that the largest element has the best possible precision.

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

**Parameter scaling**

Set the gain scaling using either binary point-only or [Slope Bias] scaling. This parameter is only visible if you select Specify via dialog for the **Parameter data type mode** parameter, and if you select Use specified scaling for the **Parameter scaling mode** parameter.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of input and <b>Gain</b> parameter for Element-wise multiplication
	Dimensionalized	Yes
	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. For limitations on the use of From and Goto blocks, see From on page 1-220. 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 in any subsystem below the Goto Tag Visibility block in the model hierarchy. 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.

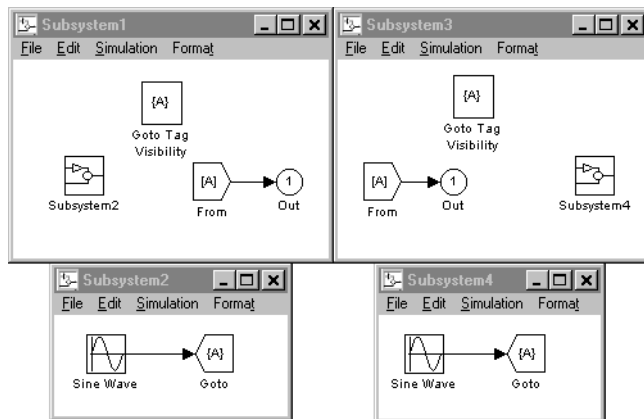
---

**Note** A scoped Goto block in a masked system is visible only in that subsystem and in the subsystems it contains. Simulink 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).

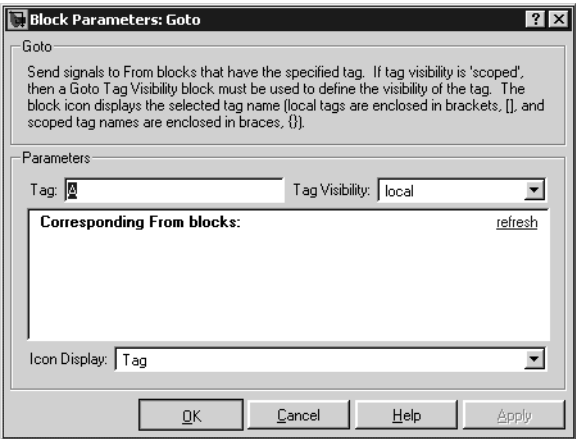


## Data Type Support

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

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

# Goto Tag Visibility

**Purpose** Define scope of Goto block tag

**Library** Signal Routing

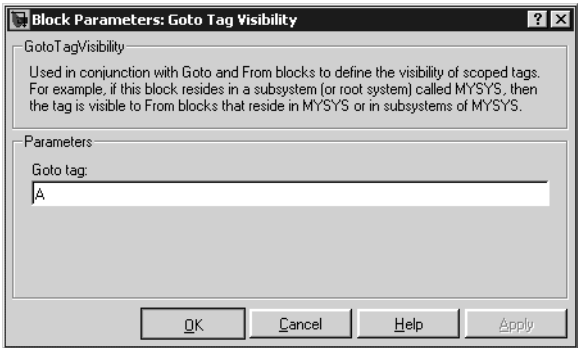
**Description** The Goto Tag Visibility block defines the accessibility of Goto block tags that have scoped visibility. The tag specified as the **Goto tag** parameter is accessible by From blocks in the same subsystem that contains the Goto Tag Visibility block and in subsystems below it in the model 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.

<b>Characteristics</b>	Sample Time	N/A
	Dimensionalized	N/A



**Purpose** Ground an unconnected input port

**Library** Sources

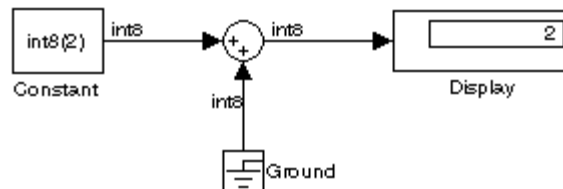
## Description



The Ground block can be used to connect blocks whose input ports are not connected to other blocks. If you run a simulation with blocks having unconnected input ports, Simulink issues warning messages. Using Ground blocks to ground those blocks avoids warning messages. The Ground block outputs a signal with zero value. The data type of the signal is the same as that of the port to which it is connected.

## Data Type Support

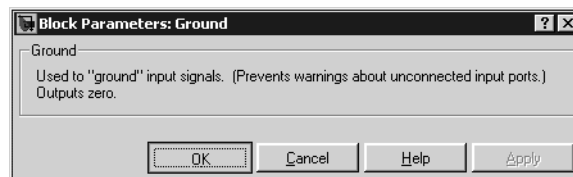
The Ground block outputs a signal of the same numeric type and data type as the port to which it is connected. For example, consider the following model.



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, including fixed-point data types.

## Parameters and Dialog Box



## Characteristics

Sample Time	Inherited from driven block
Dimensionalized	Yes

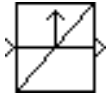
# Hit Crossing

---

**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** parameter.



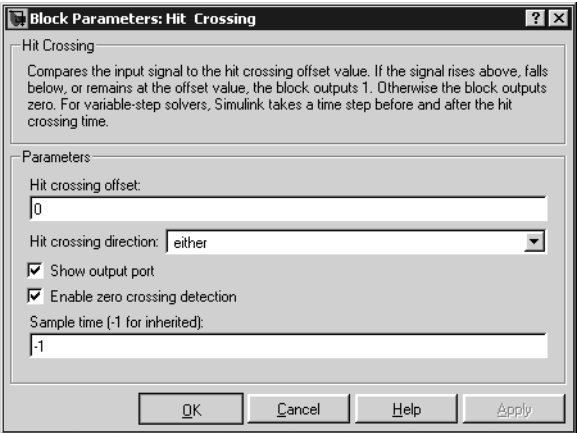
The block accepts one input of type `double`. If you select the **Show output port** check box, the block output indicates when the crossing occurs. If the input signal is exactly the value of the offset value after the hit crossing is detected, the block continues to output a value of 1. If the input 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.

When the block's **Hit crossing direction** parameter 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 `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 `clutch` demo, the Hit Crossing block is in the Lockup Detection subsystem.

**Data Type Support** The Hit Crossing block outputs a signal of type `Boolean` if Boolean logic signals are enabled (see “Enabling Strict Boolean Type Checking”). 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 Using Simulink documentation.

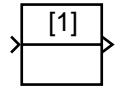
## Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes
Dimensionalized	Yes

**Purpose** Set the initial value of a signal

**Library** Signal Attributes

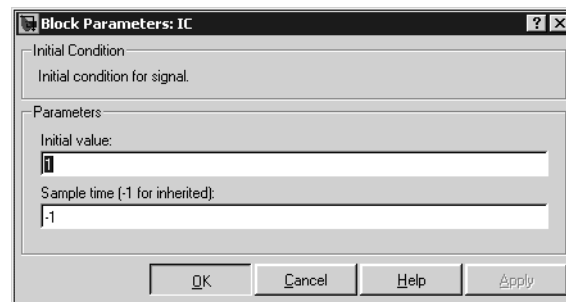
**Description** The IC block sets the value of the signal at its output port at  $t=0$ .



The IC block is also useful for providing an initial guess for the algebraic state variables in the loop. For more information, see “Algebraic Loops” in the Using Simulink documentation.

**Data Type Support** The IC block accepts and outputs a signal of type double.

## Parameters and Dialog Box



### Initial value

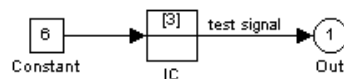
Specify the initial value for the input signal.

### Sample time (-1 for inherited)

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

## Examples

These blocks illustrate how the IC block initializes a signal labeled “test signal.”



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

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of parameter only
	Dimensionalized	Yes
	Zero Crossing	No

# If

## Purpose

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

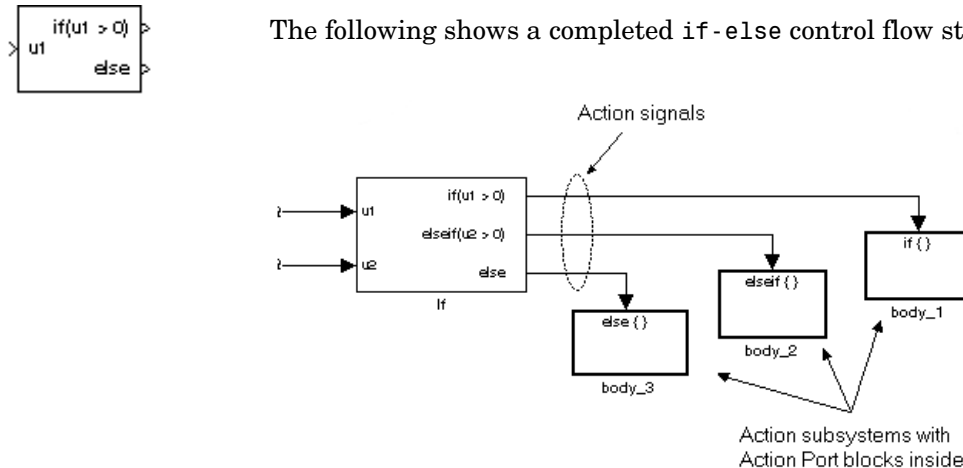
## Library

Ports & Subsystems

## Description

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

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



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

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

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

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

- 1 Place an If block in the current system.
- 2 Open the **Block Parameters** dialog of the If block and enter as follows:
  - Enter the **Number of inputs** field with the required number of inputs necessary to define conditions for the if-else control flow statement. Elements of vector inputs can be accessed for conditions using (row, column) arguments. For example, you can specify the fifth element of the vector u2 in the condition  $u2(5) > 0$  in an **If expression** or **Elseif expressions** field.
  - Enter the expression for the if condition of the if-else control flow statement in the **If expression** field.

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

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

The else port is optional and not required for the operation of the If block.
- 3 Create If Action subsystems to connect to each of the if, else, and elseif ports. These consist of a subsystem with an Action Port block. When you place an Action Port block inside each subsystem, an input port named Action is added to the subsystem.
- 4 Connect each if, else, and elseif port of the If block to the Action port of an If Action subsystem.

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

---

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

---

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

In the preceding example, this is shown as body\_1, body\_2, and body\_3.

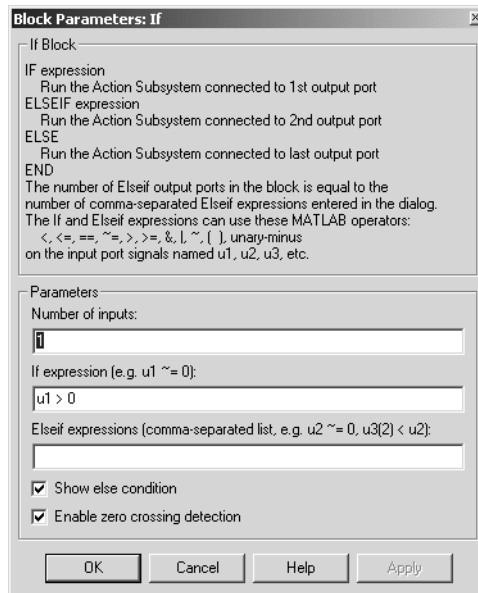
## Data Type Support

Inputs  $u_1, u_2, \dots, u_n$  can be scalar or vector of any built-in Simulink data type. For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.

Outputs from the if, else, and elseif ports are Action signals to If Action subsystems that are created with Action Port blocks and subsystems. See Action Port on page 1-5.



## 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.

### 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.

**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**

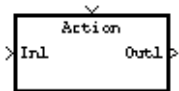
Select to enable zero crossing detection. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes

**Purpose** Represent a subsystem whose execution is triggered by an 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. For more information, see the If block and “Control Flow Blocks” in the Using Simulink documentation.

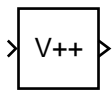


# Increment Real World

**Purpose** Increase the real world value of the 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 data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	No

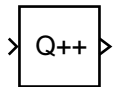
**See Also** Decrement Real World, Increment Stored Integer

**Purpose**

Increase the stored integer value of a signal by one

**Library**

Additional Math & Discrete / Additional Math: Increment - Decrement

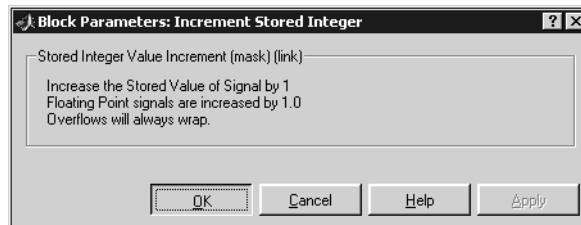
**Description**

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 data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box****Characteristics**

Direct Feedthrough    Yes

Scalar Expansion        No

**See Also**

Decrement Stored Integer, Increment Real World

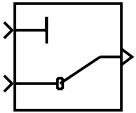
# Index Vector

---

**Purpose** Switch output between different inputs based on the value of the first input

**Library** Signal Routing

**Description** The Index Vector block is an implementation of the Multiport Switch block. See [Multiport Switch](#) on page 1-341 for more information.



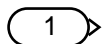
## Purpose

Create an input port for a subsystem or an external input

## Library

Ports & Subsystems, Sources

## Description



Simulink 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, it is assigned 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, you must renumber the block or you will 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, or let Simulink determine it automatically by providing a value of -1.

The **Sample time** parameter is the rate at which the signal is coming into the system. The value of -1 causes the block to inherit its sample time from the block driving it. It might be appropriate to set this parameter for Inport blocks in the top-level system or in models where Inport blocks are driven by blocks whose sample times cannot be determined. See “Specifying Sample Time” in the online documentation for more information.

### 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, choose **Hide Name** from the **Format** menu, then choose **Update Diagram** from the **Edit** menu.

## Inport Blocks in a Top-Level System

Inport blocks in a top-level system have two uses: to supply external inputs from 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 perturb the model.

- To supply external inputs from the workspace, use either the **Configuration Parameters** dialog (see “Loading Input from the Base 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. Inport blocks define the points where inputs are injected into the system. For information about using Inport blocks with analysis commands, see “Analyzing Simulation Results” in the Using Simulink documentation.

## Data Type Support

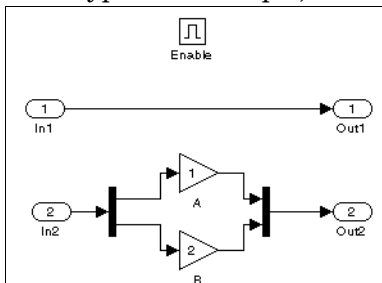
The Inport block accepts complex or real signals of any data type supported by Simulink, including fixed-point data types. For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.

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 inport can be of differing numeric and data types except in the following circumstance: If the subsystem contains an Enable or Trigger block and the



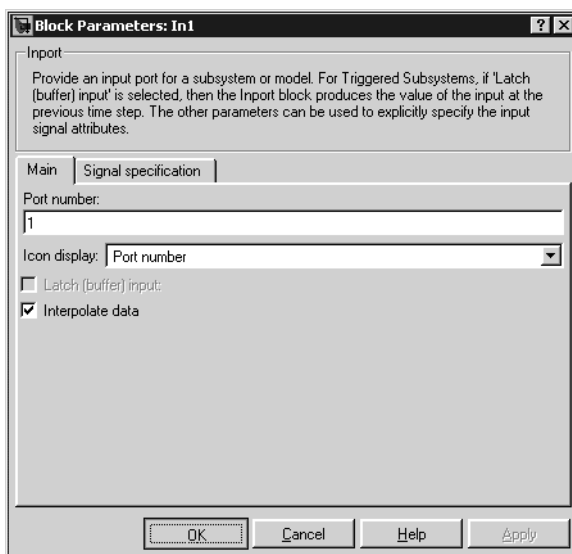
inport is connected directly to an output, the input elements must be of the same type. For example, consider the follow enabled subsystem.



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 appears as follows:



**Port number**

Specify the port number of the Inport block.

**Icon display**

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

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

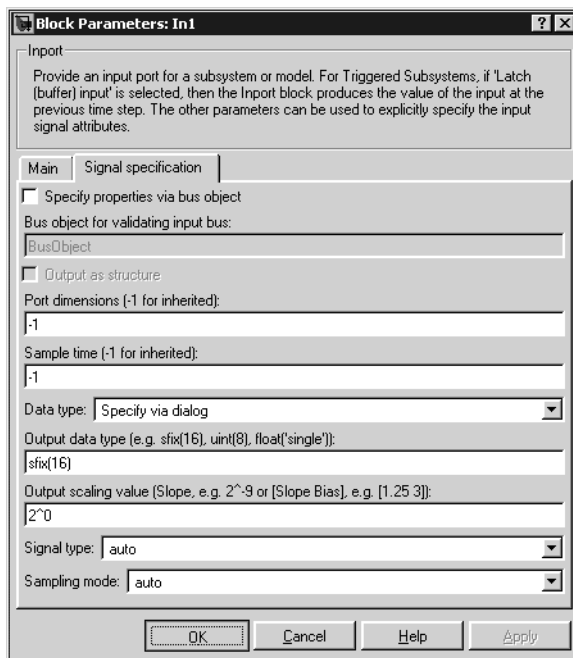
**Latch (buffer) input**

If selected, the block outputs the value of the input signal at the previous time step. This field is enabled only if the Inport block resides in a triggered subsystem.

**Interpolate data**

Select this parameter to cause the block to interpolate or extrapolate output at time steps for which no corresponding workspace data exists when loading data from the workspace. See “Loading Input from the Base Workspace” for more information.

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



## Specify properties via bus object

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

## Bus object for validating input bus

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

## Output as structure

This option is enabled only if you select the **Specify properties via bus object option**. If this option is selected, this block outputs a nonvirtual bus;

otherwise, it outputs a virtual bus (see “Virtual Versus Nonvirtual Buses” in the online Simulink documentation). Select this option if you want code generated from this model to use a C structure to define the structure of the bus signal output by this block.

**Port dimensions**

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

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

**Sample time**

Specify the sample time of the input signal. See “Specifying Sample Time” in the online documentation for more information.

**Data type**

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

**Output data type**

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

**Output scaling value**

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

**Signal type**

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

**Sampling mode**

Specify the sampling mode (Sample based or Frame based) that the input signal must match. To accept any sampling mode, set this parameter to auto. This parameter is intended to support signal processing applications

based on Simulink. See the documentation for the `buffer` function provided by the Signal Processing Toolbox or “Frame-Based Operations” in the documentation for the Signal Processing Blockset for information about frame-based signals.

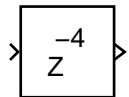
<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Dimensionalized	Yes

# Integer Delay

**Purpose** Delay a signal N sample periods

**Library** Discrete

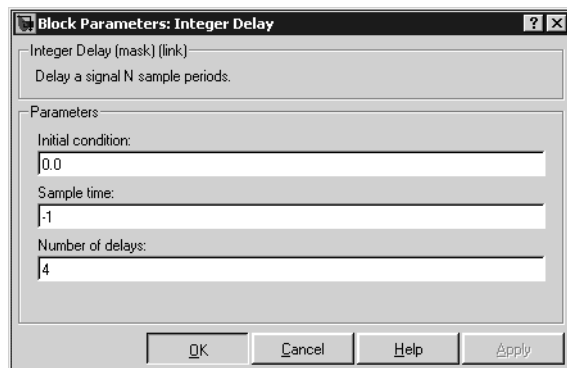
**Description** The Integer Delay block delays its input by N sample periods.



The block accepts one input and generates one output, both of which can be scalar or vector. If the input is a vector, all elements of the vector are delayed by the same sample period.

**Data Type Support** The Integer Delay block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box



### Initial condition

The initial output of the simulation. The **Initial condition** parameter is converted from a double to the input data type offline using round-to-nearest and saturation.

### Sample time (-1 for inherited)

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

### Number of delays

The number of periods to delay the input signal.

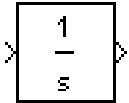
<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of input or initial conditions

# Integrator

**Purpose** Integrate a signal

**Library** Continuous

**Description** The Integrator block outputs the integral of its input at the current time step. The following equation represents the output of the block  $y$  as a function of its input  $u$  and an initial condition  $y_0$ , where  $y$  and  $u$  are vector functions of the current simulation time  $t$ .



$$y(t) = \int_{t_0}^t u(t)dt + y_0$$

Simulink can use a number of different numerical integration methods to compute the Integrator block's output, each with advantages in particular applications. The **Solver** pane of the **Configuration parameters** dialog box (see "The Solver Pane") allows you to select the technique best suited to your application.

Simulink treats the Integrator block as a dynamic system with one state, its output. The Integrator block's input is the state's time derivative.

$$x = y(t)$$

$$x_0 = y_0$$

$$\dot{x} = u(t)$$

The currently selected solver computes the output of the Integrator block at the current time step, using the current input value and the value of the state at the previous time step. To support this computational model, the Integrator block saves its output at the current time step for use by the solver to compute its output at the next time step. The block also provides the solver with an initial condition for use in computing the block's initial state at the beginning of a simulation run. The default value of the initial condition is 0. The block's parameter dialog box allows you to specify another value for the initial condition or create an initial value input port on the block.



The parameter dialog box also allows you to

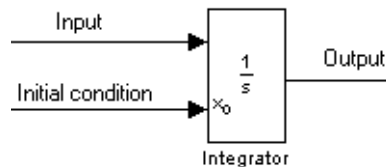
- Define upper and lower limits on the integral
- Create an input that resets the block's output (state) to its initial value, depending on how the input changes
- Create an optional state output that allows you to use the value of the block's output to trigger a block reset

Use the Discrete-Time Integrator block to create a purely discrete system.

## Defining Initial Conditions

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

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



---

**Note** If the integrator limits its output (see “**Limiting the Integral**”), the initial condition must fall inside the integrator's saturation limits. If the initial condition is outside the block's saturation limits, the block displays an error message.

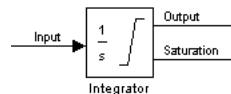
---

## Limiting the Integral

To prevent the output from exceeding specifiable levels, select the **Limit output** check box and enter the limits in the appropriate parameter fields. Doing so causes the block to function as a limited integrator. When the output 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 on this figure.



The signal has one of three values:

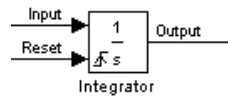
- 1 indicates that the upper limit is being applied.
- 0 indicates that the integral is not limited.
- -1 indicates that the lower limit is being applied.

When you select this option, the block has three zero crossings: one to detect when it enters the upper saturation limit, one to detect when it enters the lower saturation limit, and one to detect when it leaves saturation.

## Resetting the State

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

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

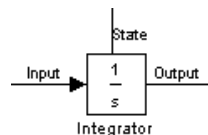


- Select rising to trigger the state reset when the reset signal has a rising edge.
- Select falling to trigger the state reset when the reset signal has a falling edge.
- Select either to trigger the reset when either a rising or falling signal occurs.
- Select level1 to trigger the reset and hold the output to the initial condition while the reset signal is nonzero.

The reset port has direct feedthrough. If the block output is fed back into this port, either directly or through a series of blocks with direct feedthrough, an algebraic loop results (see “Algebraic Loops”). The Integrator block's state port allows you to feed back the block's output without creating an algebraic loop.

## About the State Port

Selecting the **Show state port** option on the Integrator block's parameter dialog box causes an additional output port, the state port, to appear atop the Integrator block.



The output of the state port is the same as the output of the block's standard output port except for the following case. If the block is reset in the current time step, the output of the state port is the value that would have appeared at the block's standard output if the block had not been reset. The state port's output appears earlier in the time step than the output of the Integrator block's output port. This allows you to avoid creating algebraic loops in the following modeling scenarios:

- Self-resetting integrators (see “Creating Self-Resetting Integrators” on page 1-268)
- Handing off a state from one enabled subsystem to another (see “Handing Off States Between Enabled Subsystems” on page 1-269)

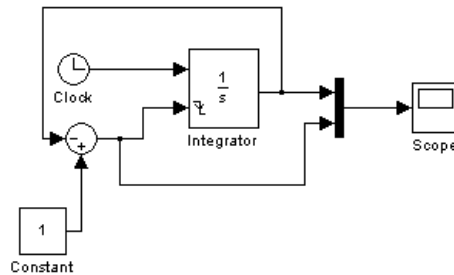
---

**Note** The state port is intended to be used specifically in these two scenarios. When updating a model, Simulink checks to ensure that the state port is being used in one of these two scenarios. If not, Simulink signals an error.

---

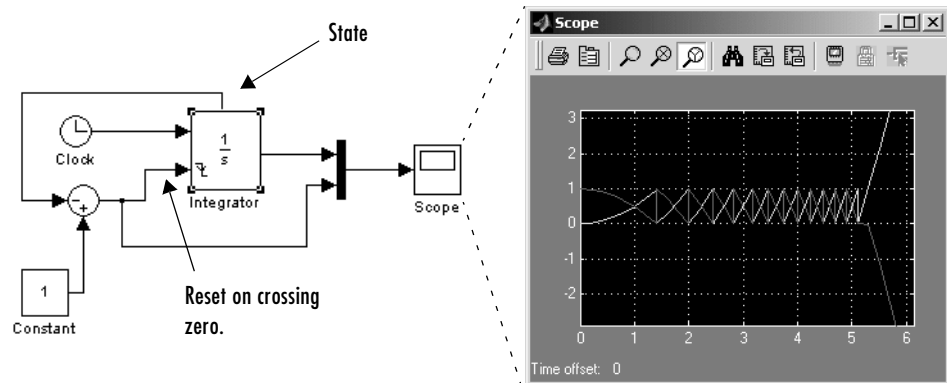
## Creating Self-Resetting Integrators

The Integrator block’s state port allows you to avoid creating algebraic loops when creating an integrator that resets itself based on the value of its output. Consider, for example, the following model.



This model tries to create a self-resetting integrator by feeding the integrator’s output, subtracted from 1, back into the integrator’s reset port. In so doing, however, the model creates an algebraic loop. To compute the integrator block’s output, Simulink needs to know the value of the block’s reset signal, and vice versa. Because the two values are mutually dependent, Simulink cannot determine either. It therefore signals an error if you try to simulate or update this model.

The following model uses the integrator's state port to avoid the algebraic loop.

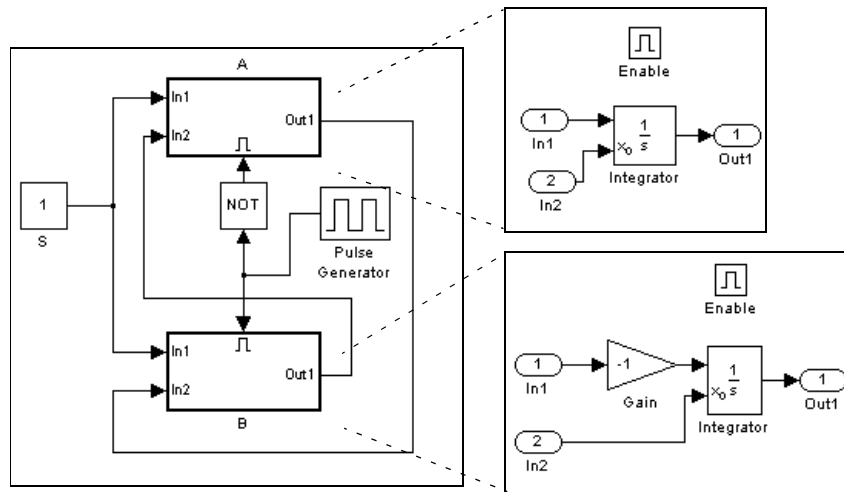


In this version, the value of the reset signal depends on the value of the state port. The value of the state port is available earlier in the current time step than the value of the integrator block's output port. Thus, Simulink can determine whether the block needs to be reset before computing the block's output, thereby avoiding the algebraic loop.

## Handing Off States Between Enabled Subsystems

The state port allows you to avoid an algebraic loop when passing a state between two enabled subsystems. Consider, for example, the following model.

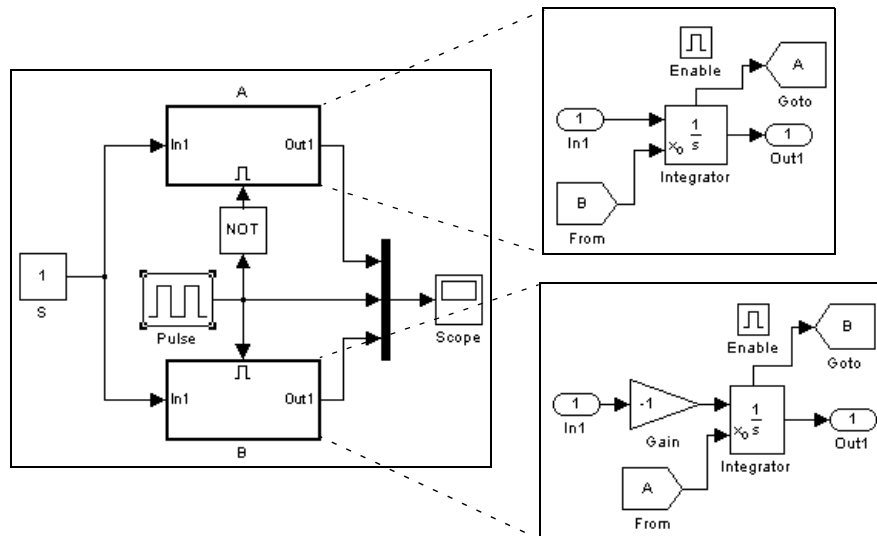
# Integrator



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. This 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 the two subsystems. However, the connection creates an algebraic loop. To compute the output of A, Simulink needs to know the output of B, and vice versa. Because the outputs are mutually dependent, Simulink cannot compute them. It therefore generates an error if you attempt to update or simulate 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 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 clutch model.

---

**Note** Simulink does not permit three or more enabled subsystems to hand off a model state. If Simulink 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

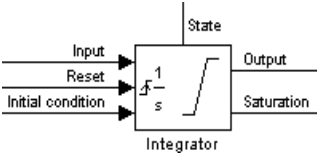
By default Simulink uses the absolute tolerance value specified in the **Configuration Parameters** dialog box (see “Error Tolerances”) to compute the output of the Integrator block. If this value does not provide sufficient error

# Integrator

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.

## Choosing All Options

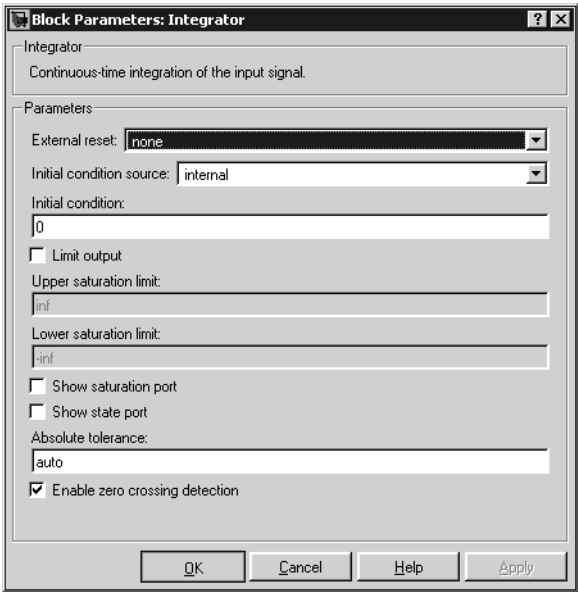
When all options are selected, the icon looks like this.



## Data Type Support

The Integrator block accepts and outputs signals of type double on its data ports. Its external reset port accepts signals of type double or Boolean.

## Parameters and Dialog Box



## External reset

Resets the states to their initial conditions when a trigger event (rising, falling, either, or level) occurs in the reset signal.



## **Initial condition source**

Gets the states' initial conditions from the **Initial condition** parameter (if set to internal) or from an external block (if set to external).

## **Initial condition**

The states' initial conditions. Set the **Initial condition source** parameter value to internal.

## **Limit output**

If selected, limits the states to a value between the **Lower saturation limit** and **Upper saturation limit** parameters.

## **Upper saturation limit**

The upper limit for the integral. The default is  $\text{inf}$ .

## **Lower saturation limit**

The lower limit for the integral. The default is  $-\text{inf}$ .

## **Show saturation port**

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

## **Show state port**

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

## **Absolute tolerance**

Absolute tolerance used to compute the block's outputs. You can enter auto or a numeric value. If you enter auto, Simulink determines the absolute tolerance (see "Error Tolerances"). If you enter a numeric value, Simulink uses the specified value to compute the block's outputs. Note that a numeric value overrides the setting for the absolute tolerance in the **Configuration Parameters** dialog box.

## **Enable zero crossing detection**

If this option, **Limit output**, and zero-crossing detection for the model as a whole are selected, the Integrator block uses zero-crossings to detect and take a time step at any of the following events: reset, entering or leaving an upper saturation state, entering or leaving a lower saturation state. For more information, see "Zero Crossing Detection" in the Using Simulink documentation.

# Integrator

---

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	If you select the <b>Limit output</b> option, one for detecting reset, one each to detect upper and lower saturation limits, one when leaving saturation

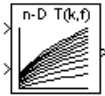
## Purpose

Perform high-performance constant or linear interpolation, mapping N input values to a sampled representation of a function in N variables via output from PreLookup Index Search block

## Library

Lookup Tables

## Description



The Interpolation (n-D) Using PreLookup block uses the precalculated indices and interval fractions from the PreLookup Index Search block to perform the equivalent operation that the Lookup Table (n-D) performs. This combination of blocks allows multiple Interpolation (n-D) blocks to feed a set of PreLookup Index Search blocks. In models that have many interpolation blocks, simulation performance can be greatly increased.

This block supports two interpolation methods: flat (constant) interval lookup and linear interpolation. These operations can be applied to 1-D, 2-D, 3-D, 4-D and higher dimensioned tables.

You define a set of output values as the **Table data** parameter. These table values must correspond to the breakpoint data sets that are in the PreLookup Index Search block. The block generates its output by interpolating the table values based on the (index, fraction) pairs fed into the block by each PreLookup Index Search block.

The block generates output based on the input values:

- If the inputs match breakpoint parameter values, the output is the table value at the intersection of the row, column, and higher dimensions' breakpoints.
- If the inputs do not match row and column parameter values, the block generates output by interpolating between the appropriate table values. If either or both block inputs are less than the first or greater than the last row or column parameter values, the block extrapolates from the first two or last two points in each corresponding dimension.

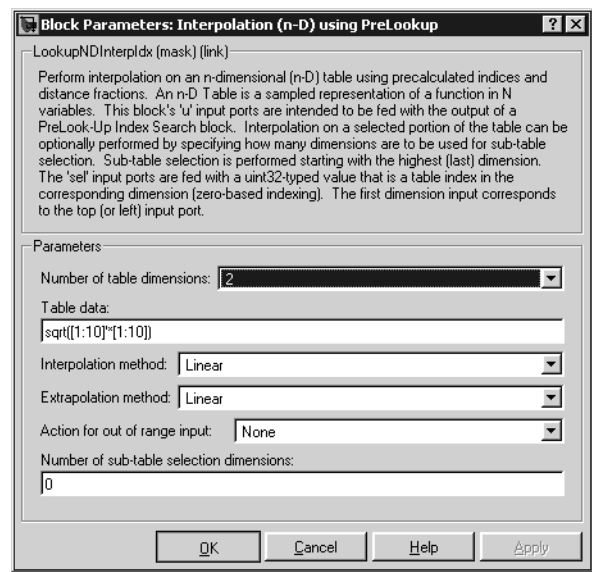
The block can perform interpolation on portion of the table. For more information, see “Lookups: Prelookup and Sub-Table Interpolation Blocks” in the “Block Diagramming Features” section of the “Simulink” section on the **Demos** pane of the MATLAB Help browser.

# Interpolation (n-D) Using PreLookup

## Data Type Support

The Interpolation (n-D) Using PreLookup block accepts signals of types `double` or `single`, but for any given block, the inputs must all be of the same type. The **Table data** parameter must be of the same type as the inputs. The output data type is set to the **Table data** data type.

## Parameters and Dialog Box



### Number of table dimensions

The number of dimensions that the **Table data** parameter must have. This determines the number of independent variables for the table and hence the number of inputs to the block (see descriptions for “Explicit Number of dimensions” and “Use one (vector) input port instead of N ports,” below).

### Table data

The table of output values. The matrix size must match the dimensions defined by the **N breakpoint set** parameter or by the **Explicit number of dimensions** parameter when the number of dimensions exceeds four. During block diagram editing, you can leave the **Table data** field empty, but for running the simulation, you must match the number of dimensions in the **Table data** parameter to the **Number of table dimensions**. For

information about how to construct multidimensional arrays in MATLAB, see “Multidimensional Arrays” in the MATLAB online documentation.

**Interpolation method**

None (flat) or Linear.

**Extrapolation method**

None (clip) or Linear.

**Action for out of range input**

None, Warning, Error.

**Number of sub-table selection dimensions**

Number of dimensions of the subtable used to compute this block’s output. Specify 0 to use the entire table specified by **Table data** parameter.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving blocks
	Scalar Expansion	Yes
	Zero Crossing	No

# Interval Test

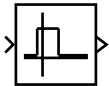
## Purpose

Determine if a signal is in a specified interval

## Library

Logic and Bit Operations

## Description

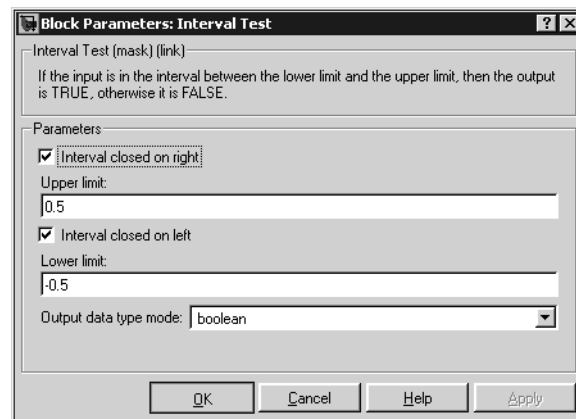


The Interval Test block outputs TRUE if the input is between the values specified by the **Lower limit** and **Upper limit** parameters. The block outputs FALSE if the input is outside those values. The output of the block when the input is equal to the **Lower limit** or the **Upper limit** is determined 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 any data type supported by Simulink, including fixed-point data types.

## 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.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

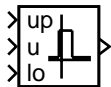
<b>See Also</b>	Interval Test Dynamic
-----------------	-----------------------

# Interval Test Dynamic

**Purpose** Determine if a signal is in a specified interval

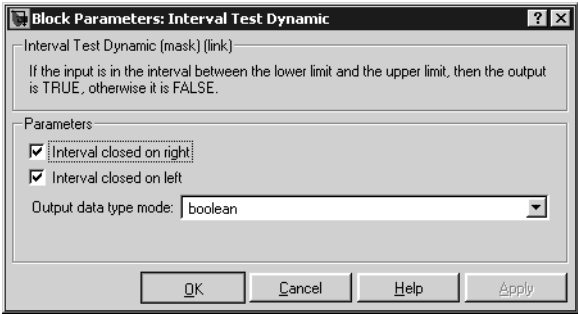
**Library** Logic and Bit Operations

**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 any data type supported by Simulink, including fixed-point data types.

**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.

**Interval closed on left**  
When you select this check box, the **Lower limit** is included in the interval for which the block outputs TRUE.

**Output data type mode**  
Select the output data type; boolean or uint8.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes



## See Also

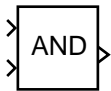
[Interval Test](#)

# Logical Operator

**Purpose** Perform the specified logical operation on the 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. The block updates to display 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
NOT	TRUE if the input is FALSE

The number of input ports is specified with the **Number of input ports** parameter. The output type is specified with the **Output data type mode** and/or the **Output data type** parameters. 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

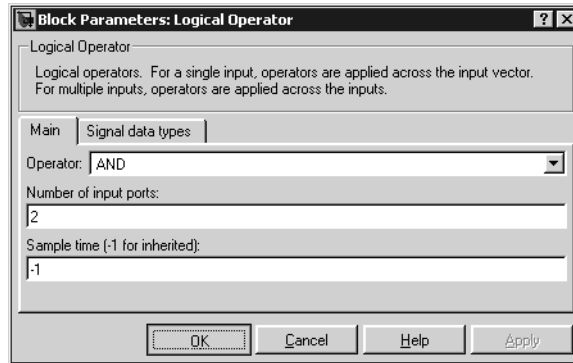
The Logical Operator block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. However, if the **Output data type mode** parameter is set to Logical, the input may only be Boolean or double.

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

## Parameters and Dialog Box

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

# Logical Operator



## Operator

The logical operator to be applied to the block inputs. Valid choices are the operators listed previously.

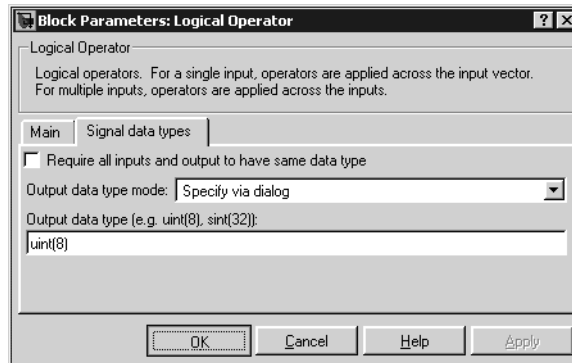
## Number of input ports

The number of block inputs. The value must be appropriate for the selected operator.

## Sample time (-1 for inherited)

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

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



## Require all inputs and output to have same data type

Select to require all inputs and the output to have the same data type.

## Output data type mode

Set the output data type to Boolean, or choose to specify the data type through the **Output data type** parameter.

Alternatively, you can select Logical to have the output data type determined by the **Implement logic signals as boolean data** parameter in the **Simulation and code generation** optimization pane of the **Configuration Parameters** dialog box. If you select Logical and **Implement logic signals as boolean data** is on, then the output data type is always Boolean. If you select Logical and **Implement logic signals as boolean data** is off, then the output data type will match the input data type, which may be Boolean or double.

## Output data type

Output data type. You should only use data types that represent zero exactly. Data types that satisfy this condition include signed and unsigned integers and any floating-point data type. This parameter is only visible if you select Specify via dialog for the **Output data type mode** parameter.

# Logical Operator

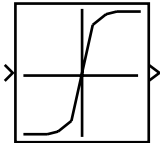
---

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of inputs
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Approximate a one-dimensional function using the specified lookup method

**Library** Lookup Tables

**Description** The Lookup Table block computes an approximation to some function  $y=f(x)$  given data vectors  $x$  and  $y$ .



---

**Note** To map two inputs to an output, use the Lookup Table (2-D) block.

---

The length of the  $x$  and  $y$  data vectors provided to this block must match. Also, the  $x$  data vector must be strictly monotonically increasing after conversion to the input's fixed-point data type, except in the following case. If the input  $x$  and the output signal are both either single or double, and if the lookup method is Interpolation-Extrapolation, then  $x$  may be monotonically increasing rather than strictly monotonically increasing. Note that due to quantization, the  $x$  data vector may be strictly monotonic in doubles format, but not so after conversion to a fixed-point data type.

You define the table by specifying the **Vector of input values** parameter as a 1-by- $n$  vector and the **Vector of output values** parameter as a 1-by- $n$  vector. The block generates output based on the input values using one of these methods selected from the **Look-up method** parameter list:

- Interpolation-Extrapolation—This is the default method; it performs linear interpolation and extrapolation of the inputs.
  - If a value matches the block's input, the output is the corresponding element in the output vector.
  - If no value matches the block's input, then the block performs linear interpolation between the two appropriate elements of the table to determine an output value. If the block input is less than the first or greater than the last input vector element, then the block extrapolates using the first two or last two points.
- Interpolation-Use End Values—This method performs linear interpolation as described above but does not extrapolate outside the end points of the input vector. Instead, the end-point values are used.

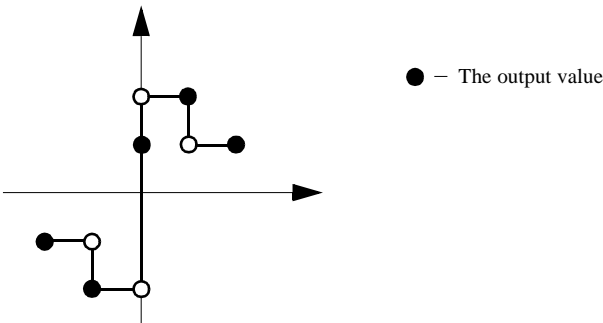
# Lookup Table

- Use Input Nearest—This method does not interpolate or extrapolate. Instead, the element in  $x$  nearest the current input is found. The corresponding element in  $y$  is then used as the output.
- Use Input Below—This method does not interpolate or extrapolate. Instead, the element in  $x$  nearest and below the current input is found. The corresponding element in  $y$  is then used as the output. If there is no element in  $x$  below the current input, then the nearest element is found.
- Use Input Above—This method does not interpolate or extrapolate. Instead, the element in  $x$  nearest and above the current input is found. The corresponding element in  $y$  is then used as the output. If there is no element in  $x$  above the current input, then the nearest element is found.

**Note** Note that there is no difference among the Use Input Nearest, Use Input Below, and Use Input Above methods when the input  $x$  corresponds exactly to table breakpoints.

To create a table with step transitions, repeat an input value with different output values. For example, these input and output parameter values create the input/output relationship described by the plot that follows:

Vector of input values:    [-2 -1 -1    0 0 0 1 1 2]  
Vector of output values:   [-1 -1 -2 -2 1 2 2 1 1]



This example has three step discontinuities: at  $u = -1$ ,  $0$ , and  $+1$ .

When there are two points at a given input value, the block generates output according to these rules:



- When the input signal  $u$  is less than zero, the output is the value connected with the point first encountered when moving away from the origin in a negative direction. In this example, when  $u$  is -1,  $y$  is -2, marked with a solid circle.
- When  $u$  is greater than zero, the output is the value connected with the point first encountered when moving away from the origin in a positive direction. In this example, when  $u$  is 1,  $y$  is 2, marked with a solid circle.
- When  $u$  is at the origin and there are two output values specified for zero input, the actual output is their average. In this example, if there were no point at  $u = 0$  and  $y = 1$ , the output would be 0, the average of the two points at  $u = 0$ . If there are three points at zero, the block generates the output associated with the middle point. In this example, the output at the origin is 1.

The Lookup Table icon displays a graph of the input vector versus the output vector. When a parameter is changed on the block's dialog box, the graph is automatically redrawn when you click the **Apply** or **Close** button.

To avoid parameter saturation errors, the automatic scaling script `autofixexp` 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 **Vector of output values** parameter (the `YDataPoints` variable).

## Data Type Support

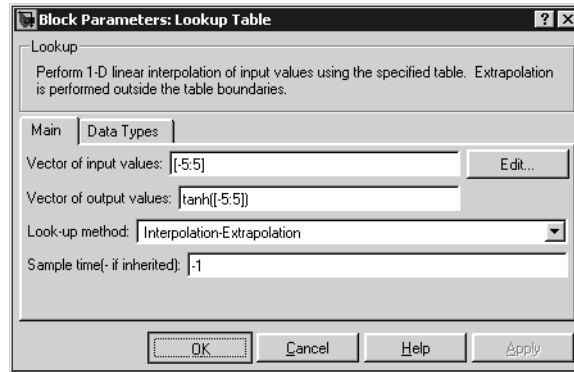
The Lookup Table block supports all data types supported by Simulink, including fixed-point data types.

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

## Parameters and Dialog Box

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

# Lookup Table



## Vector of input values

Specify the vector of input values. The input values vector must be the same size as the output values vector. Also, the input values vector must be strictly monotonically increasing after conversion to the input's fixed-point data type, except in the following case. If the input values vector and the output signal are both either single or double, and if the lookup method is Interpolation-Extrapolation, then the input values vector may be monotonically increasing rather than strictly monotonically increasing. 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 from doubles to the input data type offline using round-to-nearest and saturation.

Click the **Edit** button to invoke the Lookup Table Editor.

## Vector of output values

Specify the vector of output values. The output values vector must be the same size as the input values vector.

The **Vector of output values** parameter is converted from doubles to the output data type offline using round-to-nearest and saturation.

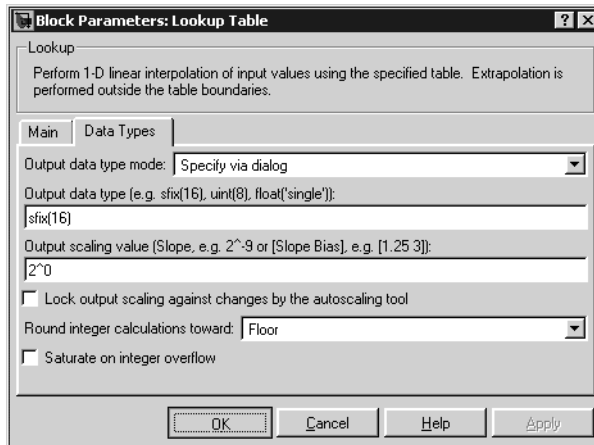
## Look-up method

Specify the lookup method. See “Description” on page 1-287 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 “Specifying Sample Time” in the online documentation for more information.

The **Data types** pane of the Lookup Table block dialog appears as follows:



## Output data type mode

You can set the output signal to a built-in data type from this drop-down list, or you can choose the output data type and scaling to be the same as the input. Alternatively, you can choose to inherit the output data type and scaling by backpropagation. Lastly, if you choose **Specify via dialog**, the **Output data type**, **Output scaling value**, and **Lock output scaling against changes by the autoscaling tool** parameters become visible.

## Output data type

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

## Output scaling value

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

# Lookup Table

## Lock output scaling against changes by the autoscaling tool

Select to lock scaling of outputs. This parameter is only visible if you select Specify via dialog for the **Output data type mode** parameter.

## Round integer calculations toward

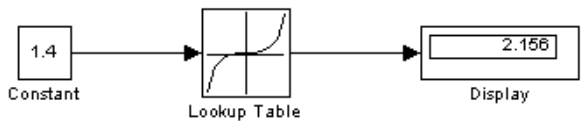
Select the rounding mode for fixed-point operations.

Note that block parameters such as **Vector of output values** are always rounded to the nearest representable value. To directly 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.

## Examples



Suppose the Lookup Table block in the above model is configured to use a vector of input values given by `[-5:5]`, and a vector of output values given by `sinh([-5:5])`. The following results are generated.

Lookup Method	Input	Output	Comment
Interpolation-Extrapolation	1.4	2.156	N/A
	5.2	83.59	N/A
Interpolation-Use End Values	1.4	2.156	N/A
	5.2	74.2	The value for <code>sinh(5.0)</code> was used.
Use Input Above	1.4	3.627	The value for <code>sinh(2.0)</code> was used.
	5.2	74.2	The value for <code>sinh(5.0)</code> was used.

Lookup Method	Input	Output	Comment
Use Input Below	1.4	1.175	The value for sinh(1.0) was used.
	-5.2	-74.2	The value for sinh(-5.0) was used.
Use Input Nearest	1.4	1.175	The value for sinh(1.0) was used.

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

See Also	Lookup Table (2-D), Lookup Table (n-D)
----------	--

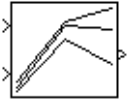
# Lookup Table (2-D)

---

**Purpose** Approximate a two-dimensional function using a selected lookup method

**Library** Lookup Tables

## Description



The Lookup Table (2-D) block computes an approximation to some function  $z=f(x,y)$  given  $x$ ,  $y$ ,  $z$  data points.

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 **Matrix of output values** parameter is an  $m$ -by- $n$  matrix of  $z$  data points. Both the row and column vectors must be monotonically increasing. These vectors must be strictly monotonically increasing 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.

The block generates output based on the input values using one of these methods selected from the **Look-up method** parameter list:

- Interpolation-Extrapolation—This is the default method; it performs linear interpolation and extrapolation of the inputs.
  - If 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.
- 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** Note that there is no difference among the **Use Input Nearest**, **Use Input Below**, and **Use Input Above** methods when the input *x* corresponds exactly to table breakpoints.

---

To avoid parameter saturation errors, the automatic scaling script `autofixexp` 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 prevents the data from being saturated to different values.

### Data Type Support

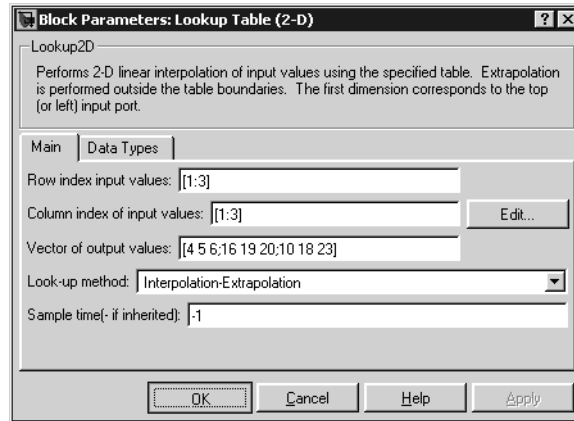
The Lookup Table (2-D) block supports all data types supported by Simulink, including fixed-point data types.

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

### Parameters and Dialog Box

The **Main** pane of the Lookup Table (2-D) block dialog appears as follows:

# Lookup Table (2-D)



## Row index input values

The row values for the table, entered as a vector. The vector values must increase monotonically.

## Column index input values

The column values for the table, entered as a vector. The vector values must increase monotonically.

Click the **Edit** button to invoke the Lookup Table Editor.

## Vector of output values

The table of output values. The matrix size must match the dimensions defined by the **Row** and **Column** parameters.

## Look-up method

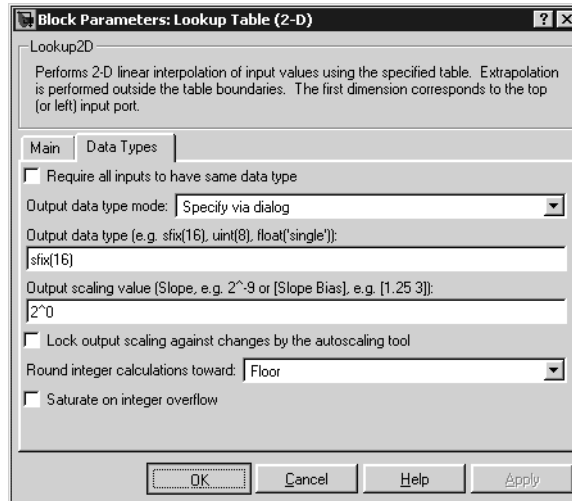
Specify the lookup method. See “Description” on page 1-294 for a discussion of the options for this parameter.

## Sample time (-1 for inherited)

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

The **Data types** pane of the Lookup Table (2-D) block dialog appears as follows:





## Require all inputs to have same data type

Select to require all inputs to have the same data type.

## Output data type mode

You can set the output signal to a built-in data type from this drop-down list, or you can choose the output data type and scaling to be the same as the input. Alternatively, you can choose to inherit the output data type and scaling by backpropagation. Lastly, if you choose **Specify via dialog**, the **Output data type**, **Output scaling value**, and **Lock output scaling against changes by the autoscaling tool** parameters become visible.

## Output data type

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

## Output scaling value

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

# Lookup Table (2-D)

## Lock output scaling against changes by the autoscaling tool

Select to lock scaling of outputs. This parameter is only visible if you select Specify via dialog for the **Output data type mode** parameter.

## Round integer calculations toward

Select the rounding mode for fixed-point operations.

Note that block parameters such as **Vector of output values** are always rounded to the nearest representable value. To directly control the rounding of a block parameter, enter an expression using a MATLAB rounding function into the mask field.

## Saturate on integer overflow

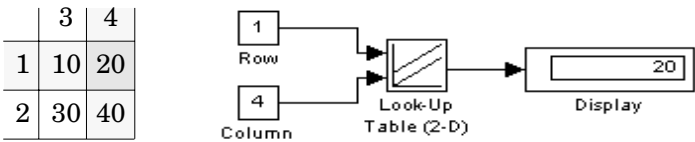
Select to have overflows saturate.

## Examples

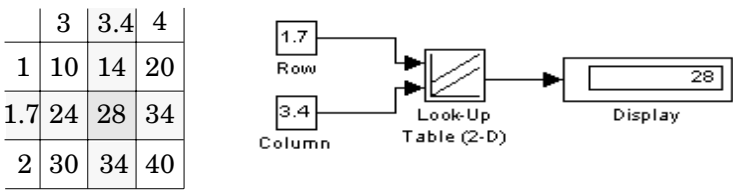
In this example, the block parameters are defined as

Row: [1 2]  
Column: [3 4]  
Table: [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).



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.



<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of one input if the other is a vector
	Dimensionalized	Yes
	Zero Crossing	No

**See Also**      Lookup Table, Lookup Table (n-D)

# Lookup Table (n-D)

**Purpose** Perform constant, linear, or spline interpolated mapping of N input values to a sampled representation of a function in N variables

**Library** Lookup Tables

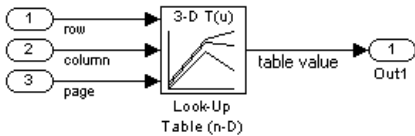
**Description** The Lookup Table (n-D) block evaluates a sampled representation of a function in N variables by interpolating between samples to give an approximate value for  $y = F(x_1, x_2, x_3, \dots, x_n)$ , even when the function  $F$  is known only empirically. The block efficiently maps the block inputs to the output value using interpolation on a table of values defined by the block's parameters. Interpolation methods supported are



- Flat (constant)
- Linear
- Natural (cubic) spline

You can apply any of these methods to 1-D, 2-D, 3-D, or higher dimensional tables.

You define a set of output values as the **Table data** parameter and the values that correspond to its rows, columns, and higher dimensions with the  $N$ th breakpoint set parameter. The block generates an output value by comparing the block inputs with the breakpoint set parameters. The first input identifies the first dimension (row) breakpoints, the second breakpoint set identifies a column, and so on, as shown by this figure.



If you are unfamiliar with how to construct N-dimensional arrays in MATLAB, see “Multidimensional Arrays” in the MATLAB online documentation.

The block generates output based on the input values:

- If the inputs match breakpoint parameter values, the output is the table value at the intersection of the row, column, and higher dimensions breakpoints.
- If the inputs do not match row and column parameter values, the block generates output by interpolating between the appropriate table values. If any of the block inputs are outside the ranges of their respective breakpoint sets, the block limits the input values to the breakpoint set's range in that dimension. If extrapolation is enabled, it extrapolates linearly or by using a cubic polynomial (if you selected cubic spline extrapolation).

---

**Note** As an alternative, you can use the Lookup Table (n-D) block with the PreLookup Index Search block to have more flexibility and potentially much higher performance for linear interpolations in certain circumstances.

---

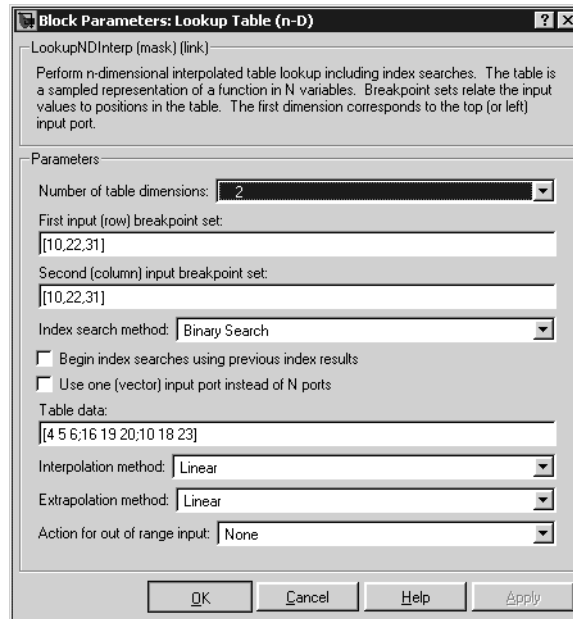
For noninterpolated table lookups, use the Direct Lookup Table (n-D) block when the lookup operation is a simple array access, for example, if you have an integer value  $k$  and you merely want the  $k$ th element of a table,  $y = \text{table}(k)$ .

## Data Type Support

The Lookup Table (n-D) block accepts signals of types `double` or `single`, but for any given Lookup Table (n-D) block, the inputs must all be of the same type. Table data and Breakpoint set parameters must be of the same type as the inputs. The output data type is also set to the input data type.

# Lookup Table (n-D)

## Parameters and Dialog Box



### Number of table dimensions

The number of dimensions that the **Table data** parameter is to have. This determines the number of independent variables for the table and hence the number of inputs to the block (see descriptions for “Explicit Number of dimensions” and “Use one (vector) input port instead of N ports”, following).

### First input (row) breakpoint set

The row values represented in the table, entered as a vector. The vector values must increase monotonically. This field is always visible.

### Second (column) input breakpoint set

The column values for the table, entered as a vector. The vector values must increase monotonically. This field is visible if the **Number of table dimensions** value is 2, 3, 4, or More.

## Third ... Nth input breakpoint set

The values corresponding to the third dimension for the table, entered as a vector. The vector values must increase monotonically. This field is visible if the **Number of table dimensions** is 3, 4, or More.

## Fourth input breakpoint set

The values corresponding to the fourth dimension for the table, entered as a vector. The vector values must increase monotonically. This field is visible if the **Number of table dimensions** is 4 or More.

## Fifth..Nth input breakpoint sets (cell array)

The cell array of values corresponding to the third, fourth, or higher dimensions for the table, entered as a 1-D cell array of vectors. For example, {[10:10:30], [0:10:100]} is a cell array of two vectors that are used for the fifth and sixth dimensions' breakpoint sets. The vector values must increase monotonically. This field is visible if the **Number of table dimensions** is More.

## Explicit number of dimensions

The number of table dimensions when the number is 5 or more. This is indicated when you set the **Number of table dimensions** field to More.

## Index search method

Choose Evenly Spaced Points, Linear Search, or Binary Search (the default). Each search method has speed advantages over the others in different circumstances. A suboptimal choice of index search method can lead to slow performance in models that rely heavily on lookup tables. If the breakpoint data is evenly spaced, e.g., 10, 20, 30, ..., you can achieve the greatest speed by selecting Evenly Spaced Points to directly calculate the indices into the table. For irregularly spaced breakpoint sets, if the input signals do not vary much from one time step to the next, selecting Linear Search and Begin index searches using previous index results at the same time will produce the best performance. For irregularly spaced breakpoint sets with rapidly varying input signals that jump more than one or two table intervals per time step, selecting Binary Search gives the best performance. Note that the Evenly Spaced Points algorithm only makes use of the first two breakpoints in determining the offset and spacing of the rest of the points.

# Lookup Table (n-D)

---

## **Begin index searches using previous index results**

Activating this option causes the block to initialize index searches using the index found on the previous time step. This is a huge performance improvement for the block when the input signals do not change much with respect to its position in the table from one time step to the next. When this option is deactivated, the linear search and binary search methods can take significantly longer, especially for large breakpoint data sets.

## **Use one (vector) input port instead of N ports**

Instead of having one input port per independent variable, the block is configured with just one input port that expects a signal that is N elements wide for an N-dimensional table. This might be useful in removing line clutter on a block diagram with large numbers of tables.

## **Table data**

The table of output values. To execute a model with this block, the matrix size must match the dimensions defined by the **N breakpoint set** parameter or by the **Explicit number of dimensions** parameter when the number of dimensions exceeds 4. During block diagram editing, you can leave this field blank because only the **Number of table dimensions** field is required to set the number of ports on the block.

## **Interpolation method**

None (flat), Linear, or Cubic Spline.

## **Extrapolation method**

None (clip), Linear, or Cubic Spline.

## **Action for out of range input**

None, Warning, or Error. An out-of-range condition during simulation results in warning messages in the command window if you select “Warning,” and the simulation halts with an error message if you select “Error.”



## Characteristics

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

## See Also

Lookup Table, Lookup Table (2-D), Lookup Table Dynamic

# Lookup Table Dynamic

---

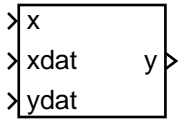
## Purpose

Approximate a one-dimensional function using a selected lookup method and a dynamically specified table

## Library

Lookup Tables

## Description



The Lookup Table Dynamic block computes an approximation to some function  $y=f(x)$  given  $x$ ,  $y$  data vectors. The lookup method can use interpolation, extrapolation, or the original values of the input.

The  $x$  data vector must be strictly monotonically increasing after conversion to the input's fixed-point data type. Note that due to quantization, the  $x$  data vector may be strictly monotonic in doubles format, but not so after conversion to a fixed-point data type.

---

**Note** Unlike the Lookup Table block, the Lookup Table Dynamic block allows you to change the table data without stopping the simulation. For example, you may want to automatically incorporate new table data if the physical system you are simulating changes.

---

You define the lookup table by inputting the  $x$  and  $y$  table data to the block as 1-by- $n$  vectors. To help reduce the ROM used by the code generated for this block, you can use different data types for the  $x$  table data and the  $y$  table data. However, these restrictions apply:

- The  $y$  table data and the output vector must have the same sign, the same bias, and the same fractional slope.
- The  $x$  table data and the  $x$  data vector must have the same sign, the same bias, and the same fractional slope. Additionally, the precision and range for the  $x$  data vector must be greater than or equal to the precision and range for the  $x$  table data.

The block generates output based on the input values using one of these methods selected from the **Lookup method** parameter list:

- **Interpolation-Extrapolation**—This is the default method; it performs linear interpolation and extrapolation of the inputs.

- If a value matches the block's input, the output is the corresponding element in the output vector.
- If no value matches the block's input, then the block performs linear interpolation between the two appropriate elements of the table to determine an output value. If the block input is less than the first or greater than the last input vector element, then the block extrapolates using the first two or last two points.
- **Interpolation-Use End Values**—This method performs linear interpolation as described above but does not extrapolate outside the end points of the input vector. Instead, the end-point values are used.
- **Use Input Nearest**—This method does not interpolate or extrapolate. Instead, the element in x nearest the current input is found. The corresponding element in y is then used as the output.
- **Use Input Below**—This method does not interpolate or extrapolate. Instead, the element in x nearest and below the current input is found. The corresponding element in y is then used as the output. If there is no element in x below the current input, then the nearest element is found.
- **Use Input Above**—This method does not interpolate or extrapolate. Instead, the element in x nearest and above the current input is found. The corresponding element in y is then used as the output. If there is no element in x above the current input, then the nearest element is found.

---

**Note** Note that there is no difference among the **Use Input Nearest**, **Use Input Below**, and **Use Input Above** methods when the input x corresponds exactly to table breakpoints.

---

The table data is converted from doubles to the x data type offline using round-to-nearest and saturation.

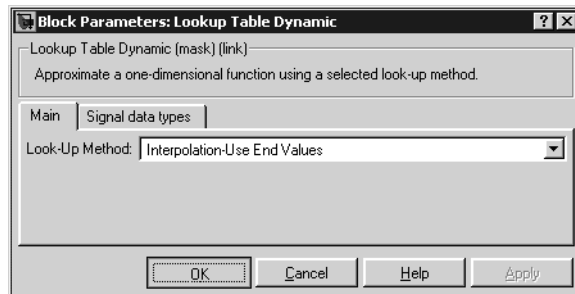
## Data Type Support

The Lookup Table Dynamic block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box

The **Main** pane of the Lookup Table Dynamic block dialog appears as follows:

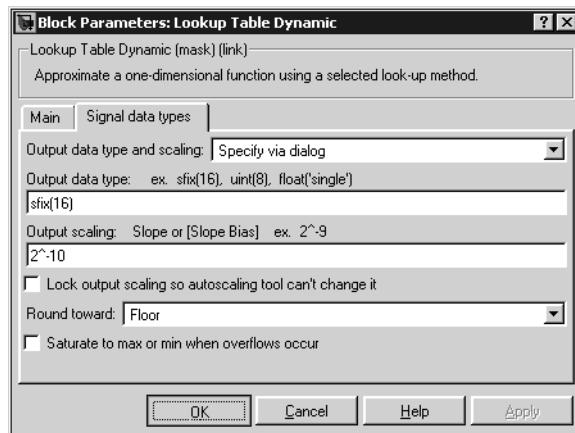
# Lookup Table Dynamic



## Lookup Method

Specify the lookup method.

The **Signal data types** pane of the Lookup Table Dynamic block dialog appears as follows:



## Output data type and scaling

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

## Output data type

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

## Output scaling

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

## Lock output scaling so autoscaling tool can't change it

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

## Round toward

Rounding mode for the fixed-point output.

## Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate.

## Examples

For an example that illustrates the lookup methods supported by this block, see the example included in the Lookup Table block reference pages.

## Characteristics

Direct Feedthrough Yes

Scalar Expansion No

## See Also

Lookup Table, Lookup Table (2-D), Lookup Table (n-D)

# Magnitude-Angle to Complex

**Purpose** Convert a magnitude and/or a phase angle signal to a complex signal

**Library** Math Operations

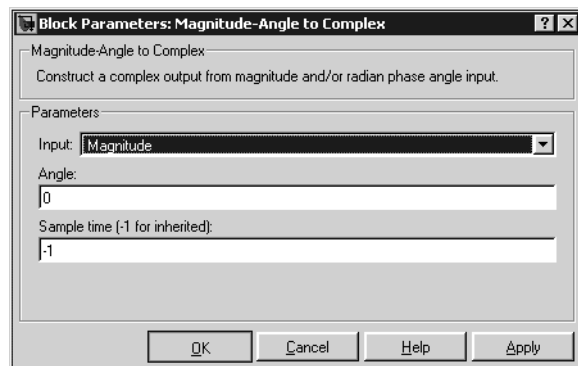
**Description** The Magnitude-Angle to Complex block converts magnitude and/or phase angle inputs to a complex-valued output signal. The inputs must be real-valued signals of type `double`. The angle input is assumed to be in radians. The data type of the complex output signal is `double`.



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 “Specifying Sample Time” in the online documentation for more information.

**Characteristics**

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes, of the input when the function requires two inputs
Dimensionalized	Yes
Zero Crossing	No

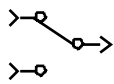
# Manual Switch

---

**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 selected input is propagated to the output, while the unselected input is discarded. You can set the switch before the simulation is started or throw it while the simulation is executing to interactively control the signal flow. The Manual Switch block retains its current state when the model is saved.



**Data Type Support** The Manual Switch block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types.

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

**Parameters and Dialog Box** None

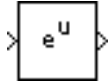
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	N/A
	Dimensionalized	Yes
	Zero Crossing	No



**Purpose** Perform a 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.

- exp
- log
- $10^u$
- log10
- $\text{magnitude}^2$
- square
- sqrt
- pow
- conj
- reciprocal
- hypot
- rem
- mod
- transpose
- hermitian

The block output is the result of the operation of the function on the input or inputs.

The name of the function appears on the block. Simulink automatically draws the appropriate number of input ports.

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.

# Math Function

## Data Type Support

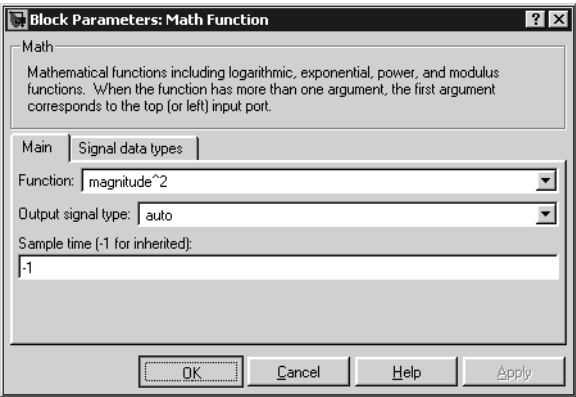
The following table shows which input data types are supported by each of the functions of the Math Function block.

Function	single	double	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	—	—
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
hermitian	yes	yes	yes	yes

All supported modes accept both real and complex inputs, except for reciprocal, which does not accept complex fixed-point inputs. The output signal type of the block is real or complex, depending on the setting of the **Output signal type** parameter.

Parameters and Dialog Box

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



Function

Specify the mathematical function.

Output signal type

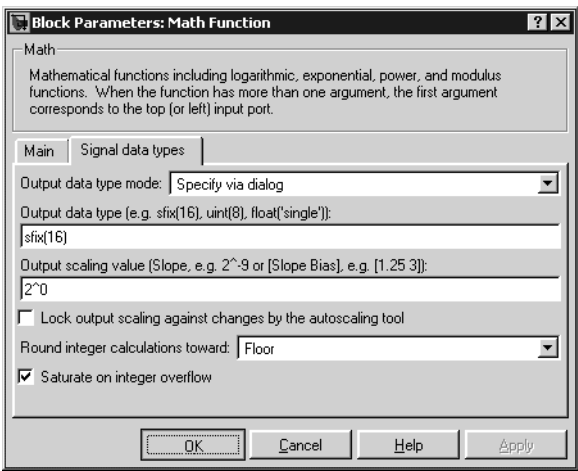
Select the output signal type of the Math Function block as real, complex, or auto.

Function	Input Signal	Output Signal Type		
		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

### Sample time (-1 for inherited)

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

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



---

**Note** The parameters on this pane are only available when the function chosen in the **Function** parameter supports fixed-point data types.

---

### Output data type mode

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

### Output data type

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

## Output scaling value

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

## Lock output scaling so autoscaling tool can't change it

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

## Round integer calculations toward

Select the rounding mode for fixed-point operations.

## Saturate on integer overflow

If selected, fixed-point overflows saturate.

## Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes, of the input when the function requires two inputs
Dimensionalized	Yes
Zero Crossing	No

# MATLAB Fcn

---

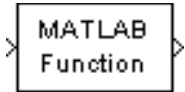
## Purpose

Apply a MATLAB function or expression to the 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)
```

---

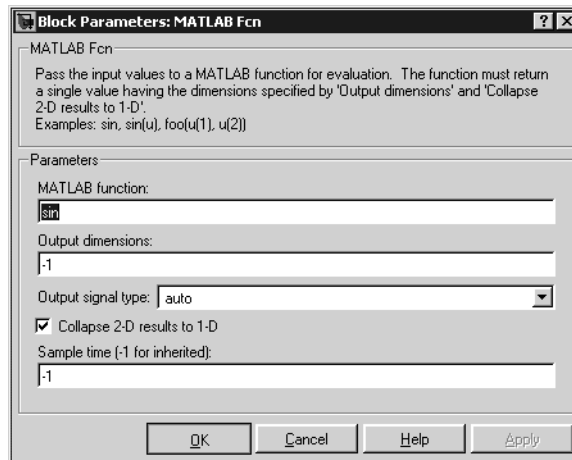
**Note** This block is slower than the Fcn block because it calls the MATLAB parser during each integration step. Consider using built-in blocks (such as the Fcn block or the Math Function block) instead, or writing the function as an M-file or MEX-file S-function, then accessing it using the S-Function 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 “Specifying Sample Time” in the online documentation for more information.

**Characteristics**

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



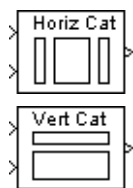
## Purpose

Concatenate inputs horizontally or vertically

## Library

Math Operations

## Description



The Matrix Concatenation block concatenates input matrices  $u_1, u_2, \dots, u_n$  along rows or columns, where  $n$  is specified by the **Number of inputs** parameter. The block accepts inputs with any combination of built-in Simulink data types and/or fixed-point data types. If all inputs are sample-based, the output is sample-based. Otherwise, the output is frame-based.

### Horizontal Matrix Concatenation

When the **Concatenation method** parameter is set to **Horizontal**, the block concatenates the input matrices along *rows*.

$$y = [u_1 \ u_2 \ u_3 \ \dots \ u_n] \quad \% \text{ Equivalent MATLAB code}$$

For horizontal concatenation, inputs must all have the same row dimension,  $M$ , but can have different column dimensions. The output matrix has dimension  $M$ -by- $\sum N_i$ , where  $N_i$  is the number of columns in input  $u_i$  ( $i = 1, 2, \dots, n$ ).

When some of the inputs are length- $M$  1-D vectors while others are  $M$ -by- $N_i$  matrices, the vector inputs are treated as  $M$ -by-1 matrices.

### Vertical Matrix Concatenation

When the **Concatenation method** parameter is set to **Vertical**, the block concatenates the input matrices along *columns*.

$$y = [u_1; u_2; u_3; \dots; u_n] \quad \% \text{ Equivalent MATLAB code}$$

For vertical concatenation, inputs must all have the same column dimension,  $N$ , but can have different row dimensions. The output matrix has dimension  $\sum M_i$ -by- $N$ , where  $M_i$  is the number of rows in input  $u_i$  ( $i = 1, 2, \dots, n$ ).

When some of the inputs are length- $M_i$  1-D vectors while others are  $M_i$ -by-1 matrices, the vector inputs are treated as  $M_i$ -by-1 matrices. (1-D vector inputs are not accepted for vertical concatenation when the other inputs have column dimension greater than 1.)

# Matrix Concatenation

---

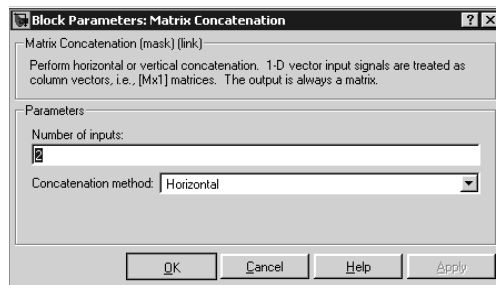
## 1-D Vector Concatenation

When all inputs to the Matrix Concatenation block are length- $M_i$  1-D vectors, the output is a  $\sum M_i$ -by-1 matrix containing all input elements concatenated in port order: the elements in the vector input to the top port appear as the first elements in the output, and the elements in the vector input to the bottom port appear as the last elements in the output.

## Data Type Support

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

## Parameters and Dialog Box



### Number of inputs

The number of matrices to concatenate.

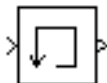
### Concatenation method

The dimension along which to concatenate the inputs.

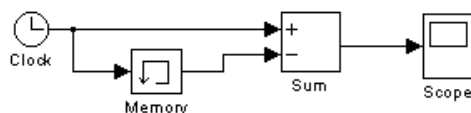
**Purpose** Output the input from the previous time step

**Library** Discrete

**Description** The Memory block outputs its input from the previous time step, applying a one integration step sample-and-hold to its input signal.



This sample model demonstrates how to display the step size used in a simulation. The Sum block subtracts the time at the previous step, generated by the Memory block, from the current time, generated by the clock.

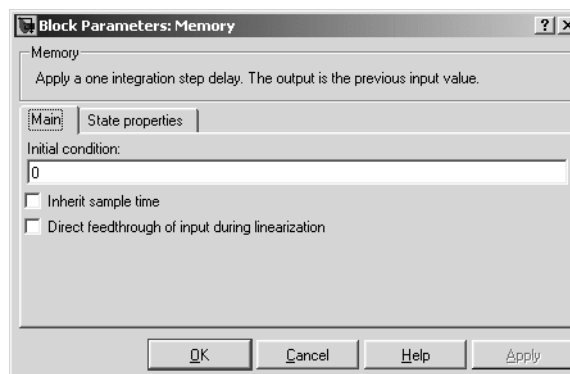


**Note** Avoid using the Memory block when integrating with ode15s or ode113, unless the input to the block does not change.

**Data Type Support** The Memory block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types.

For a discussion of the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using 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.

## Inherit sample time

Check this check box to cause the sample time to be inherited from the driving block. If this option is not selected, the block's sample time depends on the type of solver used to simulate the model. If the solver is a variable-step solver, the sample time is zero-order-hold (ZOH) continuous. If the solver is a fixed-step solver, the sample time equals the solver's step size.

## Direct feedthrough of input during linearization

Causes the block to output its input during linearization and trim. This sets the block's mode to direct feedthrough.

Enabling this check box can cause a change in the ordering of states in the model when using the functions `linmod`, `dlinmod`, or `trim`. To extract this new state ordering, use the following commands.

First compile the model using the following command, where `model` is the name of the Simulink model.

```
[sizes, x0, x_str] = model([],[],[],'lincompile');
```

Next, terminate the compilation with the following command.

```
model([],[],[],'term');
```

The output argument, `x_str`, which is a cell array of the states in the Simulink model, contains the new state ordering. When passing a vector of states as input to the `linmod`, `dlinmod`, or `trim` functions, the state vector must use this new state ordering.

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

Characteristics	Direct Feedthrough	No, except when <b>Direct feedthrough of input during linearization is enabled</b> .
	Sample Time	Continuous, but inherited from the driving block if you select the <b>Inherit sample time</b> check box
	Scalar Expansion	Yes, of the <b>Initial condition</b> parameter
	Dimensionalized	Yes
	Zero Crossing	No

# Merge

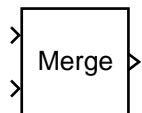
## Purpose

Combine multiple signals into a single signal

## Library

Signal Routing

## Description



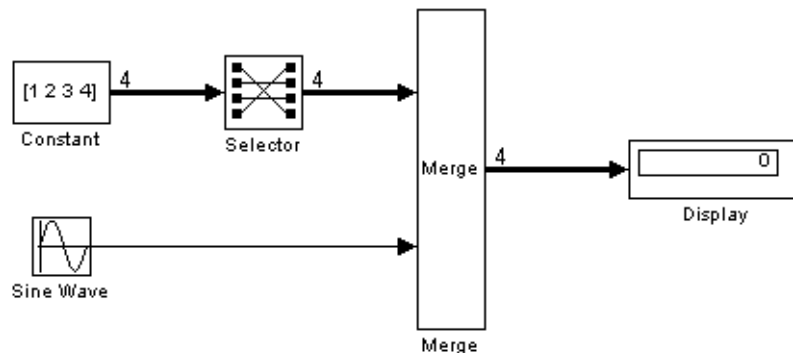
The Merge block combines its inputs into a single output line whose value at any time is equal to the most recently computed output of its driving blocks. You can specify any number of inputs by setting the block's **Number of inputs** parameter.

---

**Note** Merge blocks facilitate creation of alternately executing subsystems. See “Creating Alternately Executing Subsystems” for an application example.

---

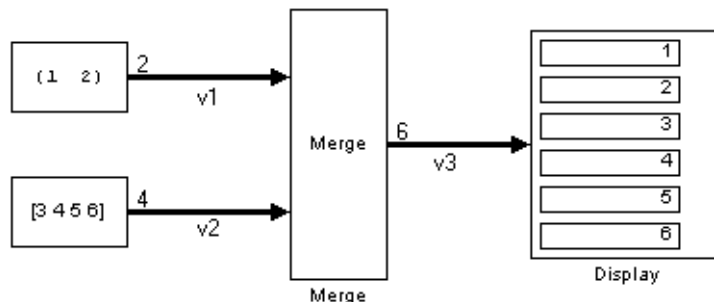
A Merge block does not accept signals whose elements have been reordered. For example, in the following diagram, the Merge block does not accept the output of the Selector block because the Selector block interchanges the first and fourth elements of the vector signal.



If the **Allow unequal port widths** parameter is not selected, the block accepts only inputs of equal dimensions and outputs a signal of the same dimensions as the inputs. If you select the **Allow unequal port widths** option, the block accepts scalars and vectors (but not matrices) having differing numbers of elements. Further, the block allows you to specify an offset for each input signal relative to the beginning of the output signal. The width of the output signal is

$$\max(w_1+o_1, w_2+o_2, \dots, w_n+o_n)$$

where  $w_1, \dots, w_n$  are the widths of the input signals and  $o_1, \dots, o_n$  are the offsets for the input signals. For example, the Merge block in the following diagram merges signals  $v_1$  and  $v_2$  to produce signal  $v_3$ .



In this example, the offset of  $v_1$  is 0 and the offset of  $v_2$  is 2, resulting in an output signal six elements wide. The Merge block maps the elements of  $v_1$  to the first two elements of  $v_3$  and the elements of  $v_2$  to the last four elements of  $v_3$ .

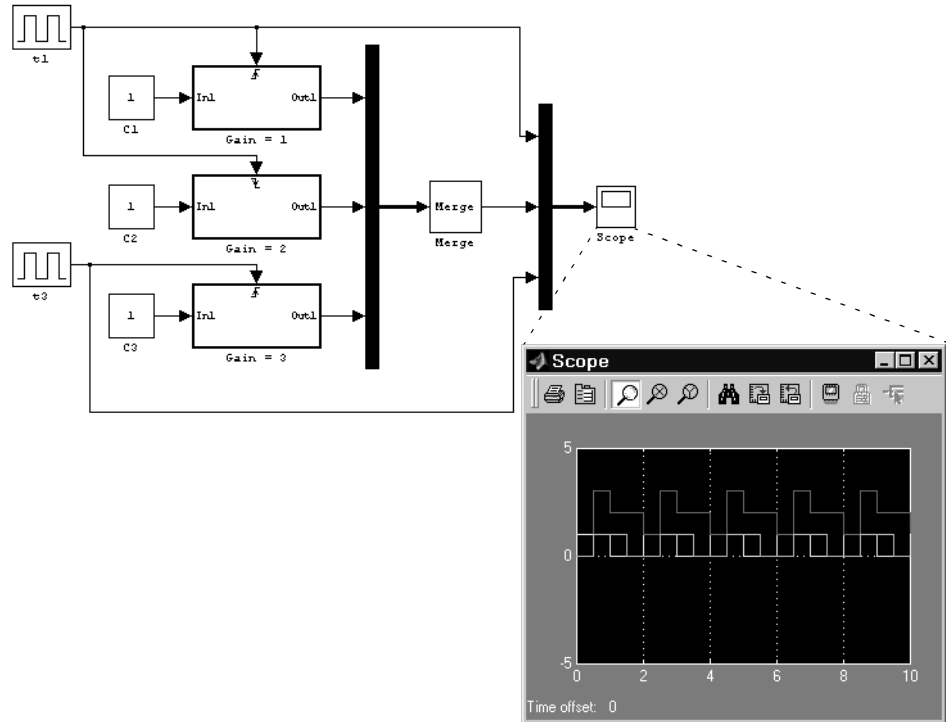
You can specify an initial output value by setting the block's **Initial output** parameter. If you do not specify an initial output and one or more of the driving blocks do, the Merge block's initial output equals the most recently evaluated initial output of the driving blocks.

## Merging S-Function Outputs

The Merge block does can merge a signal from an S-Function block only if the memory used to store the S-Function block's output is reusable. Simulink displays an error message if you attempt to update or simulate a model that connects a nonreusable port of an S-Function block to a Merge block. See `ssSetOutputPortOptimOpts` for more information.

## Muxing Signals to be Merged

Instead of connecting signals directly to a Merge block, you can connect them via a Mux block as illustrated in the following example.



This example connects three amplifiers to a Merge block via a Mux block. The top and bottom amplifiers trigger on a rising pulse; the middle, on a falling pulse. The trigger signal connected to the bottom amplifier has a phase delay of .5 s compared to the trigger signal connected to the bottom amplifier. The output of the Merge block at each time step equals that of the amplifier triggered at that time step. Muxing the signals to be merged rather than connecting them directly to the Merge block can result in a clearer diagram.

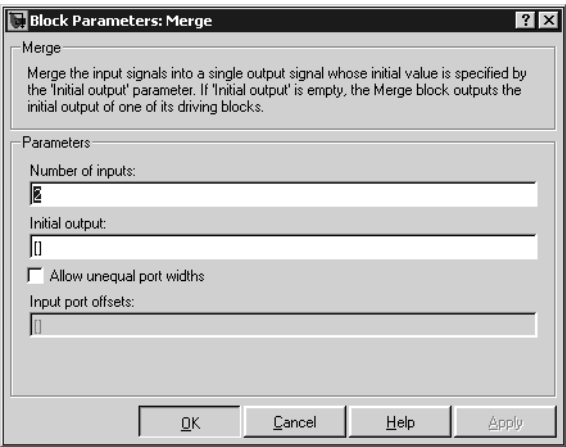
## Data Type Support

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

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

Allow unequal port widths

Allows the block to accept inputs having different numbers of elements.

Input port offsets

Vector specifying the offset of each input signal relative to the beginning of the output signal.

Characteristics

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

# M-File S-Function

**Purpose** Use a Level-2 M-file S-function in a 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.

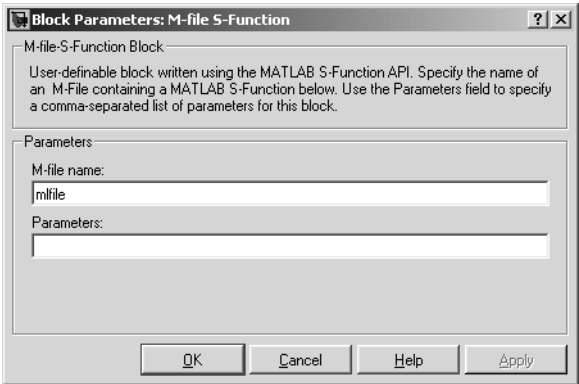


**Note** Use the S-Function block to include a Level-1 M-file S-function in a block.

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.

**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
Zero Crossing	No

# MinMax

**Purpose** Output the minimum or maximum input value

**Library** Math Operations

**Description** The MinMax block outputs either the minimum or the maximum element or elements of the inputs. You can choose the function to apply by selecting one of the choices from the **Function** parameter list.



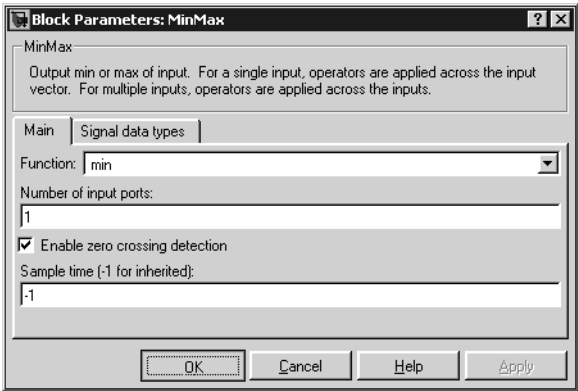
If the block has one input port, the input must be a scalar or a vector. The block outputs a scalar equal to the minimum or maximum element of the input vector.

If the block has multiple input ports, the nonscalar inputs must all have the same dimensions. The block expands any scalar inputs to have the same dimensions as the nonscalar inputs. The block outputs a signal having the same dimensions as the input. Each output element equals the minimum or maximum of the corresponding input elements.

**Data Type Support** The MinMax block accepts and outputs real signals of any data type supported by Simulink, except Boolean. The MinMax block supports fixed-point data types.

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

**Parameters and Dialog Box** The **Main** pane of the MinMax block dialog appears as follows:



## Function

Specify whether to apply the function min or max to the input.

## Number of input ports

Specify the number of inputs to the block.

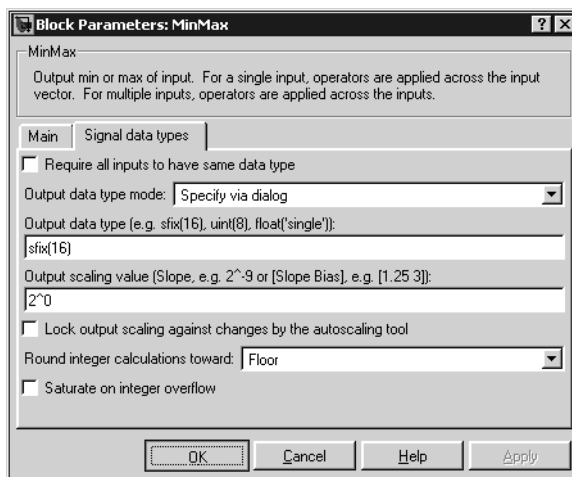
## Enable zero crossing detection

Select to enable zero crossing detection to detect minimum and maximum values. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

## Sample time (-1 for inherited)

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

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



## Require all inputs to have same data type

Select this parameter to require that all inputs must have the same data type.

## Output data type mode

Specify the output data type and scaling by choosing a built-in data type from the drop-down list, or inherit the data type and scaling by an internal

rule or by backpropagation. Lastly, if you select Specify via dialog, the **Output data type**, **Output scaling value**, and **Lock output scaling against changes by the autoscaling tool** parameters become visible.

**Output data type**

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

**Output scaling value**

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

**Lock output scaling against changes by the autoscaling tool**

Select to lock scaling of outputs. This parameter is only visible if you select Specify via dialog for the **Output data type mode** parameter.

**Round integer calculations toward**

Select the rounding mode for fixed-point operations.

**Saturate on integer overflow**

Select to have overflows saturate.

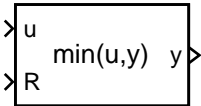
**Characteristics**

Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes, of the inputs
Dimensionalized	Yes

**Purpose** Determine the minimum or maximum of a signal over time

**Library** Math Operations

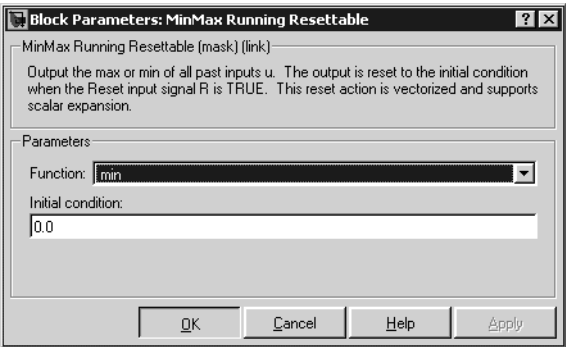
**Description** The MinMax Running Resettable block outputs the minimum or maximum of all past inputs *u*. You specify whether the block outputs the minimum or the maximum with the **Function** parameter.



The block can reset its state based on an external reset signal *R*. When the reset signal *R* is TRUE, the block resets the output to the value of the **Initial condition** parameter.

**Data Type Support** The MinMax Running Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



**Function** Specify whether the block outputs the minimum or the maximum.

**Initial condition** Initial condition.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

# Model

**Purpose** Include a model as a block in another model.

**Library** Ports & Subsystems

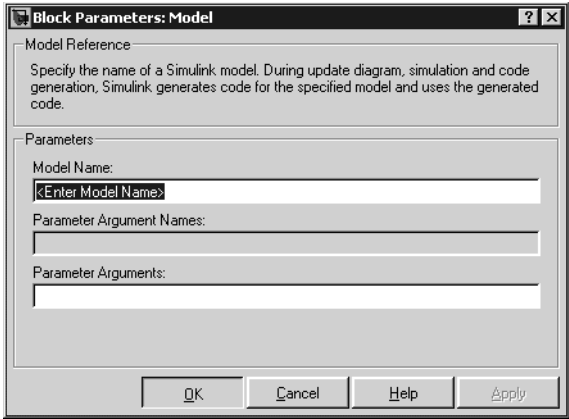
**Description** The Model block allows you to include a model as a block in another model. The Model block displays input ports and output ports corresponding to the model's top-level input and output ports. This allows you to connect the included model to other blocks in the containing model.



During simulation, Simulink invokes an S-function called the simulation target to compute the model's outputs. If the simulation target does not exist at the beginning of a simulation or when you update the model's block diagram, Simulink generates the target from the referenced model. If the target exists, Simulink checks whether the included model has changed since the last time the target was built. If so, Simulink regenerates the target to reflect changes in the model. Simulink uses the same simulation target for all instances of an included model whether in the same model or different model. See Referencing Models for more information.

**Data Type Support** Determined by the root-level inputs and outputs of the model referenced by the Model block.

## Parameters and Dialog Box





## Model Name

Name of the model referenced by this block. The model must exist on the MATLAB path and the MATLAB path must contain no other model having the same name.

## Parameter Argument Names

Names of parameter arguments accepted by the model referenced by this block (see Parameterizing Model References for more information).

## Parameter Arguments

Values to be passed as parameter arguments to the model referenced by this block each time the model is invoked during a simulation. Enter the values in this field as a comma-separated list in the same order as the corresponding argument names appear in the **Parameter Argument Names** field.

## Characteristics

Direct Feedthrough	Depends on model referenced by this block.
Scalar Expansion	Depends on model referenced by this block.

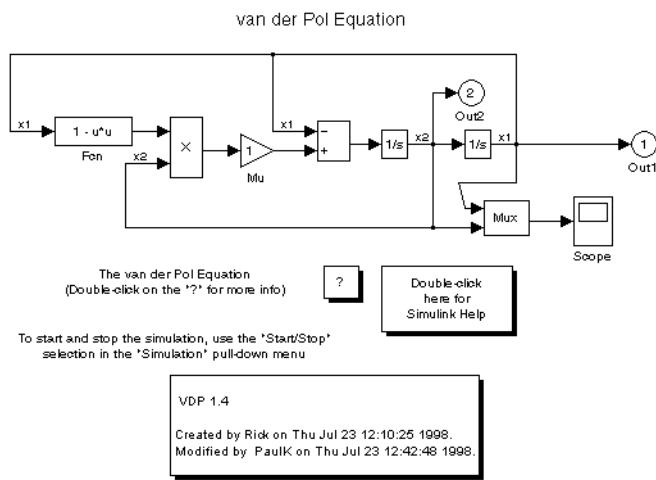
# Model Info

**Purpose** Display revision control information in a model

**Library** Model-Wide Utilities

**Description** The Model Info block displays revision control information about a model as an annotation block in the model's block diagram. The following diagram illustrates use of a Model Info block to display information about the vdp model.

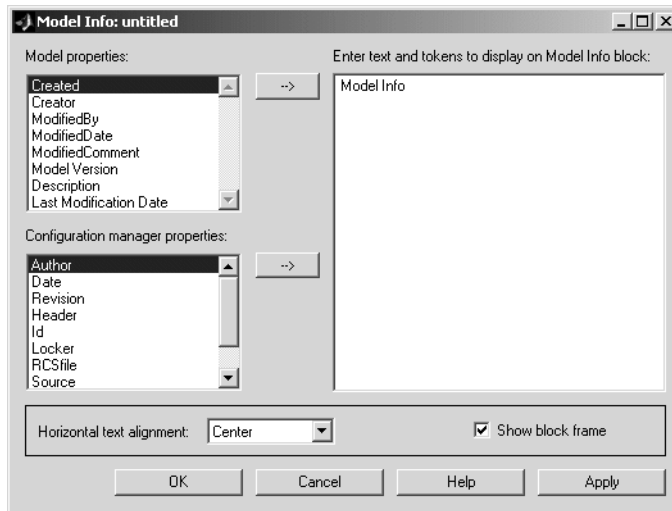
Model Info  
Annotation



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

# Model Info

---

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 “Selecting and Viewing the Source Control System” in the online 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.

---

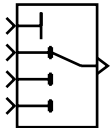
**Note** The selected item does not appear in the Model Info block until you check the model in or out of the repository maintained by the configuration manager and you have closed and reopened the model.

---

**Purpose** Choose between multiple block inputs

**Library** Signal Routing

## Description



The Multiport Switch block chooses between a number of inputs. The first (top) input is called the *control input*, while the rest of the inputs are called *data inputs*. The value of the control input determines which data input is passed through to the output port.

If the control input is an integer value, then the specified data input is passed through to the output. For example, suppose the **Use zero-based indexing** parameter is not selected. If the control input is 1, then the first data input is passed through to the output. If the control input is 2, then the second data input is passed through to the output, and so on. If the control input is not an integer value, the block first truncates the value to an integer by rounding to floor. If the truncated control input is less than 1 or greater than the number of input ports, an out-of-bounds error is returned.

You specify the number of data inputs with the **Number of input ports** parameter. The data inputs can be scalar or vector. The block output is determined by these rules:

- If you specify only one data input and that input is a vector, the block behaves as an “index selector,” and not as a multi-port 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 multi-port 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. Any scalar inputs are expanded to vectors.
- If the inputs are scalar, the output is a scalar.

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 and data inputs of a Multiport Switch block can be signals of any data type supported by Simulink, except Boolean. The Multiport Switch block supports fixed-point data types.

# Multiport Switch

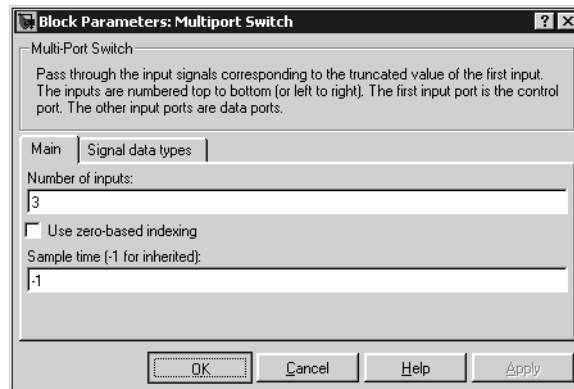
---

The control inputs must be real. The data inputs can be real or complex.

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

## Parameters and Dialog Box

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



### Number of input ports

Specify the number of data inputs to the block.

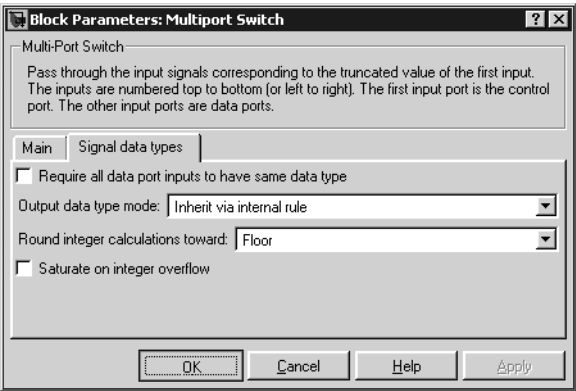
### Use zero based indexing

If selected, the block uses zero-based indexing. Otherwise, the block uses one-based indexing.

### Sample time (-1 for inherited)

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

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



**Require all data port inputs to have same data type**

Select to require all data port inputs to have the same data type.

**Output data type mode**

You can choose to inherit the output data type and scaling by backpropagation or by an internal rule. The internal rule causes the output of the block to have the same data type and scaling as the input with the larger positive range.

**Round integer calculations toward**

Select the rounding mode for fixed-point operations.

**Saturate on integer overflow**

Select to have overflows saturate.

**Characteristics**

Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	No

**Purpose** Combine several input signals into a vector or bus output signal

**Library** Signal Routing

## Description



The Mux block combines its inputs into a single output. An input can be a scalar, vector, or matrix signal. Depending on its inputs, the output of a Mux block is a vector or a composite signal, i.e., a signal containing both matrix and vector elements. If all of a Mux block's inputs are vectors or vector-like, the block's output is a vector. A *vector-like* signal is any signal that is a scalar (one-element vector), a vector, or a single-column or single-row matrix. If any input is a non-vector-like matrix signal, the output of the Mux block is a bus signal. Bus signals can drive only virtual blocks, e.g., Demux, Subsystem, or Goto blocks.

The Mux block's **Number of Inputs** parameter allows you to specify input signal names and dimensionality 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 signals of any dimensionality. Simulink 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 dimensionality 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 displays an error message. A value of `-1` specifies that the corresponding port can accept vectors or matrices of any dimensionality.

- **Cell array**

The length of the cell array specifies the number of inputs. The value of each cell specifies the dimensionality of the corresponding input. A scalar value `N` specifies a vector of size `N`. A vector value `[M N]` specifies an `M`-by-`N` matrix. A value of `-1` means that the corresponding port can accept signals of any dimensionality.



- Signal name list

You can enter a list of signal names separated by commas. Simulink 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.

---

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

---

## Data Type Support

The Mux block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. The Mux block supports mixed-type vectors.

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

## Parameters and Dialog Box




### Number of inputs

The number and dimensionality of inputs. You can enter a comma-separated list of signal names for this parameter field.

Display option

The appearance of the block in the model.

Display Option	Appearance of Block in Model
none	Mux appears inside the block
signals	Displays signal names next to each port
bar	Displays the block in a solid foreground color

<b>Purpose</b>	Create an output port for a subsystem or an external output
<b>Library</b>	Ports & Subsystems, Sinks
<b>Description</b>	Outport blocks are the links from a system to a destination outside the system.
	Simulink 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.
- If you copy an Outport block into a system, its port number is *not* renumbered unless its current number conflicts with an Outport block already in the system. If the copied Outport block port number is not in sequence, you must renumber the block or you will get an error message when you run the simulation or update the block diagram.

## Output 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 automatically renumbers the ports on the blocks.

The Output block name appears in the Subsystem icon as a port label. To suppress display of the label, select the Output block and choose **Hide Name** from the **Format** menu.

## Output Blocks in a Conditionally Executed Subsystem

When an Output block is in an enabled subsystem, you can specify what happens to its output when the subsystem is disabled: it can be reset to an initial value or held at its most recent value. The **Output when disabled** pop-up menu provides these options. The **Initial output** parameter is the value of the output before the subsystem executes and, if the reset option is chosen, while the subsystem is disabled.

## Output Blocks in a Top-Level System

Output 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 “Saving Output to the Workspace”) or the `sim` command (see `sim`). For example, if a system has more than one Output 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 Output block. The column order matches the order of the port numbers for the Output blocks.

If you specify more than one variable name after the second (state) argument, data from each Output block is written to a different variable. For example, if the system has two Output blocks, to save data from Output block 1 to `speed` and the data from Output 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 “Running a Simulation”)

## Data Type Support

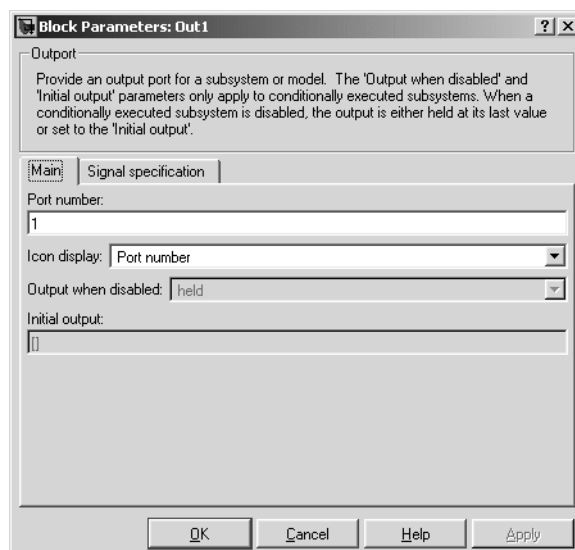
The Outport block accepts complex or real signals of any data type supported by Simulink. An Outport block can also accept fixed-point data types if it is not a root-level outport. The complexity and data type of the block's output are the same as those of its input. For a discussion on the data types supported by Simulink, refer to "Data Types Supported by Simulink" in the Using 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 outport 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 outport's **Initial output** parameter. If the initial output value is in the range of the block's output data type, Simulink 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 halts the simulation and signals an error.

## Parameters and Dialog Box

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



**Port number**

Specify the port number of the Output block.

**Icon Display**

Specify the information to be displayed on the icon of this Output block.  
The options are:

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

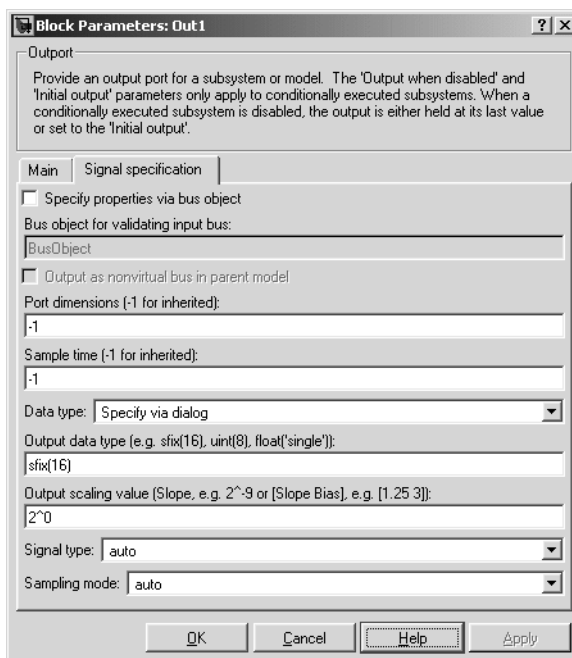
**Output when disabled**

This option is enabled only if the Output resides in an Enabled Subsystem.  
It specifies what happens to the block output when the system is disabled.

**Initial output**

For conditionally executed subsystems, specify the block output before the subsystem executes and while it is disabled. You can specify [ ] if your model does not depend on the initial output of the conditionally executed subsystem.

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



## Specify properties via bus object

Select this option to use a bus object (see “Defining Bus Structures”) to define the properties of a bus connected to this Outputport block.

## Bus object for validating input bus

Specifies the name of the bus object that defines the structure of that a bus must have to be connected to this Outputport block. At the beginning of a simulation or when you update the model’s diagram, Simulink checks whether the bus connected to this block has the specified structure. If not, Simulink displays an error message.

## Output as structure in parent model

Select this option if you want code generated from this model to use a C structure to define the structure of the bus signal output by this block.

## Port dimensions (-1 for inherited)

Specifies the dimensions that a signal must have in order to be connected to this Output block. Valid values are:

-1	A signal of any dimensions can be connected to this port.
N	The signal connected to this port must be a vector of size N.
[R C]	The signal connected to this port must be a matrix having R rows and C columns.

**Sample time (-1 for inherited)**

Specify the sample time of this Output block. See “Specifying Sample Time” in the online documentation for information on specifying sample times. The output of this block changes at the specified rate to reflect the value of its input.

**Data type**

Specify the data type of the signal output by this block. To output any data type, set this parameter to auto.

**Output data type**

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

**Output scaling value**

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



Signal type

Specifies the numeric type of the signal output by this block. The options are:

real	This Output block outputs a real-valued signal. The signal connected to this block must be real. If it is not, Simulink displays an error if you try to update the diagram or simulate the model that contains this block.
complex	This Output block outputs a complex signal. The signal connected to this block must be complex. If it is not, Simulink displays an error if you try to update the diagram or simulate the model that contains this block.
auto	This block outputs the numeric type of the signal that is connected to its input.

Sampling mode

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

Characteristics

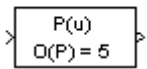
Sample Time	Inherited from driving block
Dimensionalized	Yes

# Polynomial

**Purpose** Perform evaluation of polynomial coefficients on input values

**Library** Math Operations

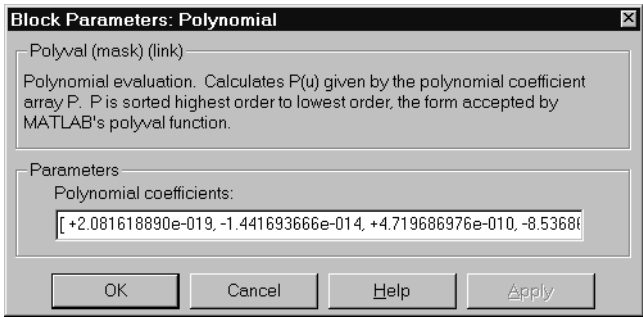
**Description** The Polynomial block uses a coefficients parameter to evaluate a real polynomial for the input value.



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.

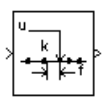
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Prelookup Index Search

**Purpose** First stage of high-performance constant or linear interpolation that performs index search and interval fraction calculation for input on a breakpoint set

**Library** Lookup Tables

**Description** The PreLookup Index Search block calculates the indices and interval fractions for the input value in the **Breakpoint data** parameter. By using this combination of blocks, you can replace multiple Interpolation (n-D) blocks with one set of PreLookup Index Search blocks. In models that have many interpolation blocks simulation performance can be greatly increased.



To use this block, you must define a set of breakpoint values. In normal use, this breakpoint data set corresponds to one dimension of a **Table data** parameter in an Interpolation (n-D) using PreLookup block. The block generates a pair of outputs for each input value by calculating the index of the breakpoint set element that is less than or equal to the input value and the resulting fractional value that is a number  $0 \leq f < 1$  that represents the input value's normalized position between the index and the next index value for in-range input.

For example, if the breakpoint data is

[ 0 5 10 20 50 100 ]

and the input value  $u$  is 55, the (index, fraction) pair is (4, 0.1), denoted as  $k$  and  $f$  on the block. Note that the index value is zero-based.

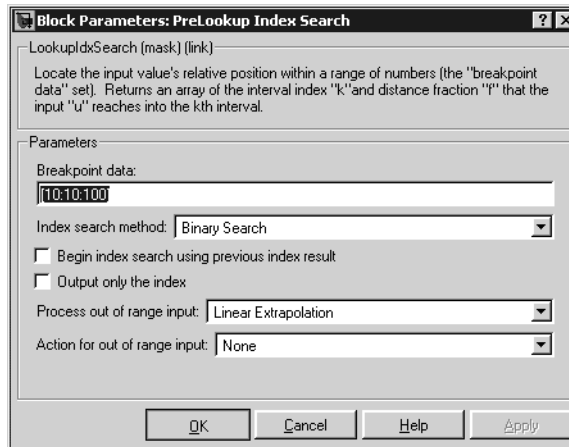
---

**Note** The interval fraction can be negative or greater than 1 for out-of-range input. See the documentation for the block's **Process out of range input parameter** for more information.

---

**Data Type Support** The PreLookup Index Search block accepts signals of types double or single, but for any given block the inputs must all be of the same type. The **Breakpoint data** parameter must be of the same type as the inputs. The output data type is set to the input data type.

## Parameters and Dialog Box



### Breakpoint data

The set of numbers to search.

### Index search method

Binary search, evenly spaced points, or linear search. Use linear search in combination with **Begin index search using previous index result** for higher performance than a binary search when the input values do not change much from one time step to the next. For large breakpoint sets, a linear search can be very slow if the input value changes by more than a few intervals from one time step to the next.

### Begin index search using previous index result

Select this option if you want the block to start its search using the index that was found on the previous time step. For inputs that change slowly with respect to the interval size, you can realize a large performance gain.

### Output only the index

If this block is not being used to feed an Interpolation (n-D) Using PreLookup block, the interval fraction output can be dropped and the resulting index value output is a uint32 instead.

**Process out of range input**

Specifies how to handle out-of-range input. Options include:

- **Clip to Range**  
If the input is less than the first breakpoint, return the index of the first breakpoint (i.e., 0) and 0 for the interval fraction. If the input is greater than the last breakpoint, return the index of the next-to-the-last breakpoint and 1 for the interval fraction. For example, suppose the range is [1 2 3] and you select this option. Then, if the input is 0.5, the block returns [0 0]; if the input is 3.5, the block returns [1 1].
- **Linear Extrapolation**  
If the input is less than the first breakpoint, return the index of the first breakpoint and an interval fraction representing the linear distance from the input to the first breakpoint. If the input is greater than the last breakpoint, return the index of the next-to-the-last breakpoint and an interval fraction that represents the linear distance from the next-to-the-last breakpoint to the input. For example, suppose the range is [1 2 3] and you select this option. Then, if the input is 0.5, the block returns [0 -0.5]; if the input is 3.5, the block returns [1 1.5].

**Action for out of range input**

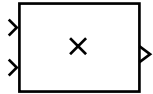
Specifies whether to produce a warning or error message if the input is out of range. The options are None (the default, no warning or error message), Warning (display a warning message in the MATLAB command window and continue the simulation), Error (halt the simulation and display an error message in the Simulation Diagnostics Viewer).

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving blocks
	Scalar Expansion	Yes
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Multiply or divide inputs

**Library** Math Operations

**Description** The Product block performs multiplication or division of its inputs.



This block produces outputs using either element-wise or matrix multiplication, depending on the value of the **Multiplication** parameter. You specify the operations with the **Number of inputs** parameter. Multiply(\*) and divide(/) characters indicate the operations to be performed on the inputs:

- If there are two or more inputs, then the number of characters must equal the number of inputs. For example, “\*/\*” requires three inputs. For this example, if the **Multiplication** parameter is set to Element-wise, the block divides the elements of the first (top) input by the elements of the second (middle) input, and then multiplies by the elements of the third (bottom) input. In this case, all nonscalar inputs to this block must have the same dimensions.

If, however, the **Multiplication** parameter is set to Matrix, the block output is the matrix product of the inputs marked “\*” and the inverse of inputs marked “/”, with the order of operations following the entry in the **Number of inputs** parameter. The dimensions of the inputs must be such that the matrix product is defined.

---

**Note** To perform a dot product on input vectors, use the Dot Product block.

---

- If only multiplication of inputs is required, then a numeric parameter value equal to the number of inputs can be supplied instead of “\*” characters. This may be used in conjunction with either element-wise or matrix multiplication.
- If a single vector is input and the **Multiplication** parameter is set to Element-wise, then a single “\*” will cause the block to output the scalar product of the vector elements. A single “/” will cause the block to output the inverse of the scalar product of the vector elements.
- If a single matrix is input and the **Multiplication** parameter is set to Element-wise, then a single “\*” or “/” will cause the block to error out. If,

however, the **Multiplication** parameter is set to **Matrix**, then a single “\*” will cause the block to output the matrix unchanged, and a single “/” will cause the block to output the inverse of the matrix.

The Product block first performs the specified multiply or divide operations on the inputs, and then converts the results to the output data type using the specified rounding and overflow modes.

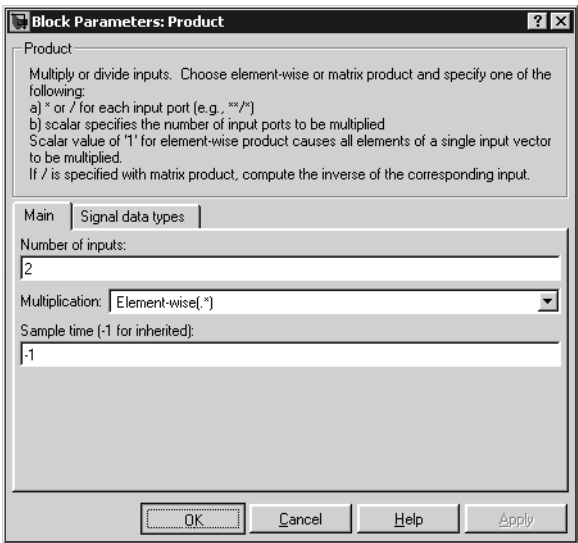
## Data Type Support

The Product block accepts real or complex signals of any data type supported by Simulink, except Boolean. The Product block supports fixed-point data types.

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

## Parameters and Dialog Box

The **Main** pane of the Product dialog appears as follows:



### Number of inputs

Enter the number of inputs or a combination of “\*” and “/” symbols. See “Description” above for a complete discussion of this parameter.



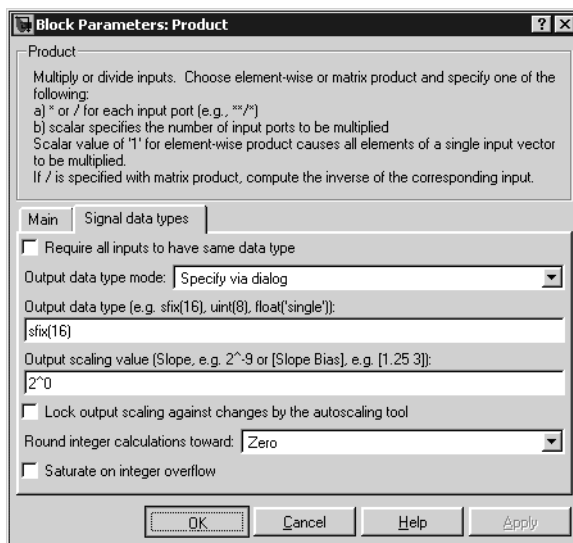
## Multiplication

Specify element-wise or matrix multiplication. See “Description” above for a complete discussion of this parameter.

## Sample time (-1 for inherited)

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

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



## Require all inputs to have same data type

Select this parameter to require that all inputs have the same data type.

## Output data type mode

Specify the output data type and scaling to be the same as the first input, or inherit the data type and scaling by an internal rule or by backpropagation. You can also choose a built-in data type from the drop-down list. Lastly, if you choose **Specify via dialog**, the **Output data type**, **Output scaling value**, and **Lock output scaling against changes by the autoscaling tool** parameters become visible.

If you select `Inherit via internal rule` for this parameter, Simulink chooses a combination of output scaling and data type that requires the smallest amount of memory consistent with accommodating the output range and maintaining the output precision (and avoiding underflow in the case of division operations). If the **Device type** parameter on the **Hardware Implementation** pane of the **Configuration Parameters** dialog is set to `custom`, Simulink chooses the data type without regard to hardware constraints. Otherwise, Simulink chooses the smallest available hardware data type capable of meeting range, precision, and underflow constraints. For example, if the block multiplies inputs of type `int8` and `int16` and `custom` is specified as the device type, the output data type is `sfix24`. If `Unspecified` (assume 32-bit generic) is specified, the output data type is `int32`. If none of the word lengths provided by the target hardware can accommodate the output range, Simulink displays an error message in the Simulation Diagnostics Viewer.

**Output data type**

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

**Output scaling value**

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

**Lock output scaling against changes by the autoscaling tool**

Select to lock scaling of outputs. This parameter is only visible if you select `Specify via dialog` for the **Output data type mode** parameter.

**Round integer calculations toward**

Select the rounding mode for fixed-point operations.

**Saturate on integer overflow**

Select to have overflows saturate.

**Characteristics**

Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes

Dimensionalized	Yes
Zero Crossing	No

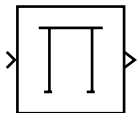
# Product of Elements

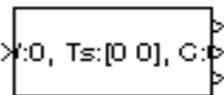
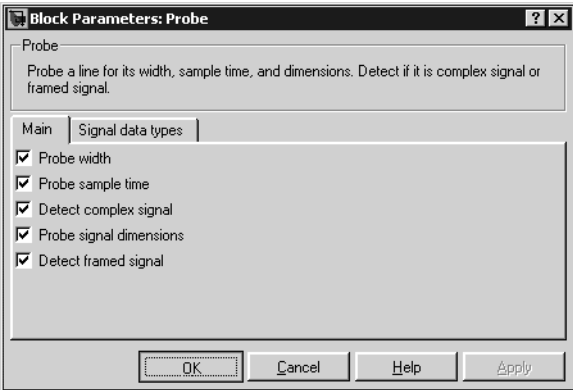
---

**Purpose** Multiply or divide inputs

**Library** Math Operations

**Description** The Product of Elements block is an implementation of the Product block. See Product on page 1-359 for more information.



<b>Purpose</b>	Output a signal's attributes, including width, dimensionality, sample time, and/or complex signal flag
<b>Library</b>	Signal Attributes
<b>Description</b>	<p>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.</p> 
<b>Data Type Support</b>	<p>The Probe block accepts and outputs any data type supported by Simulink, including fixed-point data types.</p> <p>For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.</p>
<b>Parameters and Dialog Box</b>	<p>The <b>Main</b> pane of the Probe block dialog appears as follows:</p> 

### 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.

## Probe complex signal

Select to output 1 if the probed signal is complex; otherwise, 0.

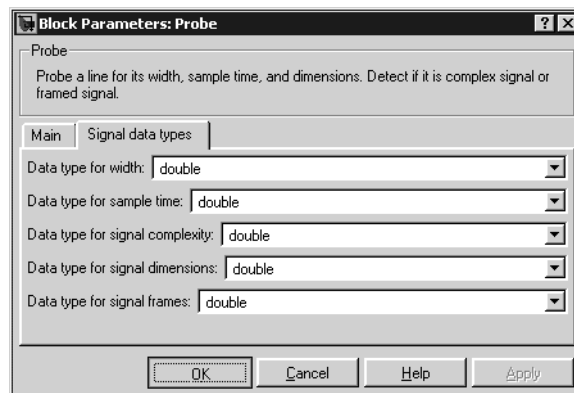
## Probe signal dimensions

Select to output the dimensions of the probed signal.

## Detect framed signal

Select to output 1 if the probed signal is framed; otherwise, 0.

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



---

**Note** The Probe block ignores the **Data Type Override** setting of the **Fixed-Point Settings** interface.

---

## Data type for width

Select the output data type for the width information.

## Data type for sample time

Select the output data type for the sample time information.

## Data type for signal complexity

Select the output data type for the complexity information.

**Data type for signal dimensions**

Select the output data type for the dimensions information.

**Data type for signal frames**

Select the output data type for the frames information.

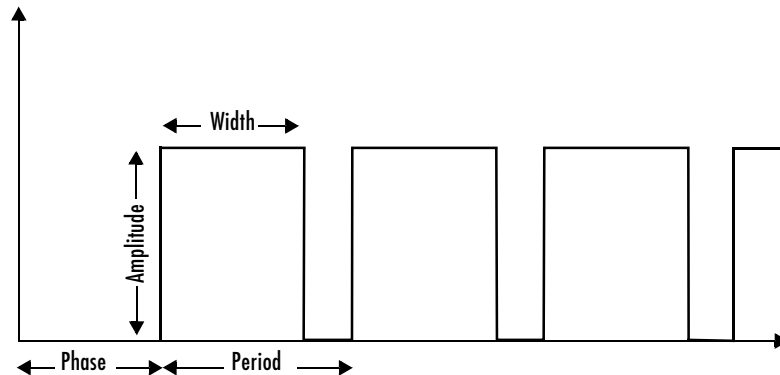
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Yes
	Dimensionalized	Yes
	Zero Crossing	No

# Pulse Generator

**Purpose** Generate square wave pulses at regular intervals

**Library** Sources

**Description** The Pulse Generator block generates square wave pulses at regular intervals. The block's waveform parameters, **Amplitude**, **Pulse Width**, **Period**, and **Phase Delay**, determine the shape of the output waveform. The following diagram shows how each parameter affects the waveform.



The Pulse Generator can emit scalar, vector, or matrix signals of any real data type. To cause the block to emit a scalar signal, use scalars to specify the waveform parameters. To cause the block to emit a vector or matrix signal, use vectors or matrices, respectively, to specify the waveform parameters. Each element of the waveform parameters affects the corresponding element of the output signal. For example, the first element of a vector amplitude parameter determines the amplitude of the first element of a vector output pulse. All the waveform parameters must have the same dimensions after scalar expansion. The data type of the output is the same as the data type of the **Amplitude** parameter.

The block's **Pulse type** parameter allows you to specify whether the block's output is time-based or sample-based. If you select *sample-based*, the block computes its outputs at fixed intervals that you specify. If you select *time-based*, Simulink computes the block's outputs only at times when the output actually changes. This can result in fewer computations being required to compute the block's output over the simulation time period.



Depending on the pulse's waveform characteristics, the intervals between changes in the block's output can vary. For this reason, Simulink cannot use a fixed solver to compute the output of a time-based pulse generator. Simulink allows you to specify a fixed-step solver for models that contain time-based pulse generators. However, in this case, Simulink computes a fixed sample time for the time-based pulse generators. It then simulates the time-based pulse generators as sample-based.

---

**Note** If you use a fixed-step solver and the **Pulse type** is time-based, you must choose the step size such that the period, phase delay, and pulse width (in seconds) are integer multiples of the step size. For example, suppose that the period is 4 seconds, the pulse width is 75% (i.e., 3 s), and the phase delay is 1 s. In this case, the computed sample time is 1 s. Therefore, you must choose a fixed-step size that is 1 or that divides 1 exactly (e.g., 0.25). You can guarantee this by setting the fixed-step solver's step size to auto on the **Configuration Parameters** dialog box.

---

If you select time-based as the block's pulse type, you must specify the pulse's phase delay and period in units of seconds. If you specify sample-based, you must specify the block's sample time in seconds, using the **Sample Time** parameter, then specify the block's phase delay and period as integer multiples of the sample time. For example, suppose that you specify a sample time of 0.5 second. And suppose you want the pulse to repeat every two seconds. In this case, you would specify 4 as the value of the block's **Period** parameter.

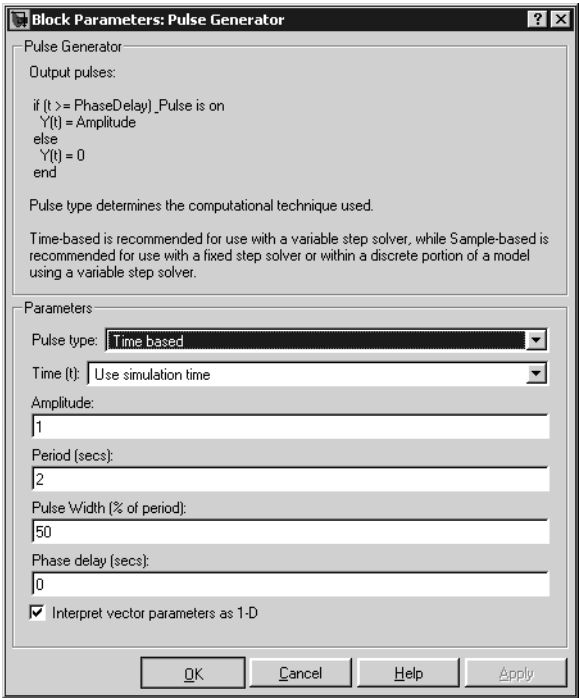
## Data Type Support

The Pulse Generator block outputs real signals of any data type supported by Simulink. The data type of the output signal is the same as that of the **Amplitude** parameter.

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

# Pulse Generator

## Parameters and Dialog Box



### 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 signal's time variable. If you specify an external source, the block displays an input port for connecting the source.

### 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 2.

### 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 50 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.

# Pulse Generator

---

## 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 “Specifying Sample Time” for more information.

## Interpret vector parameters as 1-D

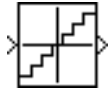
If you select this option and the other parameters are one-row or one-column matrices, after scalar expansion, the block outputs a 1-D signal (vector). Otherwise the output dimensionality is the same as that of the other parameters.

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

**Purpose** Discretize input at a 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.

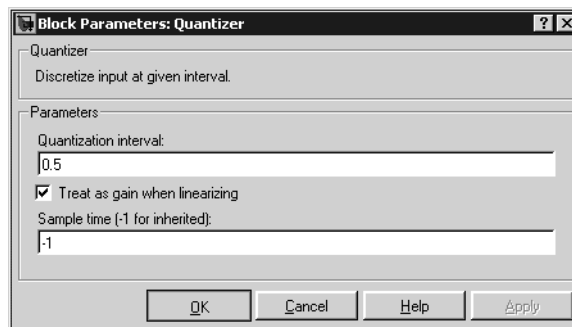
$$y = q * \text{round}(u/q)$$

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 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.

# Quantizer

---

<b>Characteristics</b>	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

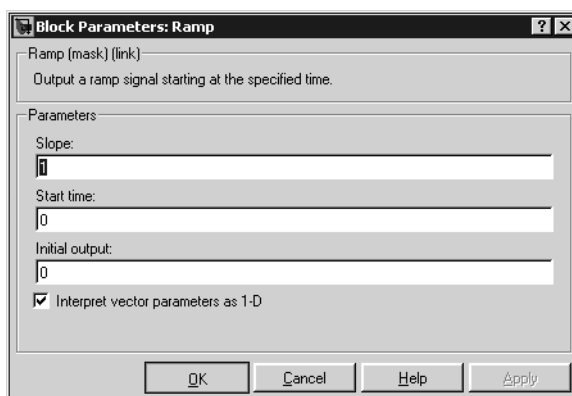
**Library** Sources

**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**, **Duty Cycle**, 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**



## 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.

# Ramp

---

<b>Characteristics</b>	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 **Initial seed** parameter as a vector.

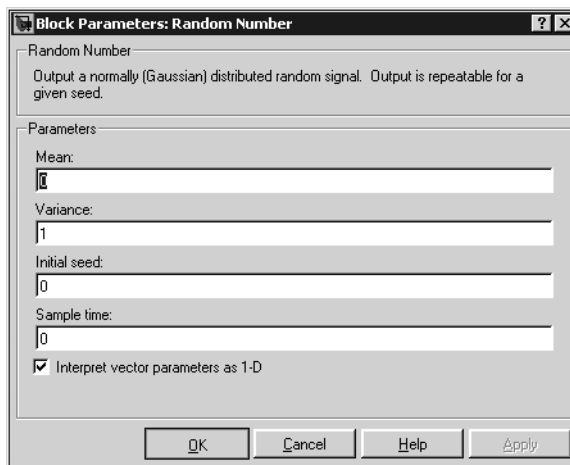
To generate uniformly distributed random numbers, use the Uniform Random Number block.

Avoid integrating a random signal, because solvers are meant to integrate relatively smooth signals. Instead, use the Band-Limited White Noise block.

All the block's numeric parameters must be of the same dimension after scalar expansion.

**Data Type Support** The Random Number block accepts and outputs signals of type double.

## Parameters and Dialog Box



# Random Number

---

**Mean**

The mean of the random numbers. The default is 0.

**Variance**

The variance of the random numbers. The default is 1.

**Initial seed**

The starting seed for the random number generator. The seed must be 0 or a positive integer. The default is 0.

**Sample time**

The time interval between samples. The default is 0, causing the block to have continuous sample time. See “Specifying Sample Time” in the online documentation for more information.

**Interpret vector parameters as 1-D**

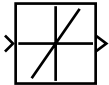
If you select this option and the other parameters are one-row or one-column matrices, after scalar expansion, the block outputs a 1-D signal (vector). Otherwise, the output dimensionality is the same as that of the other parameters.

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of parameters
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Limit the rate of change of a signal

**Library** Discontinuities

## Description



The Rate Limiter block limits the first derivative of the signal passing through it. The output changes no faster than the specified limit. The derivative is calculated using this equation.

$$rate = \frac{u(i) - y(i-1)}{t(i) - t(i-1)}$$

$u(i)$  and  $t(i)$  are the current block input and time, and  $y(i-1)$  and  $t(i-1)$  are the output and time at the previous step. The output is determined by comparing  $rate$  to the **Rising slew rate** and **Falling slew rate** parameters:

- If  $rate$  is greater than the **Rising slew rate** parameter ( $R$ ), the output is calculated as

$$y(i) = \Delta t \cdot R + y(i-1)$$

- If  $rate$  is less than the **Falling slew rate** parameter ( $F$ ), the output is calculated as

$$y(i) = \Delta t \cdot F + y(i-1)$$

- If  $rate$  is between the bounds of  $R$  and  $F$ , the change in output is equal to the change in input:

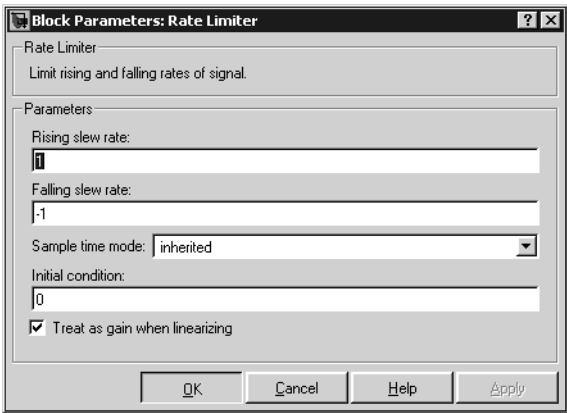
$$y(i) = u(i)$$

## Data Type Support

The Rate Limiter block accepts and outputs signals of any data type supported by Simulink, except Boolean. The Rate Limiter block supports fixed-point data types.

# Rate Limiter

## 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.

### Treat as gain when linearizing

Linearization commands in Simulink treat this block as a gain in state space. Select this check box to cause the linearization commands to treat the gain as 1; otherwise, the commands treat the gain as 0.

## Characteristics

Direct Feedthrough	Yes
Sample Time	Continuous or inherited (specified in the <b>Sample time mode</b> parameter)
Scalar Expansion	Yes, of input and parameters

Dimensionalized	Yes
Zero Crossing	No

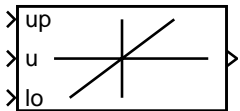
**See Also**      Rate Limiter Dynamic

# Rate Limiter Dynamic

**Purpose** Limit the rising and falling rates of the signal

**Library** Discontinuities

**Description** The Rate Limiter Dynamic block limits the rising and falling rates of the signal.

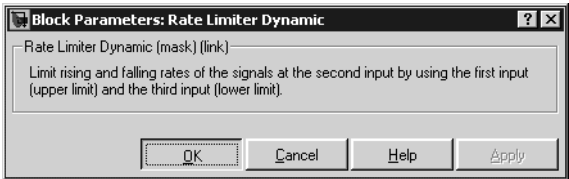


The external signal up sets the upper limit on the rising rate of the signal.

The external signal lo sets the lower limit on the falling rate of the signal.

**Data Type Support** The Rate Limiter Dynamic block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



**Characteristics**

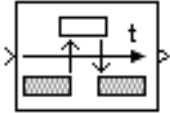
Direct Feedthrough	Yes
Scalar Expansion	Yes

**See Also** Rate Limiter

**Purpose** Handle transfer of data between blocks operating at different rates

**Library** Signal Attributes

**Description** The Rate Transition block transfers data from the output of a block operating at one rate to the input of another block operating at a different rate. The Rate Transition block's parameters allow you to specify options that trade data integrity and deterministic transfer for faster response and/or lower memory requirements.



---

**Note** See “Data Transfer Problems” in the online Real-Time Workshop documentation for a discussion of data integrity and deterministic data transfer.

---

In particular, the block supports the following options:

- Deterministic transfer of data with data integrity between blocks operating at different speeds at the cost of maximum latency of data transfer. This is the default option.
- Nondeterministic data transfer with minimum latency and assured data integrity but increased memory requirements

To specify this option, check the **Ensure data integrity during data transfer** parameter and uncheck the **Ensure deterministic data transfer** parameter.

- Minimum latency and target size at the cost of nondeterministic data transfer and possible loss of data integrity

To specify this option, uncheck the **Ensure data integrity during data transfer** and **Ensure deterministic data transfer** parameters.

See “Sample Rate Transitions” in the online Real-Time Workshop documentation for more information.

# Rate Transition

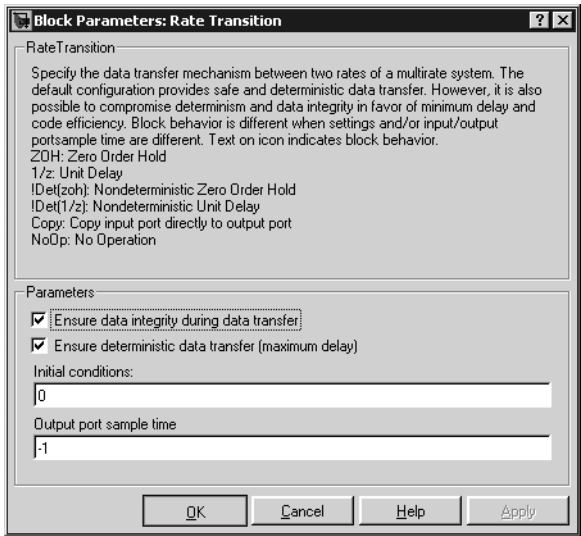
**Note** The Zero-Order Hold and Unit Delay blocks also enable transfer of data between blocks operating at different rates. However, you should use the Rate Transition block for this purpose because it offers a wider range of options and is easier to use.

## Data Type Support

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

For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the online Simulink documentation.

## Parameters and Dialog Box



### Ensure data integrity during data transfer

Selecting this option results in generation of code that ensures the integrity of data transferred by the Rate Transition block. If you select this option and the transfer is nondeterministic (see **Ensure deterministic data transfer** option below), the generated code uses double-buffering to prevent the fast block from interrupting the data transfer. Otherwise the generated code uses a copy operation to effect the data transfer. The copy operation consumes less memory than double-buffering but is also



interruptible and hence can lead to loss of data during nondeterministic data transfers. Thus, you should select this option if you want the generated code to operate both with maximum responsiveness (i.e., nondeterministically) and assured data integrity. See “Rate Transition Block Options” in the online Real-Time Workshop documentation for more information.

**Ensure deterministic data transfer (maximum delay)**

Selecting this option causes code generation to generate code that transfers data at the sample rate of the slower block, i.e., deterministically. If this option is not selected, data transfers occur as soon as new data is available from the source block and the receiving block is ready to receive the data. This avoids the need to delay transfers, thus ensuring that the system operates with maximum responsiveness. However, it also means that transfers can occur unpredictably, which is undesirable in some applications. See “Rate Transition Block Options” in the online Real-Time Workshop documentation for more information.

**Initial conditions**

This parameter applies only to Slow to fast transitions. It specifies the Rate Transition’s initial output at the beginning of a transition when there is not yet any output from the slow block connected to the Rate Transition block’s input.

**Output port sample time**

Specifies the output rate to which the input rate is converted. The default value (- 1) specifies that the output rate is inherited from the block to which the Rate Transition block’s output port is connected. See “Specifying Sample Time” in the online documentation for information on how to specify the output rate.

<b>Characteristics</b>	Direct Feedthrough	No for slow-to-fast transitions that are protected, i.e., for which you have checked the <b>Ensure data integrity during data transfer</b> option; otherwise, yes.
	Sample Time	This block supports discrete-to-discrete and discrete-to-continuous transitions.
	Scalar Expansion	Yes, of input.

# Rate Transition

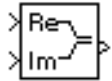
---

Dimensionalized	Yes
Zero Crossing	No

**Purpose** Convert real and/or imaginary inputs to a complex signal

**Library** Math Operations

**Description** The Real-Imag to Complex block converts real and/or imaginary inputs to a complex-valued output signal.



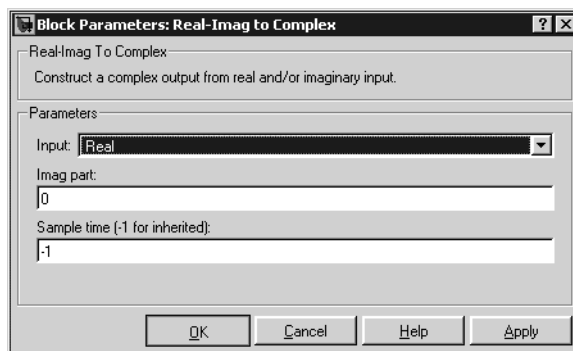
The inputs can both be arrays (vectors or matrices) of equal dimensions, or one input can be an array and the other a scalar. If the block has an array input, the output is a complex array of the same dimensions. The elements of the real input are mapped to the real parts of the corresponding complex output elements. The imaginary input is similarly mapped to the imaginary parts of the complex output signals. If one input is a scalar, it is mapped to the corresponding component (real or imaginary) of all the complex output signals.

The input signals and real or imaginary output parameter can be of any data type supported by Simulink, except Boolean. The Real-Imag to Complex block supports fixed-point data types. The output is of the same type as the input or parameter that determines the output.

For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using 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 to Complex

---

**Real (Imag) part**

If the input is a real-part signal, this parameter specifies the constant imaginary part of the output signal. If the input is the imaginary part, this parameter specifies the constant real part of the output signal. Note that the title of this field changes to reflect its usage.

**Sample time (-1 for inherited)**

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

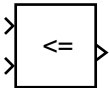
**Characteristics**

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes, of the input when the function requires two inputs
Dimensionalized	Yes
Zero Crossing	No

**Purpose** Perform the specified relational operation on the inputs

**Library** Logic and Bit Operations

**Description** The Relational Operator block performs the specified comparison of its two inputs.



You select the relational operator connecting the two inputs with the **Relational Operator** parameter. The block updates to display the selected operator. The supported operations are given below.

Operation	Description
==	TRUE if the first input is equal to the second input
~=	TRUE if the first input is not equal to the second input
<	TRUE if the first input is less than the second input
<=	TRUE if the first input is less than or equal to the second input
>=	TRUE if the first input is greater than or equal to the second input
>	TRUE if the first input is greater than the second input

You can specify inputs as scalars, arrays, or a combination of a scalar and an array:

- For scalar inputs, the output is a scalar.
- For array inputs, the output is an array of the same dimensions, where each element is the result of an element-by-element comparison of the input arrays.
- For mixed scalar/array inputs, the output is an array, where each element is the result of a comparison between the scalar and the corresponding array element.

# Relational Operator

The input with the smaller positive range is converted to the data type of the other input offline using round-to-nearest and saturation. This conversion is performed prior to comparison.

The output data type is specified with the **Output data type mode** and **Output data type** parameters. The output equals 1 for TRUE and 0 for FALSE.

**Note** The output data type selected should represent zero exactly. Data types that satisfy this condition include signed and unsigned integers and any floating-point data type.

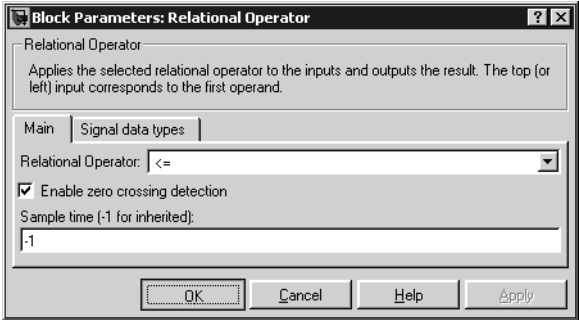
## Data Type Support

The Relational Operator block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. However, if the **Output data type mode** parameter is set to Logical, the input may only be Boolean or double. One input can be real and the other complex if the operator is == or !=.

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

## Parameters and Dialog Box

The **Main** pane of the Relational Operator block appears as follows:



## Relational Operator

Designate the relational operator used to compare the two inputs.

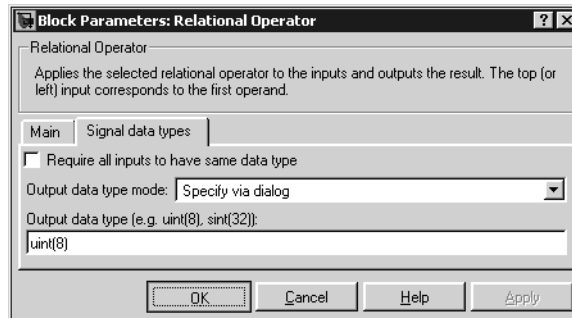
## Enable zero crossing detection

Select to enable zero crossing detection. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

## Sample time (-1 for inherited)

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

The **Signal data types** pane of the Relational Operator block appears as follows:



## Require all inputs to have same data type

Select to require inputs to have the same data type.

## Output data type mode

Set the output data type to Boolean, or choose to specify the data type through the **Output data type** parameter.

Alternatively, you can select Logical to have the output data type determined by the **Implement logic signals as boolean data** parameter in the **Simulation and code generation** optimization pane of the **Configuration Parameters** dialog box. If you select Logical and **Boolean Logic Signals** is on, then the output data type is always Boolean. If you select Logical and **Implement logic signals as boolean data** is off, then the output data type will match the input data type, then the output data type will match the input data type, which is always double.

# Relational Operator

---

## Output data type

Specify the output data type. You should only use data types that represent zero exactly. Data types that satisfy this condition include signed and unsigned integers and any floating-point data type. This parameter is only visible if you select Specify via dialog for the **Output data type mode** parameter.

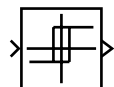
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of inputs
	Dimensionalized	Yes
	Zero Crossing	No, unless you select <b>Enable zero crossing detection</b> .



**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.

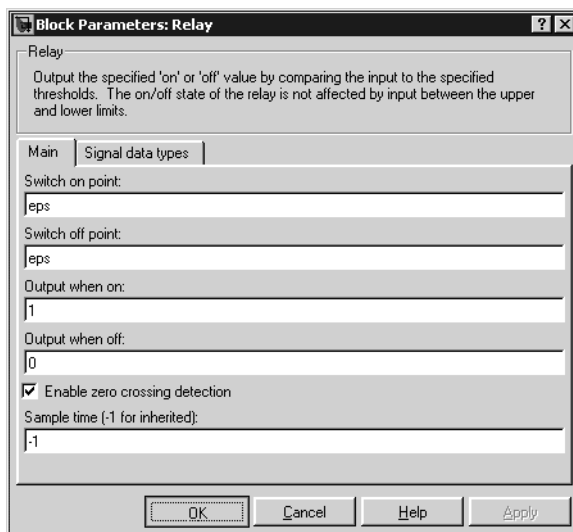
## Data Type Support

The Relay block accepts real or complex signals of any data type supported by Simulink, except Boolean. The Relay block supports fixed-point data types.

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

## Parameters and Dialog Box

The **Main** pane of the Relay block dialog 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.

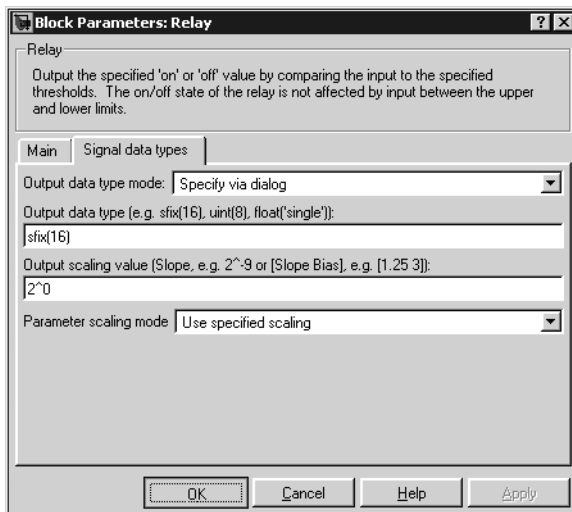
**Enable zero crossing detection**

Select to enable zero crossing detection to detect switch-on and switch-off points. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

**Sample time (-1 for inherited)**

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

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



## Output data type mode

Specify the output data type and scaling to be the same as the inputs, or inherit the data type and scaling by backpropagation. Lastly, if you choose Specify via dialog, the **Output data type**, **Output scaling value**, and **Parameter Scaling** parameters become visible.

## Output data type

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

## Output scaling value

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

## Parameter Scaling

- **Use Specified Scaling**—This mode allows you to specify the output scaling in the **Output scaling value** parameter

- **Best Precision: Vector-wise**—This mode produces a common binary point for each element of the output vector based on the best precision for the largest value of the vector.  
This parameter is only visible if you select **Specify via dialog** for the **Output data type mode** parameter.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes
	Dimensionalized	Yes

**Purpose** Generate an arbitrarily shaped periodic signal

**Library** Sources

## Description

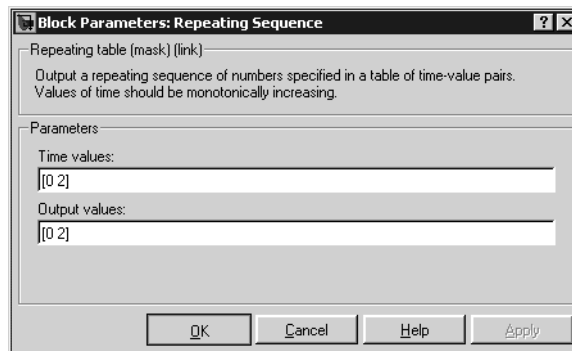


The Repeating Sequence block outputs a periodic scalar signal having a waveform that you specify. You can specify any waveform, using the block dialog's **Time values** and **Output values** parameters. The **Times value** parameter specifies a vector of sample times. The **Output values** parameter 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 (i.e., the signal's period). For example, 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 specified sample points.

## Data Type Support

The Repeating Sequence block outputs real signals of type `double`.

## Parameters and Dialog Box



### Time values

A vector of monotonically increasing time values. The default is `[0 2]`.

### Output values

A vector of output values. Each corresponds to the time value in the same column. The default is `[0 2]`.

# Repeating Sequence

---

<b>Characteristics</b>	Sample Time	Continuous
	Scalar Expansion	No
	Dimensionalized	No
	Zero Crossing	No
<b>See Also</b>	Repeating Sequence Interpolated, Repeating Sequence Stair	

# Repeating Sequence Interpolated

**Purpose** Output discrete-time sequence and repeat, interpolating between data points

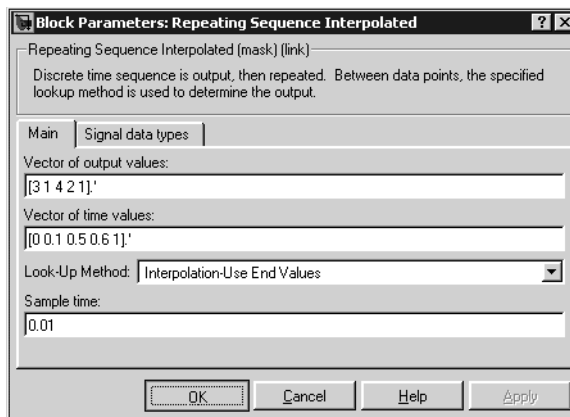
**Library** Sources

**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 any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box** The **Main** pane of the Repeating Sequence Interpolated block dialog 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 a strictly increasing and the vector must have the same size as the vector of output values.

## Lookup Method

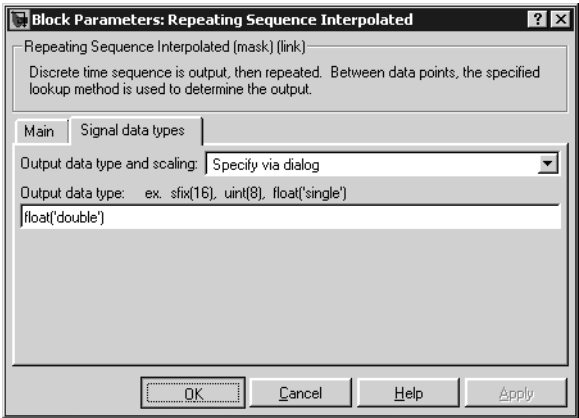
Specify the lookup method to determine the output between data points.

# Repeating Sequence Interpolated

## Sample time (-1 for inherited)

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

The **Signal data types** pane of the Repeating Sequence Interpolated block dialog appears as follows:



## Output data type and scaling

Specify the output data type and scaling via the dialog box, or by inheriting the data type and scaling by backpropagation.

## Output data type

Specify any data type supported by the blockset.

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes

**See Also** Repeating Sequence, Repeating Sequence Stair



## Purpose

Output and repeat the discrete time sequence

## Library

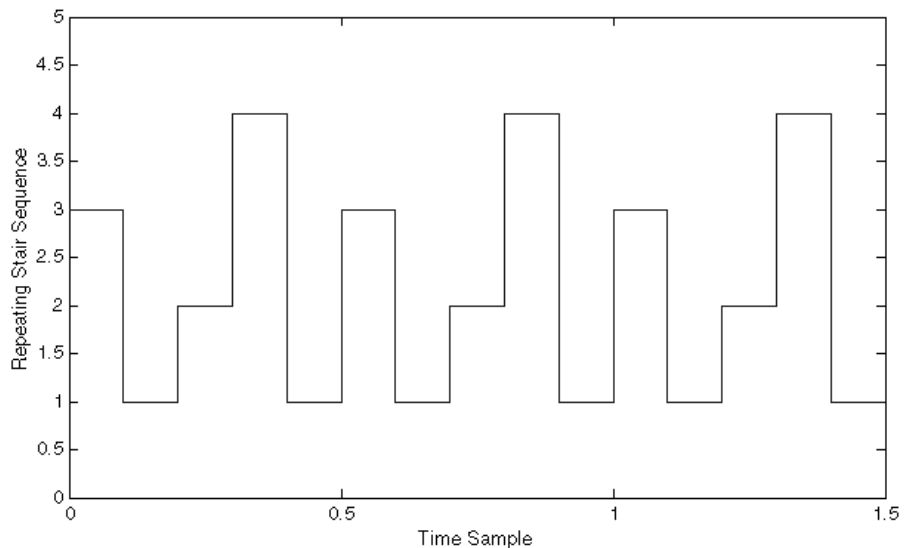
Sources

## Description



The Repeating Sequence Stair block outputs and repeats a discrete time sequence.

You can specify the stair sequence with the **Vector of output values** parameter. For example, the vector can be specified as `[3 1 2 4 1]'`, producing the stair sequence shown in the plot.



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

You can select the output data type and scaling with the **Output data type and scaling** parameter, and set the output data type with the **Output data type** parameter.

For fixed-point data types, you can set the output scaling with the **Output scaling** parameter, and, below that parameter, select the method for scaling the output with the **Output scaling** parameter.

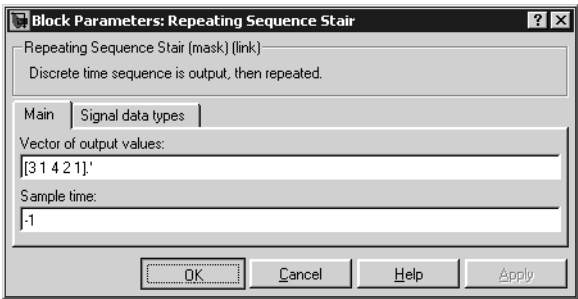
# Repeating Sequence Stair

## Data Type Support

The Repeating Sequence Stair block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box

The **Main** pane of the Repeating Sequence Stair block dialog appears as follows:



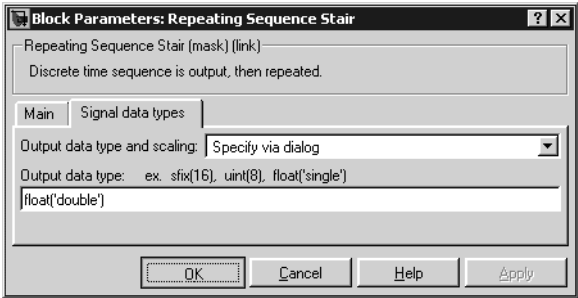
### Vector of output values

Vector containing values of the repeating stair sequence.

### Sample time (-1 for inherited)

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

The **Signal data types** pane of the Repeating Sequence Stair block dialog appears as follows:



### Output data type and scaling

Specify the output data type and scaling via the dialog box, or by inheriting the data type and scaling by backpropagation.

# Repeating Sequence Stair

---

**Output data type**  
Specify any data type supported by the blockset.

**Characteristics**

Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	No

**See Also**      Repeating Sequence, Repeating Sequence Interpolated

**Purpose** Change the dimensionality of a signal

**Library** Math Operations

**Description** The Reshape block changes the dimensionality of the input signal to a dimensionality that you specify, using the block's **Output dimensionality** parameter. For example, you can use the block to change an N-element vector to a 1-by-N or N-by-1 matrix signal, and vice versa.



The **Output dimensionality** parameter lets you select any of the following output options.

Output Dimensionality	Description
1-D array	Converts a matrix (2-D array) to a vector (1-D array) array signal. The output vector consists of the first column of the input matrix followed by the second column, etc. (This option leaves a vector input unchanged.)
Column vector	Converts a vector or matrix input signal to a column matrix, i.e., an M-by-1 matrix, where M is the number of elements in the input signal. For matrices, the conversion is done in column-major order.

# Reshape

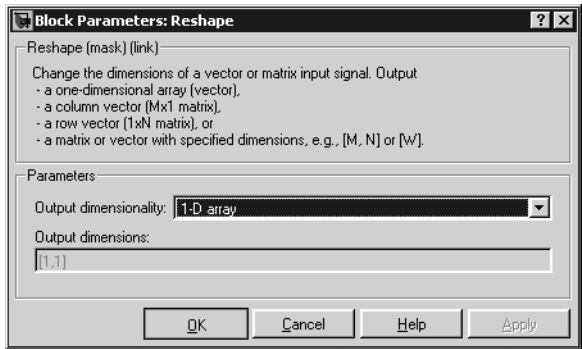
Output Dimensionality	Description
Row vector	Converts a vector or matrix input signal to a row matrix, i.e., a 1-by-N matrix where N is the number of elements in the input signal. For matrices, the conversion is done in column-major order.
Customize	Converts the input signal to an output signal whose dimensions you specify, using the <b>Output dimensions</b> parameter. The value of the <b>Output dimensions</b> parameter can be a one- or two-element vector. A value of [N] outputs a vector of size N. A value of [M N] outputs an M-by-N matrix. The number of elements of the input signal must match the number of elements specified by the <b>Output dimensions</b> parameter. For matrices, the conversion is done in column-major order.

## Data Type Support

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

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

## Parameters and Dialog Box



### Output dimensionality

The dimensionality of the output signal.

**Output dimensions**

Specifies a custom output dimensionality. This option is enabled only if you select Customize as the value of the **Output dimensionality** parameter.

**Characteristics**

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

# Rounding Function

---

**Purpose** Apply a rounding function to a signal

**Library** Math Operations

**Description** The Rounding Function block applies a rounding function to the input signal to produce the output signal.



You can select one of the following rounding functions from the **Function** list:

- **floor**  
Rounds each element of the input signal to the nearest integer value towards minus infinity.
- **ceil**  
Rounds each element of the input signal to the nearest integer towards positive infinity.
- **round**  
Rounds each element of the input signal to the nearest integer.
- **fix**  
Rounds each element of the input signal to the nearest integer towards zero.

The name of the selected function appears on the block.

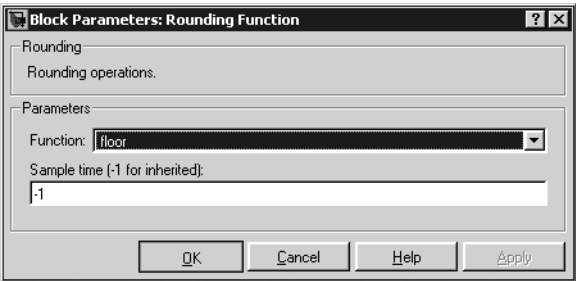
The input signal can be a scalar, vector, or matrix signal having real- or complex-valued elements of type double. The output signal has the same dimensions, data type, and numeric type as the input. Each element of the output signal is the result of applying the selected rounding function to the corresponding element of the input signal.

Use the Rounding Function block instead of the Fcn block when you want vector or matrix output, because the Fcn block can produce only scalar output.

**Data Type Support** The Rounding Function block accepts and outputs real signals of type double or single.



## Parameters and Dialog Box



### Function

The rounding function.

### Sample time (-1 for inherited)

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

### Characteristics

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

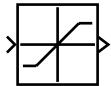
# Rounding Function

---

**Purpose** Limit the range of a signal

**Library** Discontinuities

**Description**



The Saturation block imposes upper and lower bounds on a signal. When the input signal is within the range specified by the **Lower limit** and **Upper limit** parameters, the input signal passes through unchanged. When the input signal is outside these bounds, the signal is clipped to the upper or lower bound.

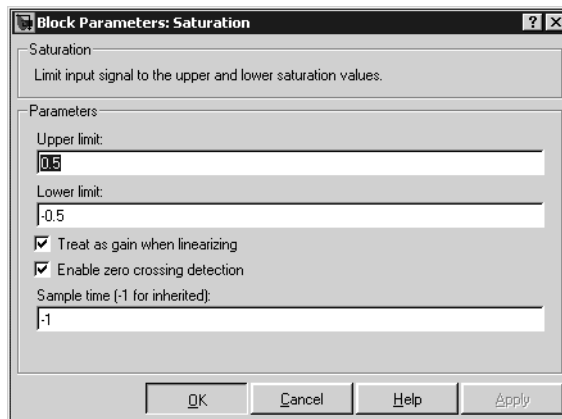
When the **Lower limit** and **Upper limit** parameters are set to the same value, the block outputs that value.

**Data Type Support**

The Saturation block accepts real signals of any data type supported by Simulink, except Boolean. The Saturation block supports fixed-point data types. The output data type is the same as the input data type.

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

**Parameters and Dialog Box**



**Upper limit**

Specify the upper bound on the input signal. When the input signal to the Saturation block is above this value, the output of the block is clipped to this value.

# Saturation

---

The **Upper limit** parameter is converted to the input data type offline using round-to-nearest and saturation.

**Lower limit**

Specify the lower bound on the input signal. When the input signal to the Saturation block is below this value, the output of the block is clipped to this value.

The **Lower limit** parameter is converted to the input data type offline using round-to-nearest and saturation.

**Treat as gain when linearizing**

Linearization commands in Simulink treat this block as a gain in state space. Select this parameter to cause the linearization commands to treat the gain as 1; otherwise, the commands treat the gain as 0.

**Enable zero crossing detection**

Select to enable zero crossing detection. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

**Sample time (-1 for inherited)**

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

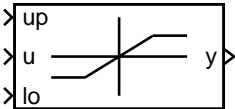
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes, of parameters and input
	Dimensionalized	Yes
	Zero Crossing	No, unless you select <b>Enable zero crossing detection</b> .

**See Also** Saturation Dynamic

**Purpose** Bound the range of the input

**Library** Discontinuities

**Description** The Saturation Dynamic block bounds the range of the input signal to upper and lower saturation values. The input signal outside of these limits saturates to one of the bounds where

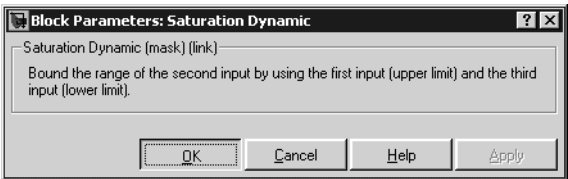


- The input below the lower limit is set to the lower limit.
- The input above the upper limit is set to the upper limit.

The input for the upper limit is the up port, and the input for the lower limit is the lo port.

**Data Type Support** The Saturation Dynamic block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



**Characteristics**

Direct Feedthrough	Yes
Scalar Expansion	Yes

**See Also** Saturation

# Scope, Floating Scope, Signal Viewer Scope

**Purpose** Display signals generated during a simulation

**Library** Sinks

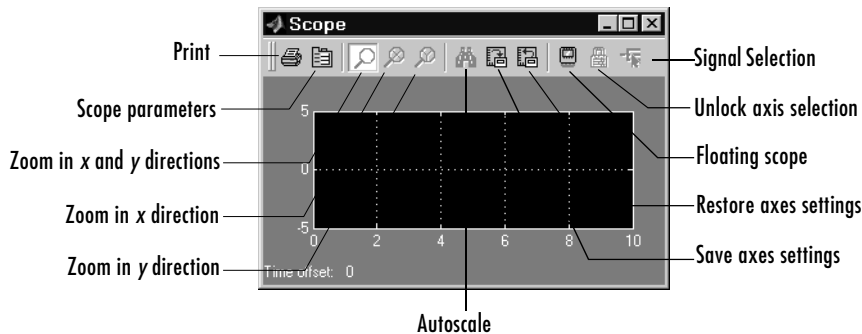
**Description** The Scope block displays its input with respect to simulation time. The Scope block can have multiple axes (one per port); all axes have a common time range with independent y-axes. The Scope allows you to adjust the amount of time and the range of input values displayed. You can move and resize the Scope window and you can modify the Scope's parameter values during the simulation.



When you start a simulation, Simulink does not open Scope windows, although it does write data to connected Scopes. As a result, if you open a Scope after a simulation, the Scope's input signal or signals will be displayed.

If the signal is continuous, the Scope produces a point-to-point plot. If the signal is discrete, the Scope produces a stair-step plot.

The Scope provides toolbar buttons that enable you to zoom in on displayed data, display all the data input to the Scope, preserve axis settings from one simulation to the next, limit data displayed, and save data to the workspace. The toolbar buttons are labeled in this figure, which shows the Scope window as it appears when you open a Scope block.



---

**Note** Do not use Scope blocks inside library blocks that you create. Instead, provide the library blocks with output ports to which scopes can be connected to display internal data.

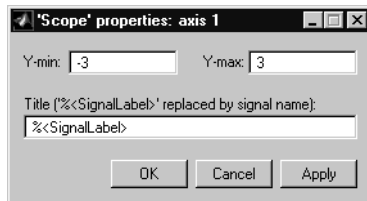
---

## Displaying Vector Signals

When displaying a vector or matrix signal, the Scope assigns colors to each signal element in this order: yellow, magenta, cyan, red, green, and dark blue. When more than six signals are displayed, the Scope cycles through the colors in the order listed.

## Y-Axis Limits

You set y-limits by right-clicking an axis and choosing **Axes Properties**. The following dialog box appears.



### Y-min

Enter the minimum value for the y-axis.

### Y-max

Enter the maximum value for the y-axis.

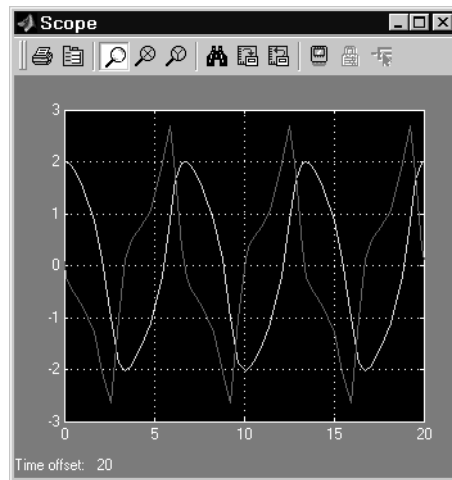
### Title

Enter the title of the plot. You can include a signal label in the title by typing %<SignalLabel> as part of the title string (%<SignalLabel> is replaced by the signal label).

# Scope, Floating Scope, Signal Viewer Scope

## Time Offset

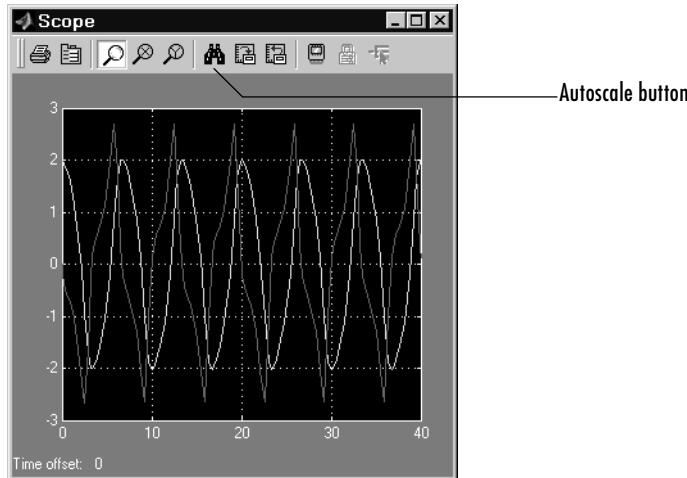
This figure shows the Scope block displaying the output of the vdp model. The simulation was run for 40 seconds. Note that this scope shows the final 20 seconds of the simulation. The **Time offset** field displays the time corresponding to 0 on the horizontal axis. Thus, you have to add the offset to 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 does not buffer the data that it displays on a floating Scope. It can therefore scale the contents of a floating Scope only when data is being displayed, i.e., when a simulation is running. When a simulation is not running, Simulink disables (grays) the **Zoom** button on the toolbar of a floating Scope to indicate that it cannot scale its contents.

---

# Scope, Floating Scope, Signal Viewer Scope

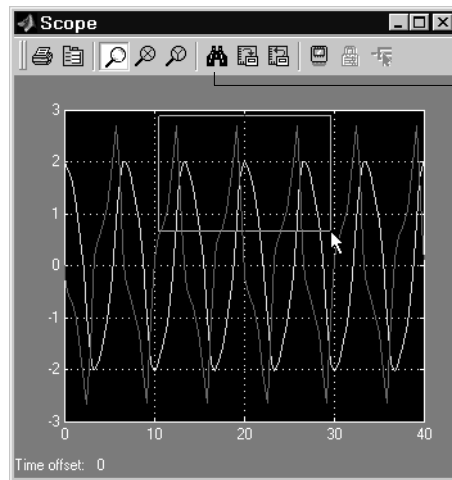
## Zooming

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).

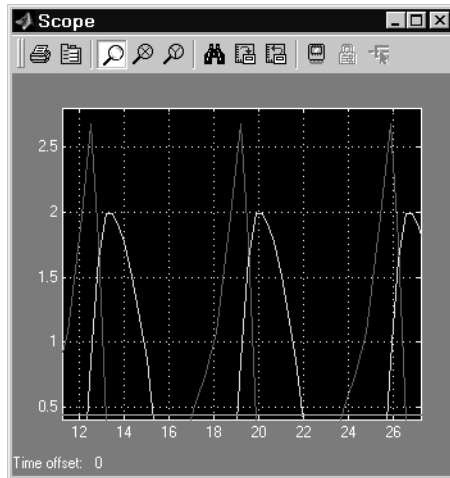
This figure shows a region of the displayed data enclosed within a bounding box.



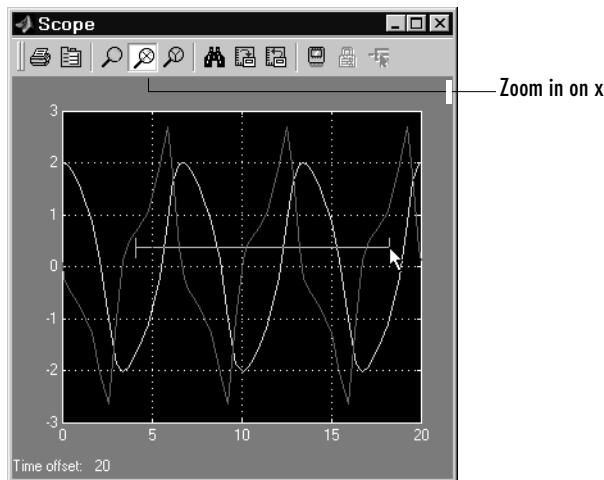
Zoom in both directions

# Scope, Floating Scope, Signal Viewer Scope

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.



# Scope, Floating Scope, Signal Viewer Scope

---

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 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 disables (grays) the **Zoom** button on the toolbar of a floating scope.

---

## Saving the Axes Settings

The **Save axes settings** toolbar button enables you to store the current  $x$ - and  $y$ -axis settings so you can apply them to the next simulation.



You might want to do this after zooming in on a region of the displayed data so you can see the same region in another simulation. The time range is inferred from the current  $x$ -axis limits.

## Scope Parameters

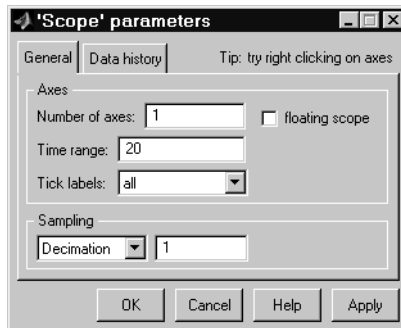
You can change axis limits, set the number of axes, time range, tick labels, sampling parameters, and saving options by choosing the **Parameters** toolbar button.



When you click the **Parameters** button, a dialog box appears. The appearance of the dialog box depends on whether the scope is a Scope block or a Scope

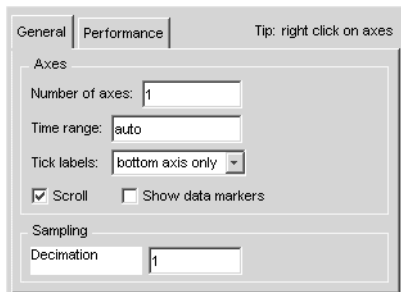
# Scope, Floating Scope, Signal Viewer Scope

viewer created the Signal and Scope Manager. If the scope is a Scope block, this dialog appears.



The dialog box has two panes: **General** and **Data history**. See the next topic for information on the **General** parameters pane. See “Data History Parameters Pane” on page 1-425 for information on the **Data History** parameters pane.

If the scope is a Scope viewer, this dialog box appears.



The dialog box also has two panes: **General** and **Performance**. See the next topic for information on the General parameters pane. See “Performance Parameters Pane” on page 1-426 for information on the **Performance** parameters pane.

# Scope, Floating Scope, Signal Viewer Scope

---

## General Parameters Pane

You can set the axis parameters, time range, and tick labels 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

You can choose to have tick labels on all axes, on one axis, or on the bottom axis only, using the **Tick labels** list.

### Floating scope

This option appears only on the **General** parameters pane for the Scope block.

Selecting this option turns a Scope block into a floating scope. A floating scope is a Scope block that can display the signals carried on one or more lines. You can create a Floating Scope block in a model either by copying a Scope block from the Simulink Sinks library into a model and selecting this option or, more simply, by copying the Floating Scope block from the Sinks library into the model window. The Floating Scope block has the **Floating scope** parameter selected by default.

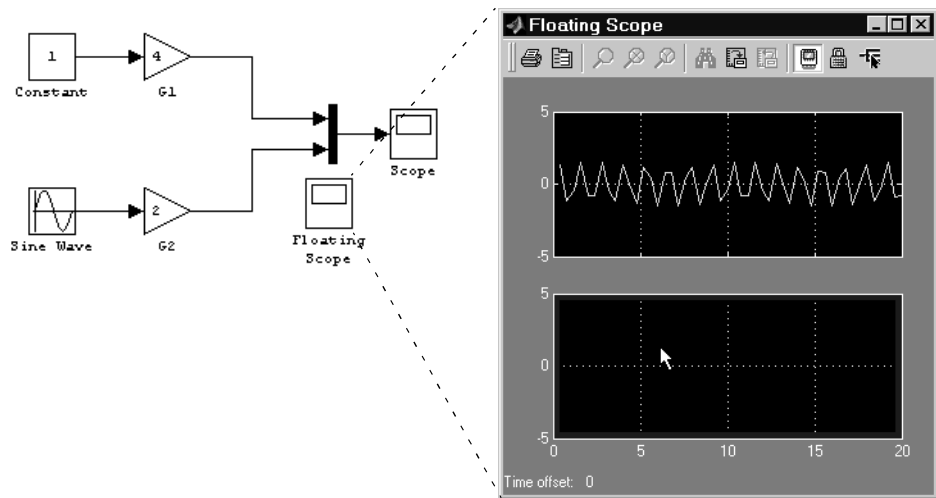
To use a floating scope during a simulation, first open the scope. To display the signals carried on a line, select the line. Hold down the **Shift** key while clicking another line to select multiple lines. It might be necessary to click the **Autoscale data** button on the floating scope's toolbar to find the signal and adjust the axes to the signal values. Or you can use the floating scope's Signal Selector (see "The Signal Selector" in the online Simulink documentation) to select signals for display. To display a floating scope's

# Scope, Floating Scope, Signal Viewer Scope

Signal Selector, first start simulation of your model with the floating scope open. Then right-click your mouse in the floating scope and select **Signal Selection** from the pop-up menu that appears.

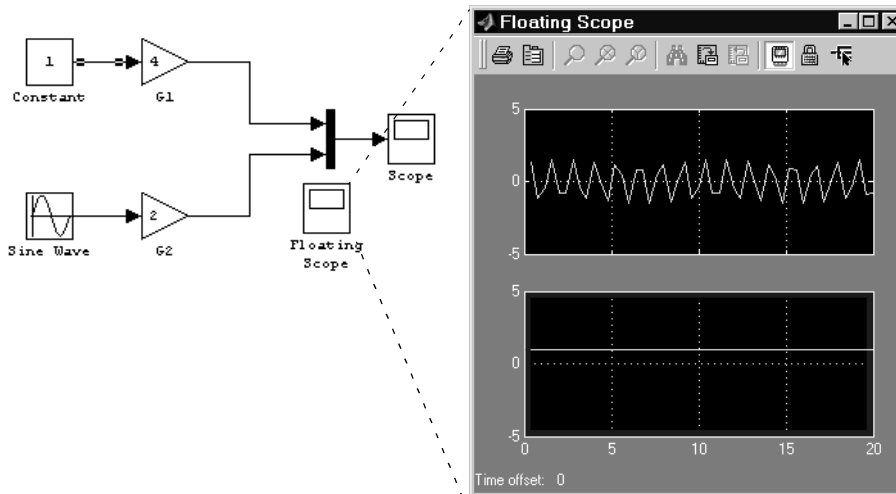
You can have more than one floating scope in a model, but only one set of axes in one scope can be active at a given time. Active floating scopes show the active axes by making them blue. Selecting or deselecting lines affects the active floating scope only. Other floating scopes continue to display the signals that you selected when they were active. In other words, inactive floating scopes are locked, in that their signal displays cannot change.

To specify display of a signal on one of the axes of a multi-axis floating scope, click the axis. Simulink draws a blue border around the axis.



Then click the signal you want to display in the block diagram or the Signal Selector. When you run the model, the selected signal appears in the selected axis.

# Scope, Floating Scope, Signal Viewer Scope



If you plan to use a floating scope during a simulation, you should disable signal storage reuse. See “Signal storage reuse” in “Optimizations” for more information.

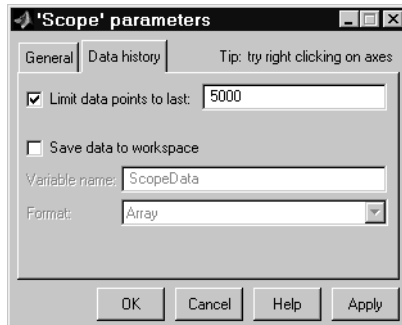
## Sampling

To specify a decimation factor, enter a number in the data field to the right of the **Decimation** choice. To display data at a sampling interval, select the **Sample time** choice and enter a number in the data field.



## Data History Parameters Pane

The **Data History** parameters pane appears only on the **Parameters** dialog box for the Scope viewer. The pane appears as follows.



This pane lets you control the amount of data that the Scope stores and displays. You can also choose to save data to the workspace in this pane. You apply the current parameters and options by clicking the **Apply** or **OK** button. The values that appear in these fields are the values that are used in the next simulation.

### Limit data points to last

You can limit the number of data points saved to the workspace by selecting the **Limit data points to last** check box and entering a value in its data field. The Scope relies on its data history for zooming and autoscaling operations. If the number of data points is limited to 1,000 and the simulation generates 2,000 data points, only the last 1,000 are available for regenerating the display.

### Save data to workspace

You can automatically save the data collected by the Scope at the end of the simulation by selecting the **Save data to workspace** check box. If you select this option, the **Variable name** and **Format** fields become active.

### Variable name

Enter a variable name in the **Variable name** field. The specified name must be unique among all data logging variables being used in the model. Other data logging variables are defined on other Scope blocks, To Workspace blocks, and simulation return variables such as time, states,

# Scope, Floating Scope, Signal Viewer Scope

---

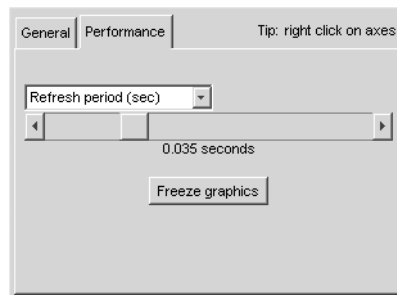
and outputs. Being able to save Scope data to the workspace means that it is not necessary to send the same data stream to both a Scope block and a To Workspace block.

## Format

Data can be saved in one of three formats: Array, Structure, or Structure with time. Use Array only for a Scope with one set of axes. For Scopes with more than one set of axes, use Structure if you do not want to store time data and use Structure with time if you want to store time data.

## Performance Parameters Pane

The **Performance** parameters pane appears only on the **Parameters** dialog box for the Scope viewer. The pane appears as follows.



This pane lets you control how frequently Simulink refreshes the Scope viewer. Reducing the refresh rate can speed up the simulation in some cases. The pane contains the following controls.

### Refresh Period

This list control lets you select the units in which the refresh period is expressed.

### Refresh Slider

Drag the slider button to the right to increase the refresh period and hence decrease the refresh rate.

### Freeze Button

Click the button to freeze (stop refreshing) or unfreeze the Scope viewer.

## Printing the Contents of a Scope Window

To print the contents of a Scope window, open the **Scope Properties** dialog by clicking the **Print** icon, the rightmost icon on the Scope toolbar.



## Data Type Support

The Scope block accepts real signals of any data type supported by Simulink, including fixed-point data types. The Scope block accepts homogeneous vectors.

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

## Characteristics

Sample Time	Inherited from driving block or can be set
States	0

# Selector

**Purpose** Select input elements from a vector or matrix signal

**Library** Signal Routing

**Description** The Selector block generates as output selected elements of an input vector or matrix.



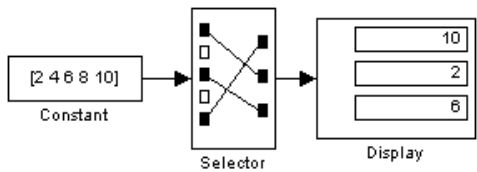
A Selector block accepts either vector or matrix signals as input. Set the **Input Type** parameter to the type of signal (vector or matrix) that the block should accept in your model. The parameter dialog box and the block's appearance change to reflect the type of input that you select. The way the block determines the elements to select differs slightly, depending on the type of input.

## Vector Input

If the input type is vector, a Selector block outputs a vector of selected elements specified by element indices. The meaning of the indices depends on the setting of the **Index mode** parameter. If the setting is One-based (the default), the index of the first input element is 1, the second 2, and so on. If the setting is Zero-based, the index of the first element is 0, the second element 1, and so on.

The block determines the indices of the elements to select either from the block's **Elements** parameter or from an external signal. Set the **Source of element indices** parameter to the source (internal, i.e., parameter value, or external) that you prefer. If you select external, the block adds an input port for the external index signal.

In either case, the elements to be selected must be specified as a vector unless only one element or a range of elements is being selected. For example, this model shows the Selector block and the output for an input vector of [2 4 6 8 10] and an **Elements** parameter value of [5 1 3].



If the block is large enough, it displays the ordering of input vector elements graphically.

If **Use index as starting value** is checked, **Elements** must specify the starting index of a range of elements that starts at the specified index and whose length is specified by **Output port dimensions**. For example, suppose that you want the block to select elements 2 through 4 from a six-element input vector. You could do this by selecting the **Use index as starting value** option, setting the **Output port dimensions** to 3, and setting **Elements** to 2.

If you select `external` as the source for element indices, the block adds an input port for the element indices signal. The signal should specify the elements to be selected in the same way they are specified, using the **Elements** parameter.

If the input type is vector, you must specify the width of the input signal or -1, using the **Input port width** parameter. If you specify a width greater than 0, the width of the input signal must equal the specified width. Otherwise, the block reports an error. If you specify a width of -1, the block accepts a vector signal of any width.

## Matrix Input

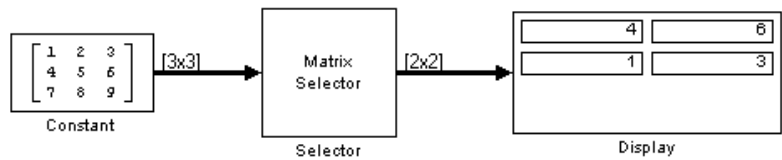
If the input type is matrix, the Selector block outputs a matrix of elements selected from the input matrix. The block determines the row and column indices of the elements to select either from its **Rows** and **Columns** parameters or from external signals. Set the block's **Source of row indices** and **Source of column indices** to the source that you prefer (`internal` or `external`). If you set either source to `external`, the block adds an input port for the external indices signal. If you set both sources to `external`, the block adds two input ports.

In either case, the indices of the row and columns to be selected must be specified as vectors (or a scalar if only one row or column is to be selected or you select the **Use index as starting value** option) of one-based or zero-based indices, depending on the setting of the **Index mode** parameter.

For example, if the **Index mode** is One-based (the default), the **Rows** expression `[ 2 1 ]` and the **Columns** expression `[ 1 3 ]` specify output of a 2-by-2 matrix whose first row contains the first and third elements of the input

# Selector

matrix’s second row and whose second row contains the first and third elements of the input matrix’s first row.



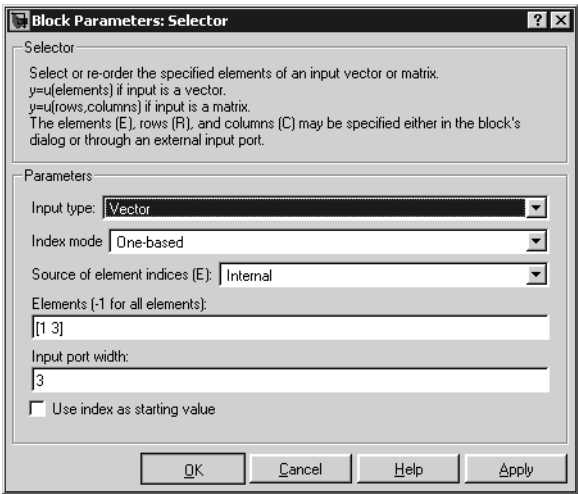
## Data Type Support

The Selector block accepts signals of any signal type and any data type supported by Simulink, including fixed-point data types. The Selector block supports mixed-type signal vectors. The elements of the output vector have the same type as the corresponding selected input elements.

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

## Parameters and Dialog Box

The parameter dialog box appears as follows when you select vector input mode.



## Input Type

The type of the input signal: vector or matrix.

**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.

**Source of element indices**

The source of the indices specifying the elements to select, either internal, i.e., the **Elements** parameter, or external, i.e., an input signal.

**Elements**

The elements to be included in the output vector.

**Input port width**

The number of elements in the input vector.

**Use index as starting value**

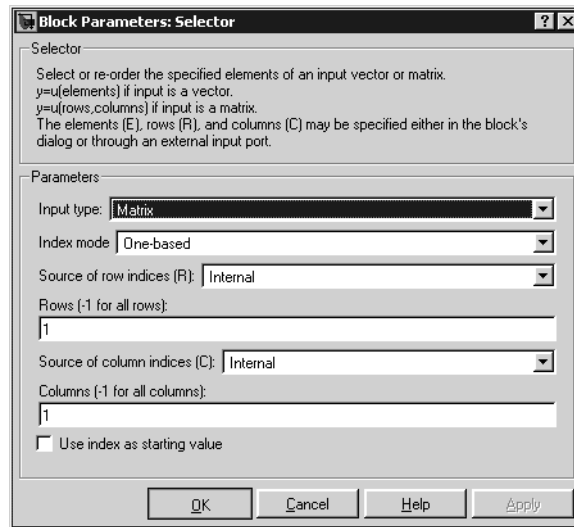
Specifies that the value in the **Elements** field or the external index source is the starting index of a range of elements whose length is the same as the length specified in the **Output port dimensions** field (see next option).

**Output port dimensions**

This field appears only if you check **Use index as starting value**. It specifies the width of the block's output signal.

The dialog box appears as follows when you select matrix input mode.

# Selector



## Input Type

The type of the input signal: vector or matrix.

## Index mode

Specifies the indexing mode: One-based or Zero-based. If One-based is selected, an index of 1 specifies the first row (or column) of the input matrix, 2, the second row, and so on. If Zero-based is selected, an index of 0 specifies the first row (or column) of the input matrix, 0, the second row, and so on.

## Source of row indices

The source of the indices specifying the rows to select from the input matrix, either internal, i.e., the **Rows** parameter, or external, i.e., an input signal.

## Rows

Indices of the rows from which to select elements to be included in the output matrix.



Source of column indices

The source of the indices specifying the columns to select from the input matrix, either internal, i.e., the **Columns** parameter, or external, i.e., an input signal.

Columns

Indices of the columns from which to select elements to be included in the output matrix.

Use index as starting value

Specifies that the values in the Row and Column fields or external index sources specify the starting row and column indexes of a range of elements whose length is the same as the dimensions specified in the **Output port dimensions** field (see next option).

Output port dimensions

This field appears only if you check **Use index as starting value**. It specifies the dimensions of the block’s output signal as a two-element vector: [R C].

Characteristics

Sample Time	Inherited from driving block
Dimensionalized	Yes

# S-Function

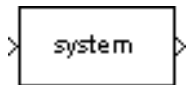
## Purpose

Include an S-function in a 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 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.

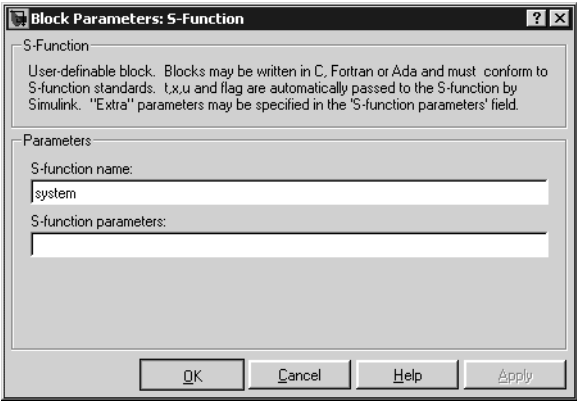
The S-Function block displays the name of the specified S-function and is always drawn with one input port and one output port, regardless of the number of inputs and outputs of the contained subsystem.

Vector lines are used when the S-function contains more than one input or output. The input vector width must match the number of inputs contained in the S-function. The block directs the first element of the input vector to the first input of the S-function, the second element to the second input, and so on. Likewise, the output vector width must match the number of S-function outputs.

## 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.

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
Zero Crossing	No

# S-Function Builder

---

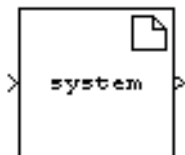
## Purpose

Create an S-function from C code that you provide

## Library

User-Defined Functions

## Description



The S-Function Builder block creates a C MEX-file S-function from specifications and C source code that you provide. See “Building S-Functions Automatically” for detailed instructions on using the S-Function Builder block to generate an S-function.

Instances of the S-Function Builder block also serve as wrappers for generated S-functions in Simulink models. When simulating a model containing instances of an S-Function Builder block, Simulink invokes the generated S-function associated with each instance to compute the instance’s output at each time step.

## Data Type Support

The S-Function Builder can accept and output complex, 1-D or 2-D signals of any data type supported by Simulink.

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

**Purpose** Shift the bits and/or binary point of a signal

**Library** Logic and Bit Operations

**Description** The Shift Arithmetic block can be used to shift the bits or the binary point of a signal, or both.

>

$$\begin{aligned} V_y &= V_u * 2^{-8} \\ Q_y &= Q_u \gg 8 \\ E_y &= E_u \end{aligned}$$

>

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 `sfixed(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

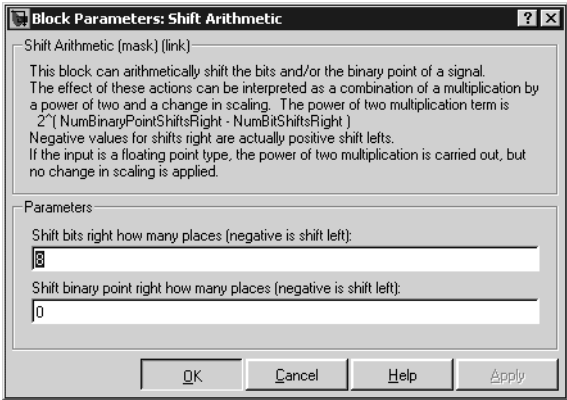
This block performs arithmetic bit shifts on signed numbers. Therefore, the most significant bit is recycled for each bit shift. The effects of bit shifts two places to the right and two places to the left on an input of data type `sfixed(8)` follow.

Shift Operation	Binary Value	Decimal Value
No shift (original number)	11001.011	-6.625
Bit shift right by two places	11110.010	-1.75
Bit shift left by two places	00101.100	5.5

**Data Type Support** The Shift Arithmetic block accepts signals of any data type supported by Simulink, including fixed-point data types, except boolean type.

# Shift Arithmetic

## Parameters and Dialog Box



### Shift bits right how many places (negative is shift left)

The number of places the bits of the input signal is shifted. A positive value indicates a shift right, while a negative value indicates a shift left.

### Shift binary point right how many places (negative is shift left)

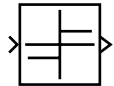
The number of places the binary point of the input signal is shifted. A positive value indicates a shift right, while a negative value indicates a shift left.

Characteristics	Direct Feedthrough	Yes
	Sample Time	Inherited
	Scalar Expansion	Yes

**Purpose** Indicate the sign of the input

**Library** Math Operations

**Description** The Sign block indicates the sign of the input:

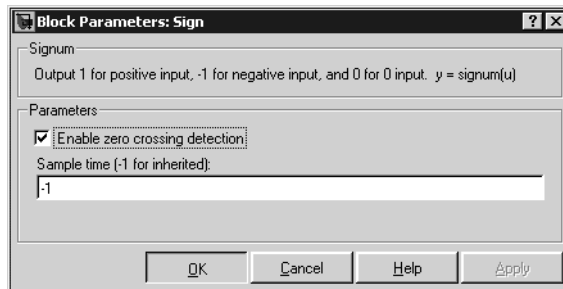


- The output is 1 when the input is greater than zero.
- The output is 0 when the input is equal to zero.
- The output is -1 when the input is less than zero.

**Data Type Support** The Sign block accepts real or complex signals of any data type supported by Simulink, 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, refer to “Data Types Supported by Simulink” in the Using 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 Using Simulink documentation.

### Sample time (-1 for inherited)

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

# Sign

---

Characteristics	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	N/A
	Dimensionalized	Yes



**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 Using Simulink documentation for more information.



**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 **Block Parameters** from the block’s context menu.

<b>Characteristics</b>	Sample Time	Continuous
	Scalar Expansion	Yes, of parameters
	Dimensionalized	Yes
	Zero Crossing	No

# Signal Conversion

## Purpose

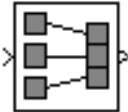
Convert a signal to a new type without altering signal values.

## Library

Signal Attributes

## Description

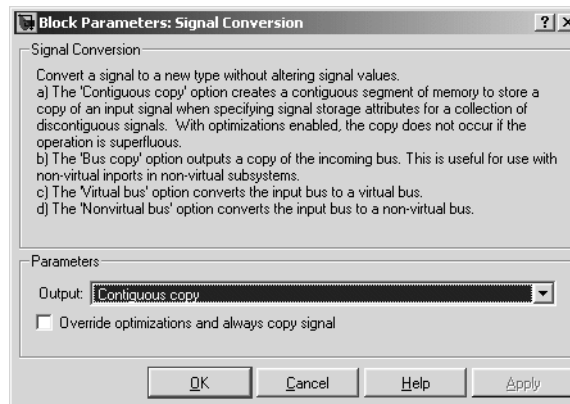
The Signal Conversion block converts a signal from one type to another. The block's **Output** parameter lets you select the type of conversion to be performed.



## 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 be performed. The options are:

- **Contiguous copy**  
Converts a muxed signal whose elements occupy discontinuous 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 discontinuous areas (represented by the block's input) to the contiguous areas (represented by the block's output) at each time step.

- **Bus copy**  
Outputs a copy of the bus connected to the block’s input.
- **Virtual bus**  
Converts a nonvirtual bus to a virtual bus. This option enables you to combine an originally nonvirtual bus with a virtual bus.
- **Nonvirtual bus**  
Converts a virtual bus that was originally a nonvirtual bus back to a nonvirtual bus.

**Override optimizations and always copy signal**

This option is enabled only for Contiguous copy conversion. Unless you select this option, Simulink eliminates the block from the compiled model as an optimization, if the elements of the input signal occupy contiguous areas of memory.

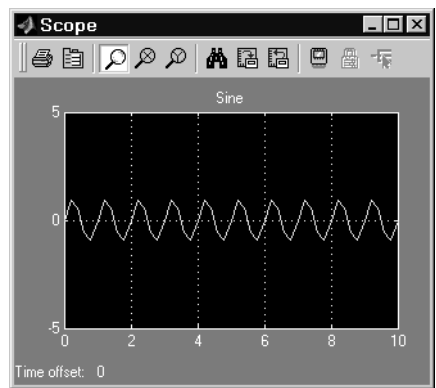
<b>Characteristics</b>	Sample Time	Inherited
	Scalar Expansion	n/a
	Dimensionalized	n/a
	Zero Crossing	No

# Signal Generator

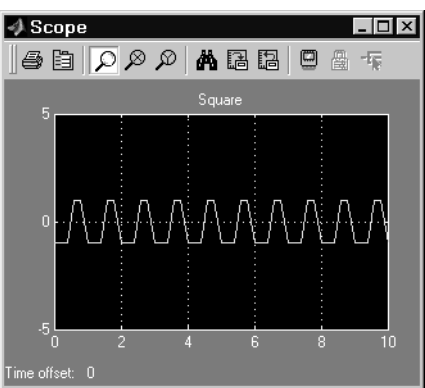
**Purpose** Generate various waveforms

**Library** Sources

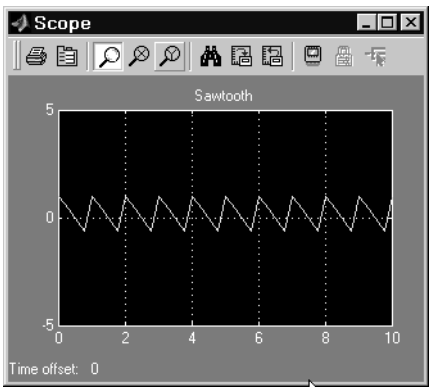
**Description** The Signal Generator block can produce one of three different waveforms: sine wave, square wave, and sawtooth wave. The signal parameters can be expressed in Hertz (the default) or radians per second. This figure shows each signal displayed on a Scope using default parameter values.



Sine Wave



Square Wave



Sawtooth Wave

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.

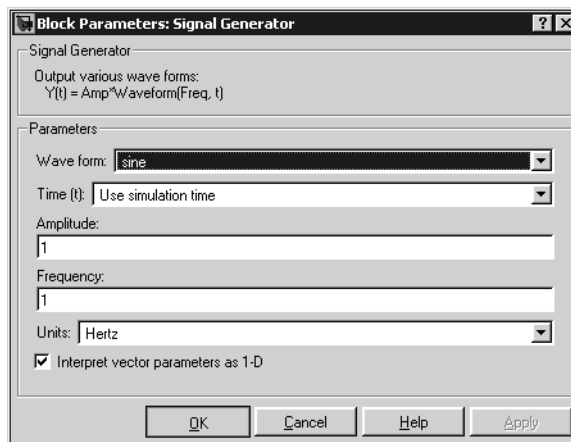
You can vary the output settings of the Signal Generator block while a simulation is in progress. This is useful to determine quickly the response of a system to different types of inputs.

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



# Signal Generator

---

**Wave form**

The wave form: a sine wave, square wave, or sawtooth wave. The default is a sine wave. This parameter cannot be changed while a simulation is running.

**Time**

Specifies whether to use simulation time as the source of values for the waveform’s time variable or an external signal. If you specify an external time source, the block displays an input port for the time source.

**Amplitude**

The signal amplitude. The default is 1.

**Frequency**

The signal frequency. The default is 1.

**Units**

The signal units: Hertz or radians/sec. The default is Hertz.

**Interpret vector parameters as 1-D**

If selected, column or row matrix values for the **Amplitude** and **Frequency** parameters result in a vector output signal.

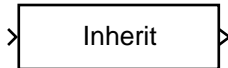
**Characteristics**

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

**Purpose** Specify the desired dimensions, sample time, data type, numeric type, and other attributes of a signal.

**Library** Signal Attributes

**Description** The Signal Specification block allows you to specify the attributes of the signal connected to its input and output ports. If the specified attributes conflict with the attributes specified by the blocks connected to its ports, Simulink displays an error when it compiles the model, for example, at the beginning of a simulation. If no conflict exists, Simulink eliminates the Signal Specification block from the compiled model. In other words, the Signal Specification block is a virtual block. It exists only to specify the attributes of a signal and plays no role in the simulation of the model.



You can use the Signal Specification block to ensure that the actual attributes of a signal meet desired attributes. For example, suppose that you and a colleague are working on different parts of the same model and you use Signal Specification blocks to connect your part of the model with your colleague's. Now, if your colleague changes the attributes of a signal without informing you, the attributes entering the corresponding Signal Specification block do not match and Simulink reports an appropriate error.

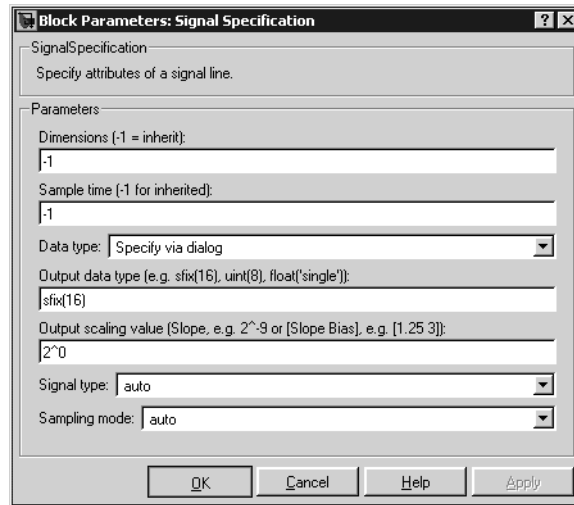
The Signal Specification block can also be used to ensure correct propagation of signal attributes throughout a model. The capability of allowing many attributes to be propagated from block to block in Simulink is very powerful. However, because blocks may not specify some or all of the attributes of the signals they accept or output, it is possible to create models that don't have enough information to correctly propagate attributes around the model. For these cases, the Signal Specification block is a good way of providing the information Simulink needs. Using the Signal Specification block also helps speed up model compilation when blocks are missing signal attributes.

**Data Type Support** The Signal Specification block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. The input data type must match the data type specified by the **Data type** parameter.

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

# Signal Specification

## Parameters and Dialog Box



### Dimensions

Specify the dimension's of the block's input and output signals. Valid values are

- -1—Inherited from the block to which it is connected
- n—Vector signal of width n
- [m n]—Matrix signal having m rows and n columns

### Sample Time

Specify the sample time at which the block is updated. Valid values are

- -1—inherited from the block to which it is connected
- period  $\geq 0$
- [offset, period]
- [0, -1]
- [-1, -1]

where period is the sample rate and offset is the offset of the sample period from time zero (see “Sample Time”).



**Data type**

Specify the data type of the input and output signals. To let Simulink determine the data type, set this parameter to auto.

**Output data type**

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

**Output scaling value**

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

**Signal type**

Specify the numeric type (real or complex) of the input and output signal. To let Simulink determine the numeric type, set this parameter to auto.

**Sampling mode**

Specify the sampling mode (sample-based or frame-based) of this block. To let Simulink determine the sampling mode, set this parameter to auto.

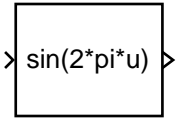
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified by the block's Sample Time parameter.
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Sine

**Purpose** Implement a sine wave in fixed-point using a lookup table approach that exploits quarter wave symmetry

**Library** Lookup Tables

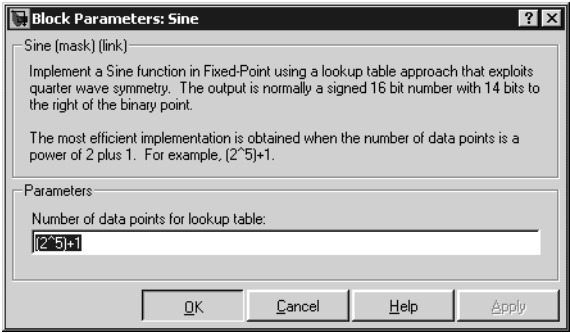
**Description** The Sine block implements a sine wave in fixed-point using a lookup table method that exploits quarter wave symmetry.



You can set the number of data points to retrieve from the lookup table with the **Number of data points for lookup table** parameter.

**Data Type Support** The Sine block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



**Number of data points for lookup table**  
Number of data points to retrieve from the lookup table.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	N/A

**Purpose** Generate a sine wave

**Library** Sources

**Description**



The Sine Wave block provides a sinusoid. The block can operate in either time-based or sample-based mode.

**Time-Based Mode**

The output of the Sine Wave block is determined by

$$y = \textit{Amplitude} \times \sin(\textit{frequency} \times \textit{time} + \textit{phase}) + \textit{bias}$$

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 “Specifying 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:

$$\begin{aligned}\sin(t + \Delta t) &= \sin(t) \cos(\Delta t) + \sin(\Delta t) \cos(t) \\ \cos(t + \Delta t) &= \cos(t) \cos(\Delta t) - \sin(t) \sin(\Delta t)\end{aligned}$$

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  $\Delta t$  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 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 increments  $k$  and recomputes the output of the block. When  $k$  reaches  $p$ , Simulink 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

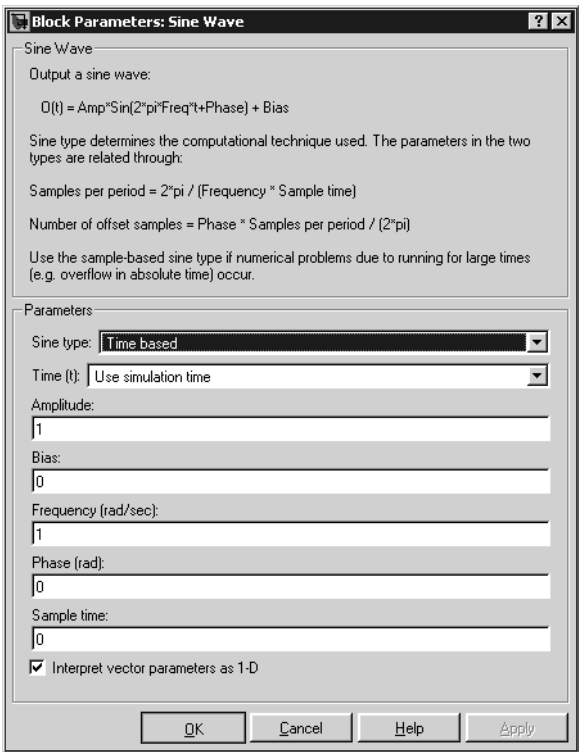
The block's numeric parameters must be of the same dimensions after scalar expansion. If the **Interpret vector parameters as 1-D** option is off, the block outputs a signal of the same dimensions and dimensionality as the parameters. If the **Interpret vector parameters as 1-D** option is on and the 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.

# Sine Wave

## Parameters and Dialog Box



### Sine type

Type of sine wave generated by this block, either time- or sample-based. Some of the other options presented by the Sine Wave dialog box depend on whether you select time-based or sample-based as the value of **Sine type** parameter.

### 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/second. The default is 1 rad/s. This parameter appears only if you choose time-based as the **Sine type** of the block.

**Samples per period**

Number of samples per period. This parameter appears only if you choose sample-based as the **Sine type** of the block.

**Phase**

The phase shift, in radians. The default is 0 radians. This parameter appears only if you choose time-based as the **Sine type** of the block.

**Number of offset samples**

The offset (discrete phase shift) in number of sample times. This parameter appears only if you choose sample-based as the **Sine type** of the block.

**Sample time**

The sample period. The default is 0. If the sine type is sample-based, the sample time must be greater than 0. See “Specifying Sample Time” in the online documentation for more information.

**Interpret vector parameters as 1-D**

If selected, column or row matrix values for the Sine Wave block’s numeric parameters result in a vector output signal; otherwise, the block outputs a signal of the same dimensionality as the parameters. If this option is not selected, the block always outputs a signal of the same dimensionality as the block’s numeric parameters.

**Characteristics**

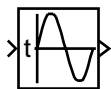
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes
Zero Crossing	No

# Sine Wave Function

## Purpose

**Library**                      Math Operations

## Description



## Data Type Support

## Parameters and Dialog Box

Block Parameters: Sine Wave Function

Sine Wave

Output a sine wave:  
$$Q(t) = \text{Amp} \cdot \sin(2 \cdot \pi \cdot \text{Freq} \cdot t + \text{Phase}) + \text{Bias}$$

Sine type determines the computational technique used. The parameters in the two types are related through:

Samples per period =  $2 \cdot \pi / (\text{Frequency} \cdot \text{Sample time})$

Number of offset samples =  $\text{Phase} \cdot \text{Samples per period} / (2 \cdot \pi)$

Use the sample-based sine type if numerical problems due to running for large times (e.g. overflow in absolute time) occur.

Parameters

Sine type: Time based

Time (t): Use external signal

Amplitude:  
1

Bias:  
0

Frequency (rad/sec):  
1

Phase (rad):  
0

OK

Cancel

Help

Apply

## Characteristics

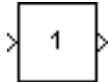
1-456



**Purpose** Vary a scalar gain using a slider

**Library** Math Operations

**Description** The Slider Gain block allows you to vary a scalar gain during a simulation using a slider. The block accepts one input and generates one output.



**Data Type Support** Data type support for the Slider Gain block is the same as that for the Gain block (see Gain on page 1-232).

**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.

# Slider Gain

---

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Yes, of the gain
	States	0
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Implement a linear state-space system

**Library** Continuous

**Description** The State-Space block implements a system whose behavior is defined by

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

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

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

	n	m
n	A	B
r	C	D

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

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

**Specifying the Absolute Tolerance for the Block’s States**

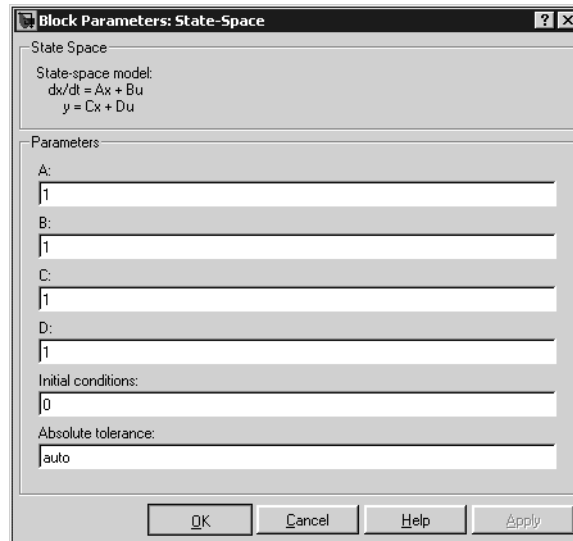
By default Simulink uses the absolute tolerance value specified in the **Configuration Parameters** dialog box (see “Error Tolerances”) to solve the states of the State-Space block. If this value does not provide sufficient error control, specify a more appropriate value in the **Absolute tolerance** field of the State-Space block’s dialog box. The value that you specify is used to solve all the block’s states.

# State-Space

## Data Type Support

A State-Space block accepts and outputs real signals of type double.

## Parameters and Dialog Box



### A, B, C, D

The matrix coefficients.

### Initial conditions

The initial state vector.

### Absolute tolerance

Absolute tolerance used to solve the block's states. You can enter auto or a numeric value. If you enter auto, Simulink determines the absolute tolerance (see "Error Tolerances"). If you enter a numeric value, Simulink uses the specified value to solve the block's states. Note that a numeric value overrides the setting for the absolute tolerance in the **Configuration Parameters** dialog box.

Characteristics	Direct Feedthrough	Only if $D \neq 0$
	Sample Time	Continuous
	Scalar Expansion	Yes, of the initial conditions
	States	Depends on the size of A
	Dimensionalized	Yes
	Zero Crossing	No

# Step

**Purpose** Generate a step function

**Library** Sources

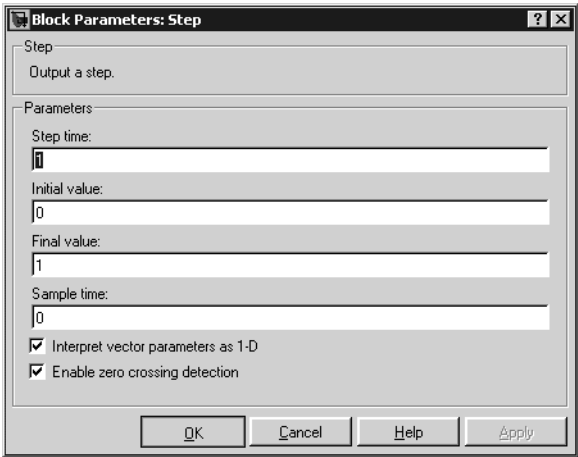
**Description** The Step block provides a step between two definable levels at a specified time. If the simulation time is less than the **Step time** parameter value, the block's output is the **Initial value** parameter value. For simulation time greater than or equal to the **Step time**, the output is the **Final value** parameter value.



The block's numeric parameters must be of the same dimensions after scalar expansion. If the **Interpret vector parameters as 1-D** option is off, the block outputs a signal of the same dimensions and dimensionality as the parameters. If the **Interpret vector parameters as 1-D** option is on and the numeric parameters are row or column vectors (i.e., single row or column 2-D arrays), the block outputs a vector (1-D array) signal; otherwise, the block outputs a signal of the same dimensionality and dimensions as the parameters.

**Data Type Support** The Step block outputs real signals of type double.

## Parameters and Dialog Box

The dialog box for the Step block parameters. It has a title bar "Block Parameters: Step" with a question mark and close button. The main area is divided into sections. The first section is labeled "Step" and contains the text "Output a step." The second section is labeled "Parameters" and contains several input fields: "Step time:" with a value of 1, "Initial value:" with a value of 0, "Final value:" with a value of 1, and "Sample time:" with a value of 0. Below these fields are two checked checkboxes: "Interpret vector parameters as 1-D" and "Enable zero crossing detection". At the bottom are four buttons: "OK", "Cancel", "Help", and "Apply".

**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 “Specifying Sample Time” in the online documentation for more information.

**Interpret vector parameters as 1-D**

If selected, column or row matrix values for the Step block’s numeric parameters result in a vector output signal; otherwise, the block outputs a signal of the same dimensionality as the parameters. If this option is not selected, the block always outputs a signal of the same dimensionality as the block’s numeric parameters.

**Enable zero crossing detection**

Select to enable zero crossing detection to detect step times. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

**Characteristics**

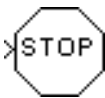
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes, of parameters
Dimensionalized	Yes

# Stop Simulation

**Purpose** Stop the simulation when the input is nonzero

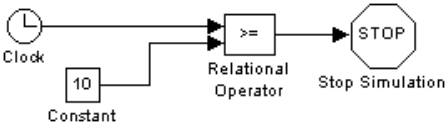
**Library** Sinks

**Description** The Stop Simulation block stops the simulation when the input is nonzero.



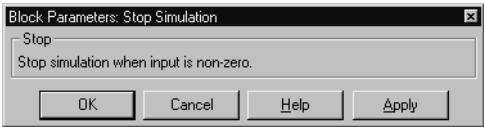
The simulation completes the current time step before terminating. If the block input is a vector, any nonzero vector element causes the simulation to stop.

You can use this block in conjunction with the Relational Operator block to control when the simulation stops. For example, this model stops the simulation when the input signal reaches 10.



**Data Type Support** The Stop Simulation block accepts real signals of type double or Boolean.

**Parameters and Dialog Box**



<b>Characteristics</b>	Sample Time	Inherited from driving block
	Dimensionalized	Yes



# Subsystem, Atomic Subsystem, CodeReuse Subsystem

**Purpose** Represent a 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 true (atomic) subsystem, depending on the value of its **Treat as Atomic Unit** parameter. An Atomic Subsystem block is a Subsystem block that has its **Treat as Atomic Unit** parameter selected by default.



You 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 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.

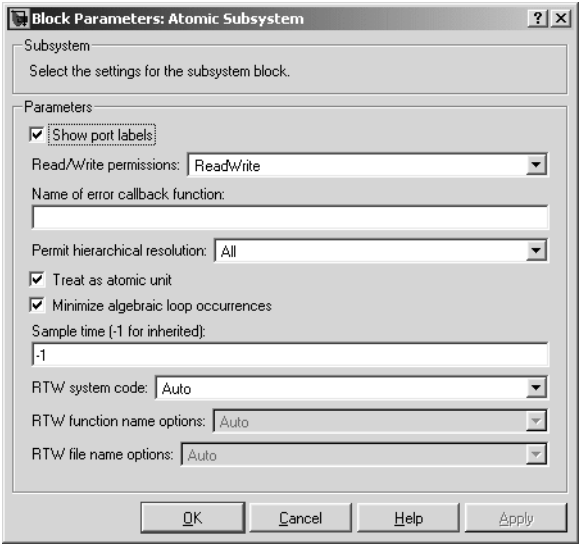
See “Creating Subsystems” for more information about subsystems.

**Data Type Support** A subsystem's enable and trigger ports accept any data type supported by Simulink, including fixed-point data types. For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.

See Inport on page 1-255 for information on the data types accepted by a subsystem's input ports. See Outport on page 1-347 for information on the data types output by a subsystem's output ports.

# Subsystem, Atomic Subsystem, CodeReuse Subsystem

## Parameters and Dialog Box



### Show port labels

Causes Simulink to display the labels of the subsystem’s ports in the subsystem’s icon.

### Read/Write permissions

Controls user access to the contents of the subsystem. You can select any of the following values.

Permissions	Description
ReadWrite	User can open and modify the contents of the subsystem.

Permissions	Description
ReadOnly	User can open but not modify the subsystem. If the subsystem resides in a block library, a user 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	User cannot open or modify the subsystem. If the subsystem resides in a library, a user can create links to the subsystem in a model but cannot open, modify, change permissions, or create local copies of the subsystem.

## Name of error callback function

Name of a function to be called if an error occurs while Simulink is executing the subsystem. Simulink 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 displays a generic error message if executing the subsystem causes an error.

## Permit hierarchical resolution

Specifies whether to resolve names of workspace variables referenced by this subsystem. The options are

- 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)

- ParametersOnly

Resolve only names of workspace variables used to specify block parameter values.

- None

Do not resolve any workspace variable names.

# Subsystem, Atomic Subsystem, CodeReuse Subsystem

---

## **Treat as atomic unit**

Causes Simulink to treat the subsystem as a unit when determining block execution order. When it comes time to execute the subsystem, Simulink executes all blocks within the subsystem before executing any other block at the same level as the subsystem block. If this option is not selected, Simulink treats all blocks in the subsystem as being at the same level in the model hierarchy as the subsystem when determining block execution order. This can cause execution of blocks within the subsystem to be interleaved with execution of blocks outside the subsystem. See “Atomic Versus Virtual Subsystems” for more information.

## **Minimize algebraic loop occurrences**

Try to eliminate any algebraic loops that include this subsystem (see “Eliminating Algebraic Loops” in the online Simulink documentation for more information).

## **Propagate execution context across subsystem boundary**

This option appears only if the subsystem is conditionally executed. If selected (the default), this option enables execution context propagation across this subsystem’s boundary (see “Propagating Execution Contexts” in the online Simulink documentation).

## **Sample time**

Specifies the sample time of this subsystem.

---

**Note** Parameters whose names begin with RTW are used by Real-Time Workshop for code generation. See the Real-Time Workshop documentation for more information.

---

# Subsystem, Atomic Subsystem, CodeReuse Subsystem

---

<b>Characteristics</b>	Sample Time	Depends on the blocks in the subsystem
	Dimensionalized	Depends on the blocks in the subsystem
	Zero Crossing	Yes, for enable and trigger ports if present

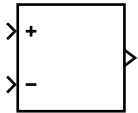
# Subtract

---

**Purpose** Add or subtract inputs

**Library** Math Operations

**Description** The Subtract block is an implementation of the Sum block. See Sum on page 1-471 for more information.



**Purpose** Add or subtract inputs

**Library** Math Operations

**Description**



The Sum block performs addition or subtraction on its inputs. This block can add or subtract scalar, vector, or matrix inputs. It can also collapse the elements of a single input vector.

You specify the operations of the block with the **List of Signs** parameter. Plus (+), minus (-), and spacer (|) characters indicate the operations to be performed on the inputs:

- If there are two or more inputs, then the number of characters must equal the number of inputs. For example, “+ - +” requires three inputs and configures the block to subtract the second (middle) input from the first (top) input, and then add the third (bottom) input.

All nonscalar inputs must have the same dimensions. Scalar inputs will be expanded to have the same dimensions as the other inputs.

- A spacer character creates extra space between ports on the block’s icon.
- If only addition of all inputs is required, then a numeric parameter value equal to the number of inputs can be supplied instead of “+” characters.
- If only one vector is input, then a single “+” or “-” will collapse the vector using the specified operation.

The Sum block first converts the input data type(s) to the output data type using the specified rounding and overflow modes, and then performs the specified operations.

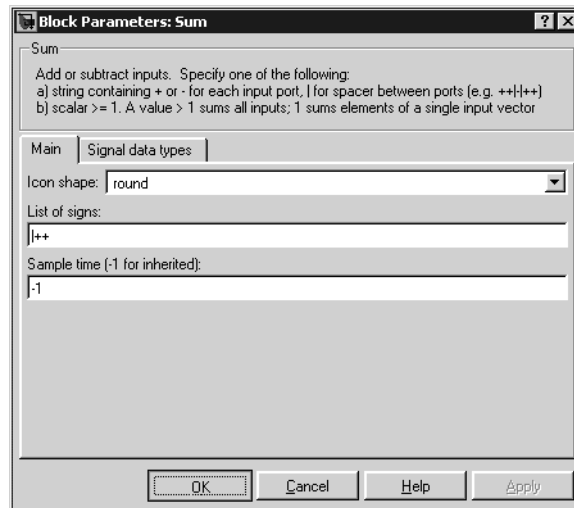
**Data Type Support**

The Sum block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. The inputs may be of different data types unless you select the **Require all inputs to have same data type** parameter.

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

**Parameters and Dialog Box**

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



## Icon shape

Designate the icon shape of the block.

## List of signs

Enter as many plus (+) and minus (-) characters as there are inputs. Addition is the default operation, so if you only want to add the inputs, enter the number of input ports. For a single vector input, “+” or “-” will collapse the vector using the specified operation.

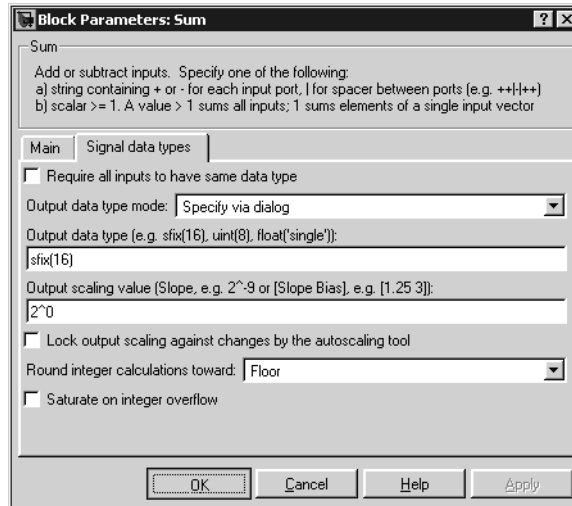
You can manipulate the positions of the input ports on the block by inserting spacers (|) between the signs in the **List of signs** parameter. For example, “++| -” creates an extra space between the second and third input ports.

## Sample time (-1 for inherited)

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

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





### Require all inputs to have same data type

Select this parameter to require that all inputs must have the same data type.

### Output data type mode

Specify the output data type and scaling to be the same as the first input, or inherit the data type and scaling from an internal rule or by backpropagation. You can also choose a built-in data type from the drop-down list. Lastly, if you choose **Specify via dialog**, the **Output data type**, **Output scaling value**, and **Lock output scaling against changes by the autoscaling tool** parameters become visible.

### Output data type

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

### Output scaling value

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

**Lock output scaling against changes by the autoscaling tool**

Select to lock scaling of outputs. This parameter is only visible if you select Specify via dialog for the **Output data type mode** parameter.

**Round integer calculations toward**

Select the rounding mode for fixed-point operations.

**Saturate on integer overflow**

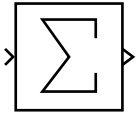
Select to have overflows saturate.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes
	States	0
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Add or subtract inputs

**Library** Math Operations

**Description** The Sum of Elements block is an implementation of the Sum block. See Sum on page 1-471 for more information.

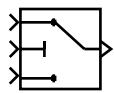


# Switch

**Purpose** Switch output between the first input and the third input based on the value of the second input

**Library** Signal Routing

**Description** The Switch block passes through the first (top) input or the third (bottom) input based on the value of the second (middle) input. The first and third inputs are called *data inputs*. The second input is called the *control input*.

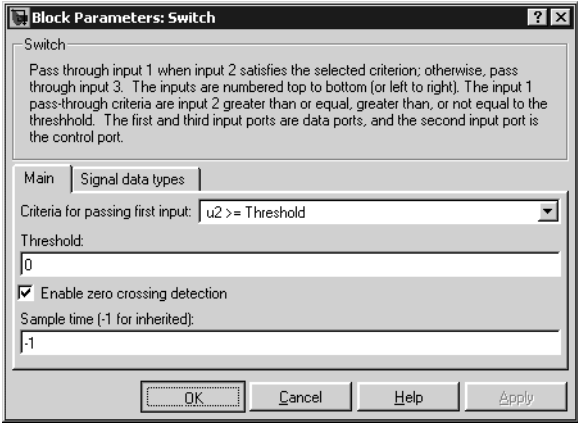


You select the conditions under which the first input is passed with the **Criteria for passing first input** parameter. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in the **Criteria for passing first input parameter**, then the first input is passed. Otherwise, the third input is passed.

**Data Type Support** The data and control inputs of a Switch block accept real or complex signals of any data type supported by Simulink, including fixed-point data types.

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

**Parameters and Dialog Box** The **Main** pane of the Switch block dialog appears as follows:



## Criteria for passing first input

Select the conditions under which the first input is passed. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in this parameter, then the first input is passed. Otherwise, the third input is passed.

## Threshold

Assign the switch threshold that determines which input is passed to the output.

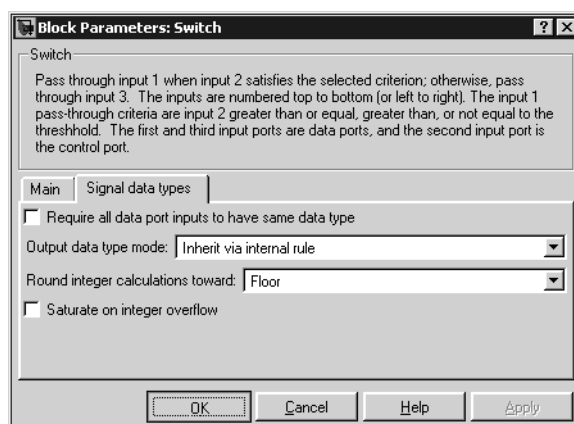
## Enable zero crossing detection

Select to enable zero crossing detection. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

## Sample time (-1 for inherited)

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

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



## Require all data port inputs to have same data type

Select to require all data inputs to have the same data type.

# Switch

---

## Output data type mode

Choose to inherit the output data type and scaling by backpropagation or by an internal rule. The internal rule causes the output of the block to have the same data type and scaling as the input with the larger positive range.

## Round integer calculations toward

Select the rounding mode for fixed-point operations.

## Saturate on integer overflow

Select to have overflows saturate.

## Characteristics

Direct Feedthrough	Yes
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	No, unless you select <b>Enable zero crossing detection</b> .

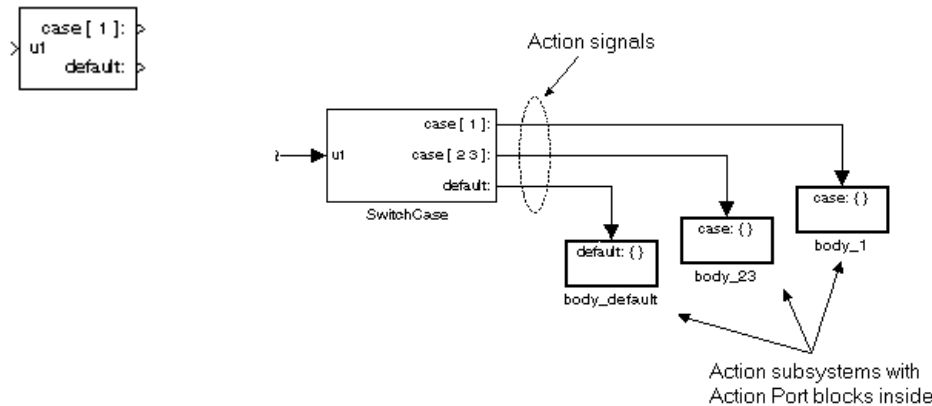
## See Also

Multipoint Switch

**Purpose** Implement a C-like switch control flow statement

**Library** Ports & Subsystems

**Description** The following shows a completed Simulink C-like switch control flow statement in the subsystem of the Switch Case block.



A Switch Case block receives a single input, which it uses to form case conditions that determine which the subsystem to execute. Each output port case condition is attached to a Switch Case Action subsystem. The cases are evaluated top down starting with the top case. If a case value (in brackets) corresponds to the actual value of the input, its Switch Case Action subsystem is executed.

The preceding switch control flow statement can be represented by the following pseudocode:

```
switch (u1) {
    case [u1=1]:
        body_1;
        break;
    case [u1=2 or u1=3]:
        body_23;
        break;
    default:
        bodydefault;
}
```

# Switch Case

---

You construct a Simulink switch control flow statement like the example shown as follows:

- 1 Place a Switch Case block in the current system and attach the input port labeled u1 to the source of the data you are evaluating.
- 2 Open the **Block Parameters** dialog of the Switch Case block and enter as follows:

- a Enter the **Case conditions** field with the individual cases.

Each case can be an integer or set of integers specified with MATLAB cell notation. See the **Case conditions** field in the “Parameters and Dialog Box” section of this reference.

- b Select the **Show default case** check box to show a default case output port on the Switch Case block.

If all other cases are false, the default case is taken.

- 3 Create a Switch Case Action subsystem for each case port you added to the Switch Case block.

These consist of subsystems with Action Port blocks inside them. When you place the Action Port block inside a subsystem, the subsystem becomes an atomic subsystem with an input port labeled Action.

- 4 Connect each case output port and the default output port of the Switch Case block to the Action port of an Action subsystem.

Each connected subsystem becomes a case body. This is indicated by the change in label for the Switch Case Action subsystem block and the Action Port block inside of it to the name case{ }.

During simulation of a switch control flow statement, the Action signals from the Switch Case block to each Switch Case Action subsystem turn from solid to dashed.

- 5 In each Switch Case Action subsystem, enter the Simulink logic appropriate to the case it handles.



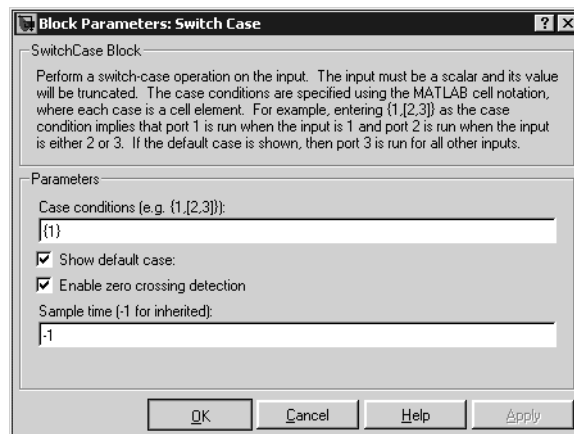
**Note** As demonstrated in the preceding pseudocode example, cases for the Switch Case block contain an implied break after their Switch Case Action subsystems are executed. There is no fall-through behavior for the Simulink switch control flow statement as found in standard C switch statements.

## Data Type Support

Input to the port labeled u1 of a Switch Case block can be a scalar value of any data type supported by Simulink except Boolean. The input to u1 can also be a fixed-point data type. Noninteger inputs are truncated. For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using Simulink documentation.

Data outputs are action signals to Switch Case Action subsystems that are created with Action Port blocks and subsystems.

## Parameters and Dialog Box



## Case conditions

Case conditions are specified using MATLAB cell notation where each cell is a case condition consisting of integers or arrays of integers. In the preceding dialog example, entering `{1,[7,9,4]}` specifies that output port case[1] is run when the input value is 1, and output port case[7 9 4] is run when the input value is 7, 9, or 4.

# Switch Case

---

You can use colon notation to specify a range of case conditions. For example, entering `{[1:5]}` specifies that output port `case[1 2 3 4 5]` is run when the input value is 1, 2, 3, 4, or 5.

Depending on block size, cases with long lists of conditions are displayed in shortened form in the Switch Case block, using a terminating ellipsis (...).

## Show default case

If you select this check box, the default output port appears as the last case on the Switch Case block. This case is run when the input value does not match any of the case values specified in the **Case conditions** field.

## Enable zero crossing detection

Select to enable use of zero crossing detection. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

## Sample time (-1 for inherited)

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

## Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	No
Zero Crossing	Yes, if zero-crossing detection is enabled.

**Purpose**

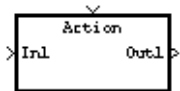
Represent a subsystem whose execution is triggered by a Switch Case block

**Library**

Ports & Subsystems

**Description**

This block is a Subsystem block that is preconfigured to serve as a starting point for creating a subsystem whose execution is triggered by a Switch Case block. See the Switch Case block and “Control Flow Blocks” for more information.



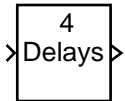
# Tapped Delay

---

**Purpose** Delay a scalar signal multiple sample periods and output all the delayed versions

**Library** Discrete

## Description



The Tapped Delay block delays its input by the specified number of sample periods, and outputs all the delayed versions.

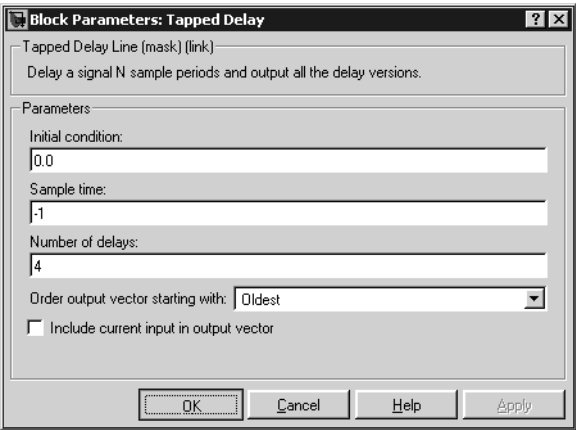
This block provides a mechanism for discretizing a signal in time, or resampling the signal at a different rate. You specify the time between samples with the **Sample time** parameter. You specify the number of delays with the **Number of delays** parameter. A value of -1 instructs the block to inherit the number of delays by backpropagation. Each delay is equivalent to the  $z^{-1}$  discrete-time operator, which is represented by the Unit Delay block.

The block accepts one scalar input and generates an output for each delay. The input must be a scalar. You specify the order of the output vector with the **Order output vector starting with** parameter list. **Oldest** orders the output vector starting with the oldest delay version and ending with the newest delay version. **Newest** orders the output vector starting with the newest delay version and ending with the oldest delay version.

The block output for the first sampling period is specified by the **Initial condition** parameter. Careful selection of this parameter can minimize unwanted output behavior.

**Data Type Support** The Tapped Delay block accepts signals of any data type supported by Simulink, including fixed-point data types.

Parameters and Dialog Box



Initial condition

Specify the initial output of the simulation. The **Initial condition** parameter is converted from a double to the input data type offline using round-to-nearest and saturation.

Sample time

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

Number of delays

Specify the number of discrete-time operators.

Order output vector starting with

Specify whether the oldest delay version is output first, or the newest delay version is output first.

Include current input in output vector

Select to include the current input in the output vector.

Characteristics

Direct Feedthrough	Yes, when <b>Include current input in output vector</b> parameter is checked. No otherwise.
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes, of initial conditions

# Terminator

---

**Purpose**                      Terminate an unconnected output port

**Library**                     Sinks

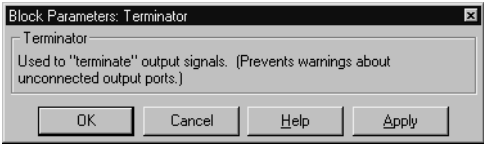
**Description**                The Terminator block can be used to cap blocks whose output ports are not connected to other blocks. If you run a simulation with blocks having unconnected output ports, Simulink issues warning messages. Using Terminator blocks to cap those blocks avoids warning messages.



**Data Type Support**                The Terminator block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types.

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

**Parameters and Dialog Box**



<b>Characteristics</b>	Sample Time	Inherited from driving block
	Dimensionalized	Yes

## Purpose

Generate linear models in the base workspace at specific times

## Library

Model-Wide Utilities

## Description

T=1

This block calls `linmod` or `dlinmod` to create a linear model for the system when the simulation clock reaches the time specified by the **Linearization time** parameter. No trimming is performed. The linear model is stored in the base workspace as a structure, along with information about the operating point at which the snapshot was taken. Multiple snapshots are appended to form an array of structures.

The name of the structure used to save the snapshots is the name of the model appended by `_Timed_Based_Linearization`, for example, `vdp_Timed_Based_Linearization`. The structure has the follow fields:

Field	Description
a	The A matrix of the linearization
b	The B matrix of the linearization
c	The C matrix of the linearization
d	The D matrix of the linearization
StateName	Names of the model's states
OutputName	Names of the model's output ports
InputName	Names of the model's input ports
OperPoint	A structure that specifies the operating point of the linearization. The structure specifies the operating point time ( <code>OperPoint.t</code> ). The states ( <code>OperPoint.x</code> ) and inputs ( <code>OperPoint.u</code> ) 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.

# Time-Based Linearization

---

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.

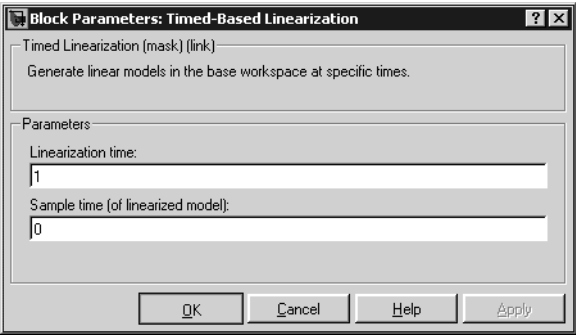
```
x1 = xout(ind1,:);  
x2 = xout(ind2,:);
```

## 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 2-4).

### Characteristics

Sample Time	Specified in the <b>Sample time</b> parameter
Dimensionalized	No

# To File

## Purpose

Write data to a file

## Library

Sinks

## Description



The To File block writes its input to a matrix in a MAT-file. The block writes one column for each time step: the first row is the simulation time; the remainder of the column is the input data, one data point for each element in the input vector. The matrix has this form.

$$\begin{bmatrix} t_1 & t_2 & \dots & t_{final} \\ u1_1 & u1_2 & \dots & u1_{final} \\ \dots & \dots & \dots & \dots \\ un_1 & un_2 & \dots & un_{final} \end{bmatrix}$$

The From File block can use data written by a To File block without any modifications. However, the form of the matrix expected by the From Workspace block is the transposition of the data written by the To File block.

The block writes the data as well as the simulation time after the simulation is completed. Its icon shows the name of the specified output file.

The amount of data written and the time steps at which the data is written are determined by block parameters:

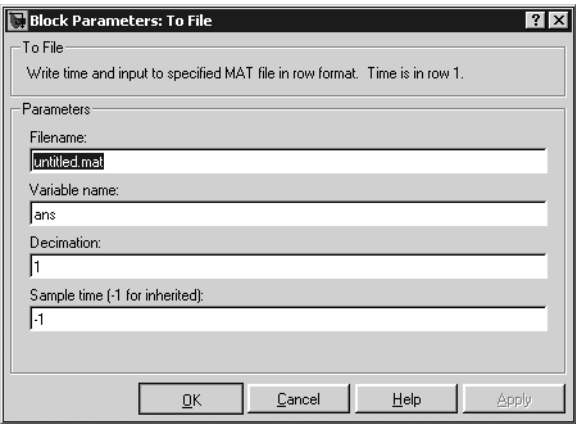
- The **Decimation** parameter allows you to write data at every  $n$ th sample, where  $n$  is the decimation factor. The default decimation, 1, writes data at every time step.
- The **Sample time** parameter allows you to specify a sampling interval at which to collect points. This parameter is useful when you are using a variable-step solver where the interval between time steps might not be the same. The default value of -1 causes the block to inherit the sample time from the driving block when determining the points to write. See “Specifying Sample Time” in the online documentation for more information.

If the file exists at the time the simulation starts, the block overwrites its contents.

## Data Type Support

The To File block accepts real signals of type double.

## Parameters and Dialog Box



### Filename

The fully qualified pathname or filename of the MAT-file in which to store the output. On UNIX, the pathname may start with a tilde (~) character signifying your home directory. The default filename is `untitled.mat`. If you specify an unqualified filename, Simulink stores the file in the MATLAB working directory. (To determine the working directory, type `pwd` at the MATLAB command line.)

### Variable name

The name of the matrix contained in the named file.

### Decimation

A decimation factor. The default value is 1.

### Sample time

The sample period and offset at which to collect points. See “Specifying Sample Time” in the online documentation for more information.

### Characteristics

Sample Time	Specified in the <b>Sample time</b> parameter
Dimensionalized	Yes

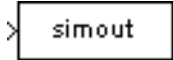
# To Workspace

---

**Purpose** Write data to the workspace

**Library** Sinks

**Description** The To Workspace block writes its input to the workspace. The block writes its output to an array or structure that has the name specified by the block's **Variable name** parameter. The **Save format** parameter determines the output format.



## Array

Selecting this option causes the To Workspace block to save the input as an N-dimensional array where N is one more than the number of dimensions of the input signal. For example, if the input signal is a 1-D array (i.e., a vector), the resulting workspace array is two-dimensional. If the input signal is a 2-D array (i.e., a matrix), the array is three-dimensional.

The way samples are stored in the array depends on whether the input signal is a scalar or vector or a matrix. If the input is a scalar or a vector, each input sample is output as a row of the array. For example, suppose that the name of the output array is `simout`. Then, `simout(1, :)` corresponds to the first sample, `simout(2, :)` corresponds to the second sample, etc. If the input signal is a matrix, the third dimension of the workspace array corresponds to the values of the input signal at specified sampling point. For example, suppose again that `simout` is the name of the resulting workspace array. Then, `simout(:, :, 1)` is the value of the input signal at the first sample point; `simout(:, :, 2)` is the value of the input signal at the second sample point; etc.

The amount of data written and the time steps at which the data is written are determined by block parameters:

- The **Limit data points to last** parameter indicates how many sample points to save. If the simulation generates more data points than the specified maximum, the simulation saves only the most recently generated samples. To capture all the data, set this value to `inf`.
- The **Decimation** parameter allows you to write data at every *n*th sample, where *n* is the decimation factor. The default decimation, 1, writes data at every time step.

- The **Sample time** parameter allows you to specify a sampling interval at which to collect points. This parameter is useful when you are using a variable-step solver where the interval between time steps might not be the same. The default value of -1 causes the block to inherit the sample time from the driving block when determining the points to write. See “Specifying Sample Time” in the online documentation for more information.

During the simulation, the block writes data to an internal buffer. When the simulation is completed or paused, that data is written to the workspace. Its icon shows the name of the array to which the data is written.

## Structure

This format consists of a structure with three fields: time, signals, and blockName. The time field is empty. The blockName field contains the name of the To Workspace block. The signals field contains a structure with three fields: values, dimensions, and label. The values field contains the array of signal values. The dimensions field specifies the dimensions of the values array. The label field contains the label of the input line.

## Structure with Time

This format is the same as Structure except that the time field contains a vector of simulation time steps.

## Using Saved Data with a From Workspace Block

If the data written using a To Workspace block is intended to be played back in another simulation using a From Workspace block, use the Structure with Time format to save the data.

## Examples

In a simulation where the start time is 0, the **Maximum number of sample points** is 100, the **Decimation** is 1, and the **Sample time** is 0.5. The To Workspace block collects a maximum of 100 points, at time values of 0, 0.5, 1.0, 1.5, ..., seconds. Specifying a **Decimation** value of 1 directs the block to write data at each step.

# To Workspace

In a similar example, the **Maximum number of sample points** is 100 and the **Sample time** is 0.5, but the **Decimation** is 5. In this example, the block collects up to 100 points, at time values of 0, 2.5, 5.0, 7.5, ..., seconds. Specifying a **Decimation** value of 5 directs the block to write data at every fifth sample. The sample time ensures that data is written at these points.

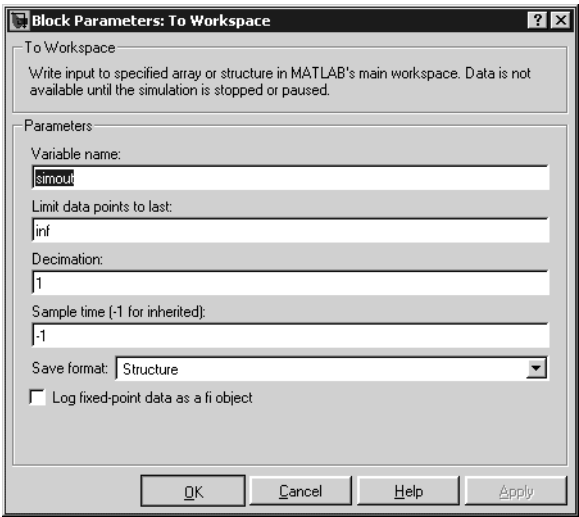
In another example, all parameters are as defined in the first example except that the **Limit data points to last** is 3. In this case, only the last three sample points collected are written to the workspace. If the simulation stop time is 100, data corresponds to times 99.0, 99.5, and 100.0 seconds (three points).

## Data Type Support

The To Workspace block can save real or complex inputs of any data type supported by Simulink, including fixed-point data types, to the MATLAB workspace.

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

## Parameters and Dialog Box



### Variable name

The name of the array that holds the data.

**Limit data points to last**

The maximum number of input samples to be saved. The default is 1000 samples.

**Decimation**

A decimation factor. The default is 1.

**Sample time**

The sample time at which to collect points. See “Specifying Sample Time” in the online documentation for more information.

**Save format**

Format in which to save simulation output to the workspace. The default is structure.

**Log fixed-point data as a fi object**

Select to log fixed-point data to the MATLAB workspace as a Simulink Fixed-Point `fi` object. Otherwise, fixed-point data is logged to the workspace as double.

**Characteristics**

Sample Time	Specified in the <b>Sample time</b> parameter
Dimensionalized	Yes

# Transfer Fcn

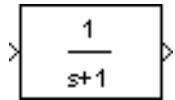
## Purpose

Implement a linear transfer function

## Library

Continuous

## Description



The Transfer Fcn block implements a transfer function where the input ( $u$ ) and output ( $y$ ) can be expressed in transfer function form as the following equation

$$H(s) = \frac{y(s)}{u(s)} = \frac{num(s)}{den(s)} = \frac{num(1)s^{nn-1} + num(2)s^{nn-2} + \dots + num(nn)}{den(1)s^{nd-1} + den(2)s^{nd-2} + \dots + den(nd)}$$

where  $nn$  and  $nd$  are the number of numerator and denominator coefficients, respectively.  $num$  and  $den$  contain the coefficients of the numerator and denominator in descending powers of  $s$ .  $num$  can be a vector or matrix,  $den$  must be a vector, and both are specified as parameters on the block dialog box. The order of the denominator must be greater than or equal to the order of the numerator.

A Transfer Fcn block takes a scalar input. If the numerator of the block's transfer function is a vector, the block's output is also scalar. However, if the numerator is a matrix, the transfer function expands the input into an output vector equal in width to the number of rows in the numerator. For example, a two-row numerator results in a block with scalar input and vector output. The width of the output vector is two.

Initial conditions are preset to zero. If you need to specify initial conditions, convert to state-space form using `tf2ss` and use the State-Space block. The `tf2ss` utility provides the A, B, C, and D matrices for the system. For more information, type `help tf2ss` or consult the Control System Toolbox documentation.

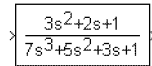
## Transfer Fcn Display

The numerator and denominator are displayed on the Transfer Fcn block depending on how they are specified:

- If each is specified as an expression, a vector, or a variable enclosed in parentheses, the icon shows the transfer function with the specified coefficients and powers of  $s$ . If you specify a variable in parentheses, the

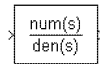


variable is evaluated. For example, if you specify **Numerator** as [3,2,1] and **Denominator** as (den) where den is [7,5,3,1], the block looks like this:



$$\frac{3s^2+2s+1}{7s^3+5s^2+3s+1}$$

- If each is specified as a variable, the block shows the variable name followed by (s). For example, if you specify **Numerator** as num and **Denominator** as den, the block looks like this:



$$\frac{\text{num}(s)}{\text{den}(s)}$$

## Specifying the Absolute Tolerance for the Block's States

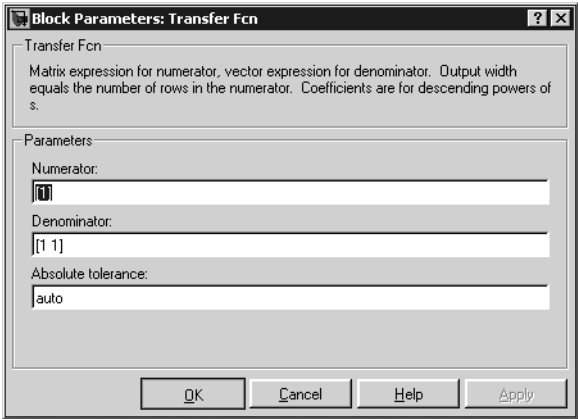
By default Simulink uses the absolute tolerance value specified in the **Configuration Parameters** dialog box (see “Error Tolerances”) to solve the states of the Transfer Fcn block. If this value does not provide sufficient error control, specify a more appropriate value in the **Absolute tolerance** field of the Transfer Fcn block's dialog box. The value that you specify is used to solve all the block's states.

## Data Type Support

The Transfer Fcn block accepts and outputs signals of type double.

# Transfer Fcn

## Parameters and Dialog Box



## Numerator

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

## Denominator

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

## Absolute tolerance

Absolute tolerance used to solve the block's states. You can enter auto or a numeric value. If you enter auto, Simulink determines the absolute tolerance (see "Error Tolerances"). If you enter a numeric value, Simulink uses the specified value to solve the block's states. Note that a numeric value overrides the setting for the absolute tolerance in the **Configuration Parameters** dialog box.

<b>Characteristics</b>	Direct Feedthrough	Only if the lengths of the <b>Numerator</b> and <b>Denominator</b> parameters are equal
	Sample Time	Continuous
	Scalar Expansion	No
	States	Length of <b>Denominator</b> -1
	Dimensionalized	Yes, in the sense that the block expands scalar input into vector output when the transfer function numerator is a matrix. See the preceding block description.
	Zero Crossing	No

<b>Purpose</b>	Implement a Direct Form II realization of a transfer function
----------------	---



**Library** Additional Math & Discrete / Additional Discrete

**Description** The Transfer Fcn Direct Form II block implements a Direct Form II realization of the transfer function specified by the **Numerator coefficients** and the **Denominator coefficients excluding lead** parameters. The block only supports single input-single output transfer functions.

The block automatically selects the data types and scalings of the output, the coefficients, and any temporary variables.

**Data Type Support** The Transfer Fcn Direct Form II block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box


**Block Parameters: Transfer Fcn Direct Form II**


Transfer Fcn Direct Form II (mask) (link)

A Direct Form II realization of the specified transfer function is used. Only single input multiple output transfer functions are supported.

The data types and scalings of the output, the coefficients, and any temporary variables are automatically selected. The automatic choices will be acceptable in many situations. In situations where the automatic choices give unacceptable results, manual layout of the filter is necessary. For manual layout, it is suggested that the blocks under this mask be used as a starting point.

Note 1:  
The full denominator should have a leading coefficient of +1.0, but this leading coefficient should be excluded when entering the parameter. For example, if the denominator is

$$den = 1 \quad -1.7 \quad 0.72$$

just enter

$$den(2:end) = -1.7 \quad 0.72$$

Note 2:  
The numerator does NOT need to be the same size as the full denominator.

Parameters

Numerator coefficients:

Denominator coefficients excluding lead (which must be 1.0):

Initial condition:

Round toward:

☐ Saturate to max or min when overflows occur

**Numerator coefficients**

Specify the numerator coefficients.

**Denominator coefficients excluding lead**

Specify the denominator coefficients, excluding the leading coefficient, which must be 1.0.

**Initial condition**

Set the initial condition.

**Round toward**

Rounding mode for the fixed-point output.

**Saturate to max or min when overflows occur**

If selected, fixed-point overflows saturate.

**Characteristics**

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

**See Also**

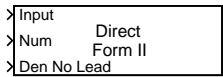
Transfer Fcn Direct Form II Time Varying

# Transfer Fcn Direct Form II Time Varying

**Purpose** Implement a time varying Direct Form II realization of a transfer function

**Library** Additional Math & Discrete / Additional Discrete

**Description** The Transfer Fcn Direct Form II Time Varying block implements a Direct Form II realization of the specified transfer function. The block only supports single input-single output transfer functions.



The block automatically selects the data types and scalings of the output, the coefficients, and any temporary variables.

**Data Type Support** The Transfer Fcn Direct Form II Time Varying block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box

**Block Parameters: Transfer Fcn Direct Form II Time Varying**

Transfer Fcn Direct Form II Time Varying (mask) (link)

A Direct Form II realization of the specified transfer function is used. Only single input single output transfer functions are supported.

The data types and scalings of the output, the coefficients, and any temporary variables are automatically selected. The automatic choices will be acceptable in many situations. In situations where the automatic choices give unacceptable results, manual layout of the filter is necessary. For manual layout, it is suggested that the blocks under this mask be used as a starting point.

**Note 1:**  
The full denominator should have a leading coefficient of +1.0, but this leading coefficient should be excluded when entering the parameter. For example, if the denominator is

den = 1 -1.7 0.72

just enter

den(2:end) = -1.7 0.72

**Note 2:**  
The numerator does NOT need to be the same size as the full denominator.

**Parameters**

Initial condition:

Round toward:

☐ Saturate to max or min when overflows occur

OK Cancel Help Apply

**Initial condition**  
Set the initial condition.

# Transfer Fcn Direct Form II Time Varying

---

**Round toward**

Rounding mode for the fixed-point output.

**Saturate to max or min when overflows occur**

If selected, fixed-point overflows saturate.

**Characteristics**

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

**See Also**

Transfer Fcn Direct Form II

# Transfer Fcn First Order

**Purpose** Implement a discrete-time first order transfer function

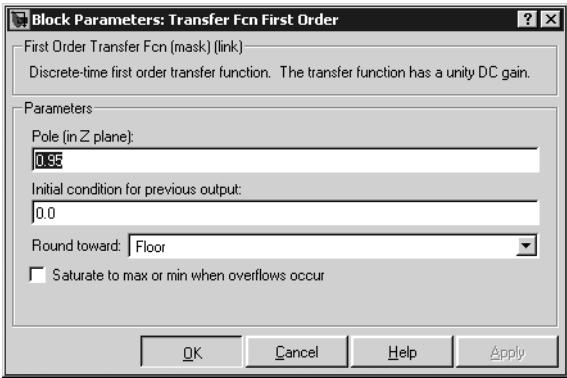
**Library** Discrete

**Description** The Transfer Fcn First Order block implements a discrete-time first order transfer function of the input. The transfer function has a unity DC gain.

$$\frac{0.05z}{z-0.95}$$

**Data Type Support** The Transfer Fcn First Order block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



**Pole (in Z plane)**  
Set the pole.

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

**Round toward**  
Rounding mode for the fixed-point output.

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



<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes, of initial conditions

# Transfer Fcn Lead or Lag

**Purpose** Implement a discrete-time lead or lag compensator

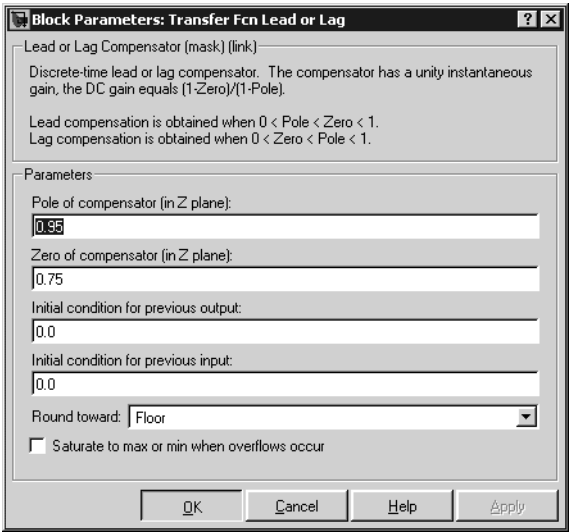
**Library** Discrete

**Description** The Transfer Fcn Lead or Lag block implements a discrete-time lead or lag compensator of the input. The instantaneous gain of the compensator is one, and the DC gain is equal to  $(1-z)/(1-p)$ , where  $z$  is the zero and  $p$  is the pole of the compensator.

The block implements a lead compensator when  $0 < z < p < 1$ , and implements a lag compensator when  $0 < p < z < 1$ .

**Data Type Support** The Transfer Fcn Lead or Lag block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box



### Pole of compensator (in Z plane)

Set the pole.

### Zero of compensator (in Z plane)

Set the zero.

**Initial condition for previous output**

Set the initial condition for the previous output.

**Initial condition for previous input**

Set the initial condition for the previous input.

**Round toward**

Rounding mode for the fixed-point output.

**Saturate to max or min when overflows occur**

If selected, fixed-point overflows saturate.

**Characteristics**

Direct Feedthrough	Yes
Scalar Expansion	Yes, of initial conditions

# Transfer Fcn Real Zero

**Purpose** Implement a discrete-time transfer function that has a real zero and no pole

**Library** Discrete

**Description** The Transfer Fcn Real Zero block implements a discrete-time transfer function that has a real zero and effectively has no pole.

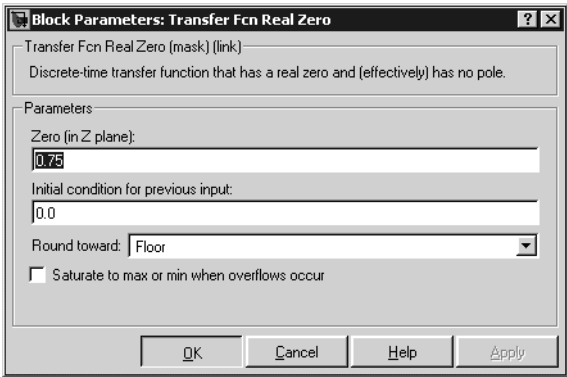
>

$$\frac{z-0.75}{z}$$

>

**Data Type Support** The Transfer Fcn Real Zero block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



- Zero (in Z plane)**  
Set the zero.
- Initial condition for previous input**  
Set the initial condition for the previous input.
- Round toward**  
Rounding mode for the fixed-point output.
- Saturate to max or min when overflows occur**  
If selected, fixed-point overflows saturate.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes, of initial conditions

# Transport Delay

---

**Purpose** Delay the input by a given amount of time

**Library** Continuous

**Description** The Transport Delay block delays the input by a specified amount of time. It can be used to simulate a time delay.



At the start of the simulation, the block outputs the **Initial input** parameter until the simulation time exceeds the **Time delay** parameter, when the block begins generating the delayed input. The **Time delay** parameter must be nonnegative.

The block stores input points and simulation times during a simulation in a buffer whose initial size is defined by the **Initial buffer size** parameter. If the number of points exceeds the buffer size, the block allocates additional memory and Simulink displays a message after the simulation that indicates the total buffer size needed. Because allocating memory slows down the simulation, define this parameter value carefully if simulation speed is an issue. For long time delays, this block might use a large amount of memory, particularly for a dimensionalized input.

When output is required at a time that does not correspond to the times of the stored input values, the block interpolates linearly between points. When the delay is smaller than the step size, the block extrapolates from the last output point, which can produce inaccurate results. Because the block does not have direct feedthrough, it cannot use the current input to calculate its output value. To illustrate this point, consider a fixed-step simulation with a step size of 1 and the current time at  $t = 5$ . If the delay is 0.5, the block needs to generate a point at  $t = 4.5$ . Because the most recent stored time value is at  $t = 4$ , the block performs forward extrapolation.

The Transport Delay block does not interpolate discrete signals. Instead, it returns the discrete value at  $t - t_{delay}$ .

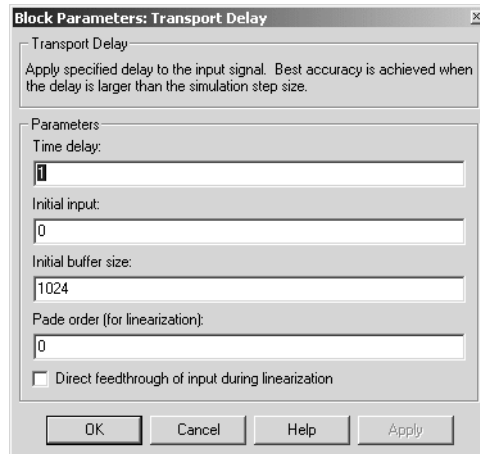
This block differs from the Unit Delay block, which delays and holds the output on sample hits only.

Using `linmod` to linearize a model that contains a Transport Delay block can be troublesome. For more information about ways to avoid the problem, see “Linearizing Models” in the Using Simulink documentation.

## Data Type Support

The Transport Delay block accepts and outputs real signals of type double.

## Parameters and Dialog Box



### Time delay

The amount of simulation time that the input signal is delayed before being propagated to the output. The value must be nonnegative.

### Initial input

The output generated by the block between the start of the simulation and the **Time delay**.

### Initial buffer size

The initial memory allocation for the number of points to store.

### Pade order (for linearization)

The order of the Pade approximation for linearization routines. The default value is 0, which results in a unity gain with no dynamic states. Setting the order to a positive integer  $n$  adds  $n$  states to your model, but results in a more accurate linear model of the transport delay.

### 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.

# Transport Delay

---

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.

Characteristics	Direct Feedthrough	No
	Sample Time	Continuous
	Scalar Expansion	Yes, of input and all parameters except <b>Initial buffer size</b>
	Dimensionalized	Yes
	Zero Crossing	No



**Purpose** Add a trigger port to a subsystem

**Library** Ports & Subsystems

## Description



Adding a Trigger block to a subsystem makes it a triggered subsystem. A triggered subsystem executes once on each integration step when the value of the signal that passes through the trigger port changes in a specifiable way (described below). A subsystem can contain no more than one Trigger block. For more information about triggered subsystems, see “Creating a Model” in the Using Simulink documentation.

The **Trigger type** parameter allows you to choose the type of event that triggers execution of the subsystem:

- rising triggers execution of the subsystem when the control signal rises from a negative or zero value to a positive value (or zero if the initial value is negative).
- falling triggers execution of the subsystem when the control signal falls from a positive or a zero value to a negative value (or zero if the initial value is positive).
- either triggers execution of the subsystem when the signal is either rising or falling.
- function-call causes execution of the subsystem to be controlled by logic internal to an S-function (for more information, see “Function-Call Subsystems”).

You can output the trigger signal by selecting the **Show output port** check box. Selecting this option allows the system to determine what caused the trigger. The width of the signal is the width of the triggering signal. The signal value is

- 1 for a signal that causes a rising trigger
- -1 for a signal that causes a falling trigger
- 2 for a function-call trigger
- 0 otherwise

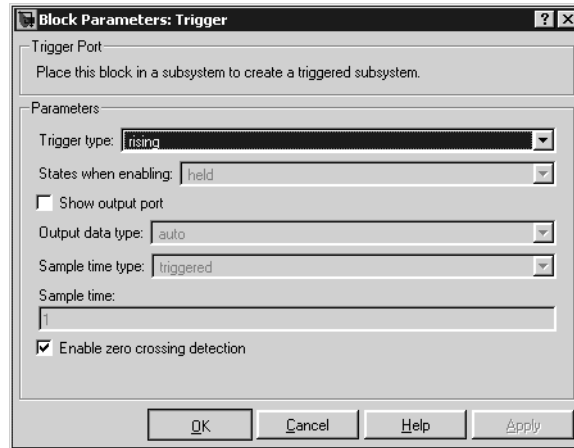
## Data Type Support

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

# Trigger

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

## Parameters and Dialog Box



### Trigger type

The type of event that triggers execution of the subsystem.

### States when enabling

This option is enabled only if you select function-call as the block's trigger type. It specifies whether a function-call enable trigger causes Simulink to reset the states of the subsystem containing this Trigger block to their initial values. Selecting *held* (the default) causes Simulink to leave the states at their current values. Selecting *reset* for this option causes Simulink to reset the states. Selecting *inherit* causes the trigger's *held/reset* setting to be the same as that of the function-call initiator's parent subsystem, for example, an enabled subsystem, or the model's root system if the function-call initiator is at the model's root level. If the parent of the initiator is the model root, the inherited setting is *held*. If the trigger has multiple initiators and its **States when enabling** setting is *inherit*, the parents of all initiators must have the same *held/reset* setting, i.e., either all *held* or all *reset*. For more information about the **States when enabling** setting, see “Function-Call Subsystems” in the “Implementing Block Features” section of *Writing S-Functions*.

## Show output port

If selected, Simulink draws the Trigger block output port and outputs the trigger signal.

## Output data type

Specifies the data type (double or int8) of the trigger output. If you select auto, Simulink sets the data type to be the same as that of the port to which the output is connected. If the port's data type is not double or int8, Simulink signals an error.

---

**Note** The Trigger block ignores the **Data Type Override** setting of the **Fixed-Point Settings** interface.

---

## Enable zero crossing detection

Select to enable zero crossing detection. For more information, see “Zero Crossing Detection” in the Using Simulink documentation.

## Sample time type

This parameter is active only when **Trigger type** is set to function-call. Its value may be triggered or periodic. Select periodic if the caller of the parent function-call subsystem, for example, a Stateflow chart, calls the subsystem once per time step when the subsystem is active (enabled). Otherwise, select triggered. See “Using Bind Actions to Control Function-Call Subsystems” in *Using Stateflow* and the “Function-Call Subsystems” section of *Writing S-functions* for more information.

## Sample time

This parameter is active only when the **Trigger type** is function-call and the **Sample time type** is periodic. Set this parameter to the sample time at which you expect the function-call subsystem that contains this block to be called. See “Specifying Sample Time” in the online documentation for information on how to the value of this parameter. Simulink displays an error if the actual rate at which the subsystem is called differs from the rate that this parameter specifies.

# Trigger

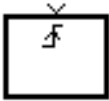
---

Characteristics	Sample Time	Determined by the sample time parameter if the trigger type is function-call and the sample time type is periodic; otherwise, by the signal at the trigger port.
	Dimensionalized	Yes

**Purpose** Generate linear models in the base workspace when triggered

**Library** Model-Wide Utilities

**Description** When triggered, this block calls `linmod` or `dlinmod` to create a linear model for the system at the current operating point. No trimming is performed. The linear model is stored in the base workspace as a structure, along with information about the operating point at which the snapshot was taken. Multiple snapshots are appended to form an array of structures.



The name of the structure used to save the snapshots is the name of the model appended by `_Trigger_Based_Linearization`, for example, `vdp_Trigger_Based_Linearization`. The structure has the follow fields:

Field	Description
<code>a</code>	The A matrix of the linearization
<code>b</code>	The B matrix of the linearization
<code>c</code>	The C matrix of the linearization
<code>d</code>	The D matrix of the linearization
<code>StateName</code>	Names of the model's states
<code>OutputName</code>	Names of the model's output ports
<code>InputName</code>	Names of the model's input ports
<code>OperPoint</code>	A structure that specifies the operating point of the linearization. The structure specifies the value of the model's states ( <code>OperPoint.x</code> ) and inputs ( <code>OperPoint.u</code> ) at the operating point time ( <code>OperPoint.t</code> ).
<code>Ts</code>	The sample time of the linearization for a discrete linearization

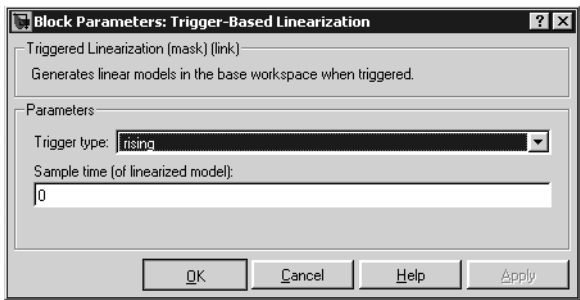
Use the Time-Based Linearization block to generate linear models at predetermined times.

# Trigger-Based Linearization

## Data Type Support

The trigger port accepts signals of any data type supported by Simulink.

## Parameters and Dialog Box



### Trigger type

Type of event on the trigger input signal that triggers generation of a linear model. See the **Trigger type** parameter of the Trigger block for an explanation of the various trigger types that you can select.

### Sample time (of linearized model)

Specify a sample time to create a discrete-time linearization of the model (see “Discrete-Time System Linearization” on page 2-4).

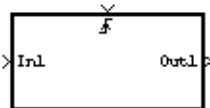
## Characteristics

Sample Time	Specified in the <b>Sample time</b> parameter
Dimensionalized	No

**Purpose** Represent a subsystem whose execution is triggered by external input

**Library** Ports & Subsystems

**Description** This block is a Subsystem block that is preconfigured to serve as the starting point for creating a triggered subsystem (see “Triggered Subsystems”).



# Trigonometric Function

**Purpose** Perform a trigonometric function

**Library** Math Operations

**Description** The Trigonometric Function block performs numerous common trigonometric functions.



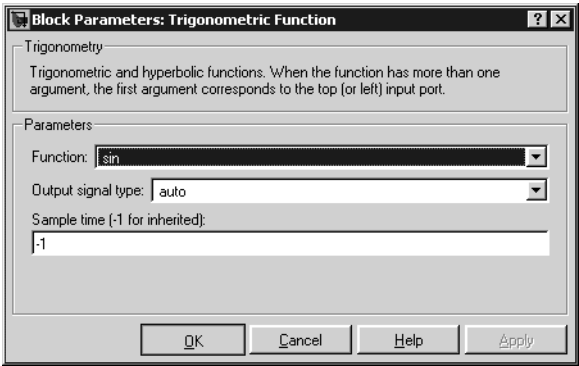
You can select one of these functions from the **Function** list: sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, and tanh. The block output is the result of the operation of the function on the input or inputs.

The name of the function appears on the block. Simulink automatically draws the appropriate number of input ports. The block accepts and outputs real or complex signals of type double.

Use the Trigonometric Function block instead of the Fcn block when you want dimensionalized output, because the Fcn block can produce only scalar output.

**Data Type Support** The Trigonometric Function block accepts and outputs real or complex signals of type double.

**Parameters and Dialog Box**



**Function** The trigonometric function.

**Output signal type** Type of signal (complex or real) to output.



**Sample time (-1 for inherited)**

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

**Characteristics**

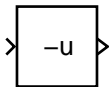
Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes, of the input when the function requires two inputs
Dimensionalized	Yes
Zero Crossing	No

# Unary Minus

**Purpose** Negate the input

**Library** Math Operations

**Description** The Unary Minus block negates the input. The block accepts only signed data types.

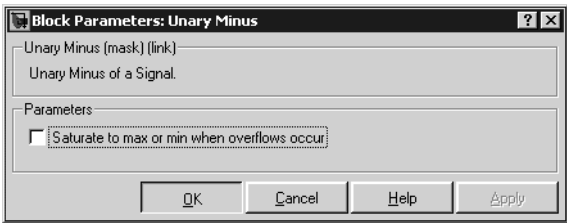


For signed data types, you cannot accurately negate the most negative value since the result is not representable by the data type. In this case, the behavior of the block is controlled by the **Saturate to max or min when overflows occur** check box. If selected, the most negative value of the data type wraps to the most positive value. If not selected, the operation has no effect. If an overflow occurs, then a warning is returned to the MATLAB command line.

For example, suppose the block input is an 8-bit signed integer. The range of this data type is from -128 to 127, and the negation of -128 is not representable. If you select the **Saturate to max or min when overflows occur** check box, then the negation of -128 is 127. If it is not selected, then the negation of -128 remains at -128.

**Data Type Support** The Unary Minus block accepts signals of any data type supported by Simulink except unsigned integers, including fixed-point data types.

**Parameters and Dialog Box**



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

<b>Characteristics</b>	Direct Feedthrough	No
	Scalar Expansion	Yes, of input or initial conditions

**Purpose** Generate uniformly distributed random numbers

**Library** Sources

## Description



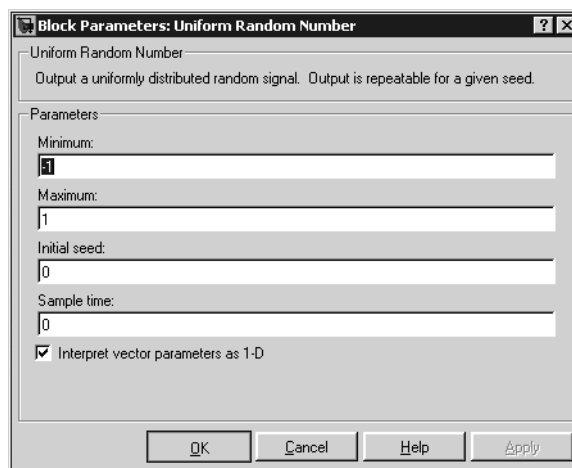
The Uniform Random Number block generates uniformly distributed random numbers over a specifiable interval with a specifiable starting seed. The seed is reset each time a simulation starts. The generated sequence is repeatable and can be produced by any Uniform Random Number block with the same seed and parameters. To generate normally distributed random numbers, use the Random Number block.

Avoid integrating a random signal, because solvers are meant to integrate relatively smooth signals. Instead, use the Band-Limited White Noise block.

The block's numeric parameters must be of the same dimensions after scalar expansion. If the **Interpret vector parameters as 1-D** option is off, the block outputs a signal of the same dimensions and dimensionality as the parameters. If the **Interpret vector parameters as 1-D** option is on and the numeric parameters are row or column vectors (i.e., single row or column 2-D arrays), the block outputs a vector (1-D array) signal; otherwise, the block outputs a signal of the same dimensions as the parameters.

**Data Type Support** The Uniform Random Number block outputs a real signal of type double.

## Parameters and Dialog Box

A screenshot of the 'Block Parameters: Uniform Random Number' dialog box. The title bar says 'Block Parameters: Uniform Random Number'. Inside, there's a description: 'Uniform Random Number' and 'Output a uniformly distributed random signal. Output is repeatable for a given seed.' Below this is a 'Parameters' section with four input fields: 'Minimum:' with a value of -1, 'Maximum:' with a value of 1, 'Initial seed:' with a value of 0, and 'Sample time:' with a value of 0. At the bottom of the parameters section is a checked checkbox labeled 'Interpret vector parameters as 1-D'. At the very bottom are four buttons: 'OK', 'Cancel', 'Help', and 'Apply'.

# Uniform Random Number

---

**Minimum**

The minimum of the interval. The default is -1.

**Maximum**

The maximum of the interval. The default is 1.

**Initial seed**

The starting seed for the random number generator. The default is 0.

**Sample time**

The sample period. The default is 0. See “Specifying Sample Time” in the online documentation for more information.

**Interpret vector parameters as 1-D**

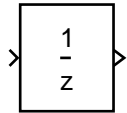
If selected, column or row matrix values for the Step block’s numeric parameters result in a vector output signal; otherwise, the block outputs a signal of the same dimensionality as the parameters. If this option is not selected, the block always outputs a signal of the same dimensionality as the block’s numeric parameters.

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Delay a signal one sample period

**Library** Discrete

**Description**



The Unit Delay block delays its input by the specified sample period. This block is equivalent to the  $z^{-1}$  discrete-time operator. The block accepts one input and generates one output, which can be either both scalar or both vector. If the input is a vector, all elements of the vector are delayed by the same sample period.

You specify the block output for the first sampling period with the **Initial conditions** parameter. Careful selection of this parameter can minimize unwanted output behavior. The time between samples is specified with the **Sample time** parameter. A setting of -1 means the sample time is inherited.

The Unit Delay block provides a mechanism for discretizing one or more signals in time, or for resampling the signal at a different rate. If your model contains multirate transitions, then you must add Unit Delay blocks between the slow-to-fast transitions. The sample rate of the Unit Delay block must be set to that of the slower block. For fast-to-slow transitions, use the Zero Order Hold block. For more information about multirate transitions, refer to the Simulink or the Real-Time Workshop documentation.

---

**Note** The Unit Delay block accepts continuous signals. When it has a continuous sample time, the block is equivalent to the Simulink Memory block.

---

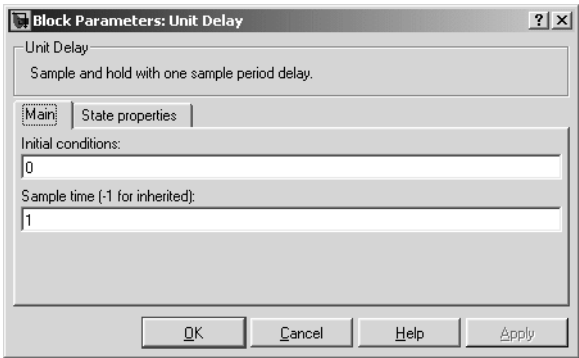
**Data Type Support**

The Unit Delay block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. If the data type of the input signal is user-defined, the initial condition must be zero.

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

# Unit Delay

## Parameters and Dialog Box



### Initial conditions

The output of the simulation for the first sampling period, during which the output of the Unit Delay block is otherwise undefined. The **Initial conditions** parameter is converted from a double to the input data type offline using round-to-nearest and saturation.

### Sample time (-1 for inherited)

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

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

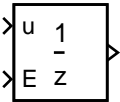
## Characteristics

Direct Feedthrough	No
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes, of input or initial conditions
States	Yes—inherited from driving block for nonfixed-point data types.
Dimensionalized	Yes
Zero Crossing	No

**Purpose** Delay a signal one sample period, if the external enable signal is on

**Library** Additional Math & Discrete / Additional Discrete

**Description** The Unit Delay Enabled block delays a signal by one sample period when the external enable signal E is on. While the enable is off, the block is disabled. It holds the current state at the same value and outputs that value. The enable signal is on when E is not 0, and off when E is 0.



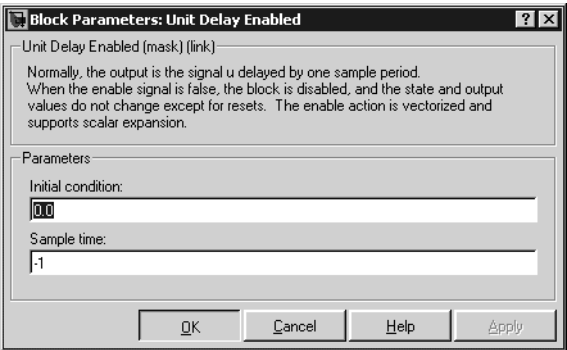
You specify the block output for the first sampling period with the value **Initial condition** parameter.

The output data type is the same as the input u data type. The data type of the input u and the enable E can be any data type.

You input the sample time with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

**Data Type Support** The Unit Delay Enabled block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box



**Initial condition**  
Initial condition.

# Unit Delay Enabled

---

**Sample time**

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

**Characteristics**

Direct Feedthrough	No
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes

**See Also**

Unit Delay, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV



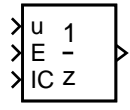
## Purpose

Delay a signal one sample period, if the external enable signal is on, with an external initial condition

## Library

Additional Math & Discrete / Additional Discrete

## Description



The Unit Delay Enabled External IC block delays a signal by one sample period when the enable signal E is on. While the enable is off, the block holds the current state at the same value and outputs that value. The enable E is on when E is not 0, and off when E is 0.

The initial condition of this block is given by the signal IC.

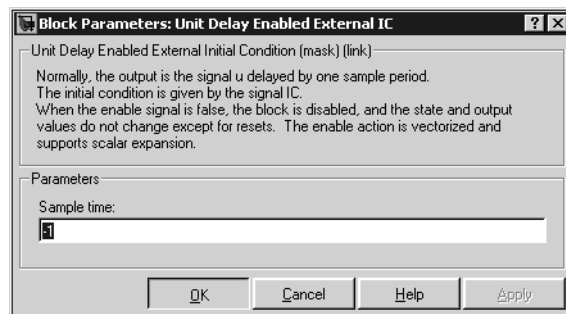
The input u and IC data types must be the same, and are any data type. The output data type is the same as u and IC. The enable E is any data type.

You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

## Data Type Support

The Unit Delay Enabled External IC block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box



### Sample time

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

# Unit Delay Enabled External IC

---

Characteristics	Direct Feedthrough	Yes, of the reset input port No, of the enable input port Yes, of the external IC port
	Sample Time	Specified in the <b>Sample time</b> 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

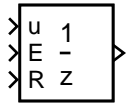
## Purpose

Delay a signal one sample period, if the external enable signal is on, with an external Boolean reset

## Library

Additional Math & Discrete / Additional Discrete

## Description



The Unit Delay Enabled Resettable block combines the features of the Unit Delay Enabled and Unit Delay Resettable blocks.

The block can reset its state based on an external reset signal R. When the enable signal E is on and the reset signal R is false, the block outputs the input signal delayed by one sample period.

When the enable signal E is on and the reset signal R is true, the block resets the current state to the initial condition, specified by the **Initial condition** parameter, and outputs that state delayed by one sample period.

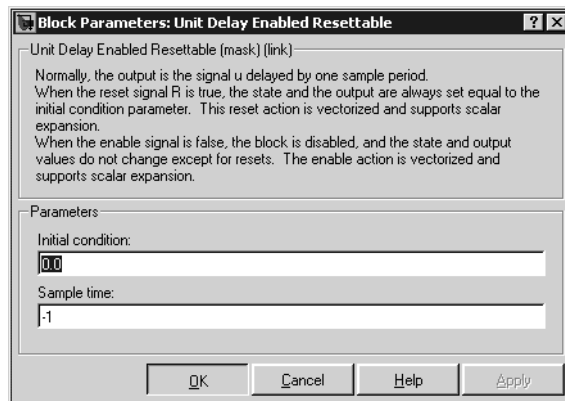
When the enable signal is off, the block is disabled, and the state and output do not change except for resets. The enable signal is on when E is not 0, and off when E is 0.

You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

## Data Type Support

The Unit Delay Enabled Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box



# Unit Delay Enabled Resettable

---

**Initial condition**

The initial output of the simulation.

**Sample time**

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

**Characteristics**

Direct Feedthrough	No, of the input port No, of the enable port Yes, of the reset port
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes

**See Also**

Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

# Unit Delay Enabled Resettable External IC

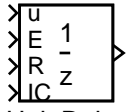
## Purpose

Delay a signal one sample period, if the external enable signal is on, with an external Boolean reset and initial condition

## Library

Additional Math & Discrete / Additional Discrete

## Description



The Unit Delay Enabled Resettable External IC block combines the features of the Unit Delay Enabled, Unit Delay External IC, and Unit Delay Resettable blocks.

The block can reset its state based on an external reset signal R. When the enable signal E is on and the reset signal R is false, the block outputs the input signal delayed by one sample period.

When the enable signal E is on and the reset signal R is true, the block resets the current state to the initial condition given by the signal IC, and outputs that state delayed by one sample period.

When the enable signal is off, the block is disabled, and the state and output do not change except for resets. The enable signal is on when E is not 0, and off when E is 0.

The output data type is the same as the input **u** and the initial condition **IC** data type, which can be any data type, but must be the same. The enable **E** and reset **R** can be any data type.

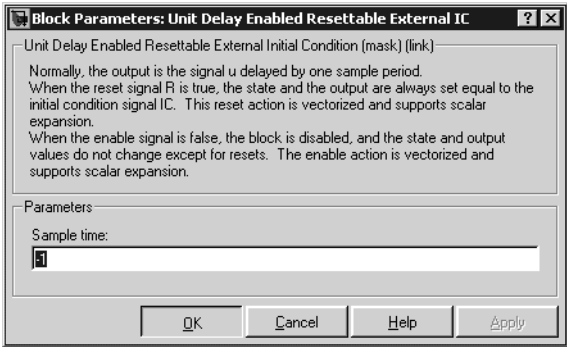
You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

## Data Type Support

The Unit Delay Enabled Resettable External IC block accepts signals of any data type supported by Simulink, including fixed-point data types.

# 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 “Specifying Sample Time” in the online documentation for more information.

### Characteristics

Direct Feedthrough	No, of the input port No, of the enable port Yes, of the enable port Yes, of the external IC port
Sample Time	Specified in the <b>Sample time</b> parameter
Scalar Expansion	Yes

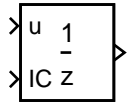
### See Also

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

**Purpose** Delay a signal one sample period, with an external initial condition

**Library** Additional Math & Discrete / Additional Discrete

**Description** The Unit Delay External IC block delays its input by one sample period. This block is equivalent to the  $z^{-1}$  discrete-time operator. The block accepts one input and generates one output, both of which can be scalar or vector. If the input is a vector, all elements of the vector are delayed by the same sample period.



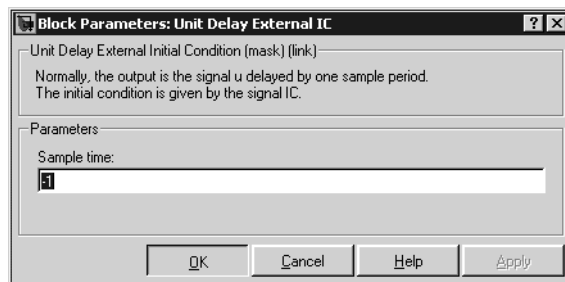
The block's output for the first sample period is equal to the signal IC.

The input *u* and initial condition IC data types must be the same, and are any data type.

You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

**Data Type Support** The Unit Delay External IC block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box



### Sample time

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

# Unit Delay External IC

---

Characteristics	Direct Feedthrough	No, of the input port Yes, of the external IC port
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes

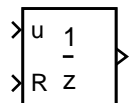
**See Also**      Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay Resettable, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV



**Purpose** Delay a signal one sample period, with an external Boolean reset

**Library** Additional Math & Discrete / Additional Discrete

**Description** The Unit Delay Resettable block delays a signal one sample period.

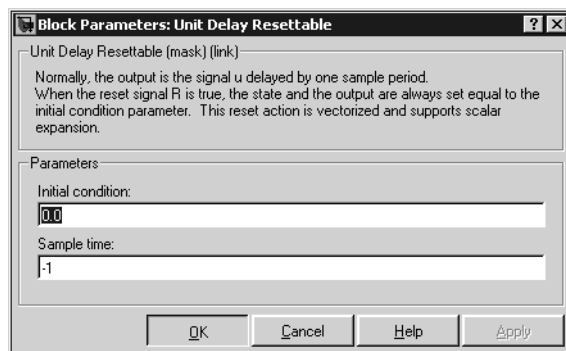


The block can reset its state based on an external reset signal R. The block has two input ports, one for the input signal u and the other for the external reset signal R. When the reset signal is false, the block outputs the input signal delayed by one time step. When the reset signal is true, the block resets the current state to the initial condition, specified by the **Initial condition** parameter, and outputs that state delayed by one time step.

You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

**Data Type Support** The Unit Delay Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box



### Initial condition

Specify the initial output of the simulation.

### Sample time

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

# Unit Delay Resettable

---

Characteristics	Direct Feedthrough	No, of the input port Yes, of the reset port
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes

**See Also**      Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable External IC, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

# Unit Delay Resettable External IC

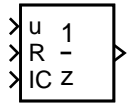
## Purpose

Delay a signal one sample period, with an external Boolean reset and initial condition

## Library

Additional Math & Discrete / Additional Discrete

## Description



The Unit Delay Resettable External IC block delays a signal one sample period.

The block can reset its state based on an external reset signal R. The block has two input ports, one for the input signal u and the other for the reset signal R. When the reset signal is false, the block outputs the input signal delayed by one time step. When the reset signal is true, the block resets the current state to the initial condition given by the signal IC and outputs that state delayed by one time step.

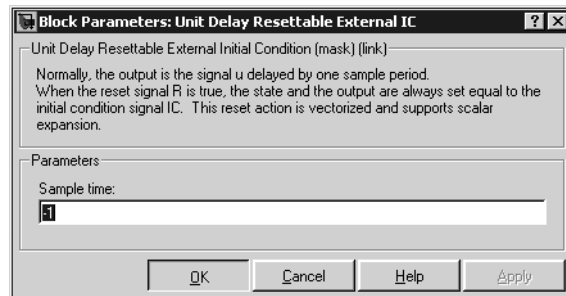
The input u and initial condition IC must be the same data type, but can be any data type. The output is the same data type as the inputs u and IC. The reset R can be any data type.

You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

## Data Type Support

The Unit Delay Resettable External IC block accepts signals of any data type supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box



### Sample time

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

# Unit Delay Resettable External IC

---

Characteristics	Direct Feedthrough	No, of the input port Yes, of the reset port Yes, of the external IC port
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	Yes

**See Also**      Unit Delay, Unit Delay Enabled, Unit Delay Enabled External IC, Unit Delay Enabled Resettable, Unit Delay Enabled Resettable External IC, Unit Delay External IC, Unit Delay Resettable, Unit Delay With Preview Enabled, Unit Delay With Preview Enabled Resettable, Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable, Unit Delay With Preview Resettable External RV

# Unit Delay With Preview Enabled

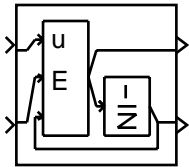
## Purpose

Output the signal and the signal delayed by one sample period, if the external enable signal is on

## Library

Additional Math & Discrete / Additional Discrete

## Description



The Unit Delay With Preview Enabled block supports calculations that have feedback and depend on the current input.

The block has two output ports. When the external enable signal **E** is on, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period. The block has two input ports, one for the input signal **u** and the other for the enable signal **E**.

When the enable signal **E** is off, the block is disabled, and the state and output values do not change, except for resets. The enable signal is on when **E** is not 0, and off when **E** is 0.

The input **u** and initial condition **IC** must be the same data type, but can be any data type. The output is the same data type as the inputs **u** and **IC**. The reset **R** can be any data type.

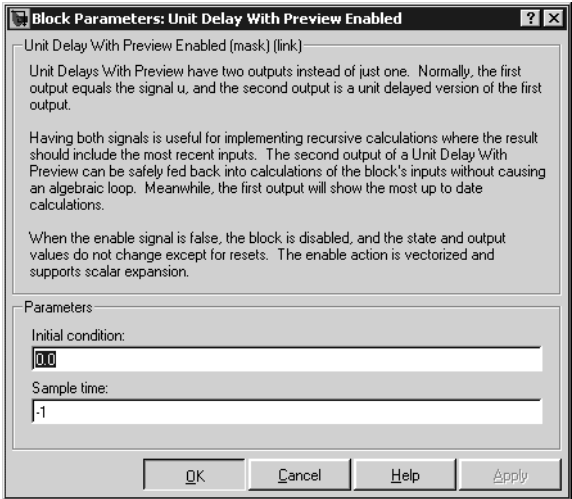
You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

## Data Type Support

The Unit Delay With Preview Enabled block accepts signals of any data type supported by Simulink, including fixed-point data types.

# Unit Delay With Preview Enabled

## Parameters and Dialog Box



### Initial condition

Specify the initial condition.

### Sample time

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

## Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the <b>Sample time</b> 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

# Unit Delay With Preview Enabled Resettable

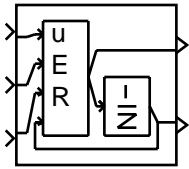
## Purpose

Output the signal and the signal delayed by one sample period, if the external enable signal is on, with an external Boolean reset

## Library

Additional Math & Discrete / Additional Discrete

## Description



The Unit Delay With Preview Enabled Resettable block supports calculations that have feedback and depend on the current input.

The block can reset its state based on an external reset signal **R**. The block has two output ports. When the external enable signal **E** is on and the reset **R** is false, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period. The block has two input ports, one for the input signal **u** and the other for the enable signal **E**.

When the enable signal **E** is on and the reset **R** is true, the block resets the current state to the initial condition given by the **Initial condition** parameter. The block outputs that state delayed by one sample time through the lower output port, and outputs the state without a delay through the upper output port.

When the Enable signal is off, the block is disabled, and the state and output values do not change, except for resets. The enable signal is on when **E** is not 0, and off when **E** is 0.

The input **u** and initial condition **IC** must be the same data type, but can be any data type. The output is the same data type as the inputs **u** and **IC**. The reset **R** can be any data type.

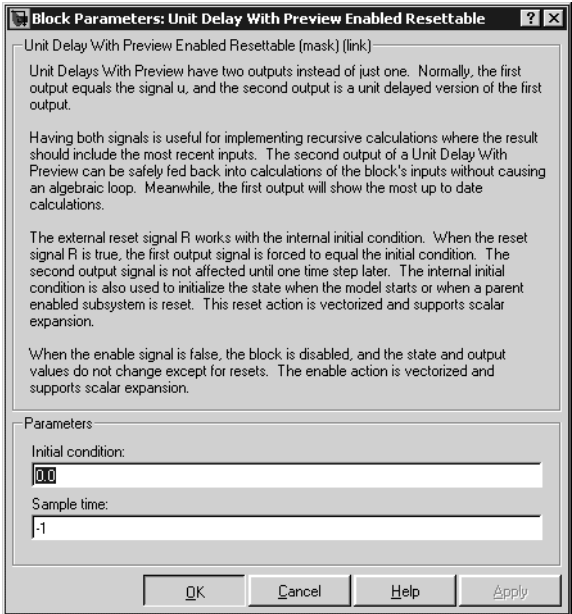
You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

## Data Type Support

The Unit Delay With Preview Enabled Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

# Unit Delay With Preview Enabled Resettable

## Parameters and Dialog Box



### Initial condition

Specify the initial condition.

### Sample time

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

### Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the <b>Sample time</b> 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, Unit Delay With Preview  
Resettable External RV

# Unit Delay With Preview Enabled Resettable External RV

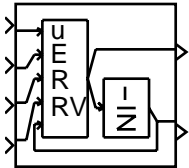
## Purpose

Output the signal and the signal delayed by one sample period, if the external enable signal is on, with an external RV reset

## Library

Additional Math & Discrete / Additional Discrete

## Description



The Unit Delay With Preview Enabled Resettable External RV block supports calculations that have feedback and depend on the current input.

The block can reset its state based on an external reset signal R. The block has two output ports. When the external enable signal E is on and the reset R is false, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period. The block has two input ports, one for the input signal u and the other for the enable signal E.

When the enable signal E is on and the reset R is true, the upper output signal is forced to equal the external reset signal RV. The lower output signal is not affected until one time step later, at which time it is equal to the external reset signal RV at the previous time step. The block uses the internal **Initial condition** only when the model starts or when a parent enabled subsystem is used. The internal **Initial condition** only affects the lower output signal. The first output is only affected through feedback.

When the Enable signal is off, the block is disabled, and the state and output values do not change, except for resets. The enable signal is on when E is not 0, and off when E is 0.

The input u and initial condition IC must be the same data type, but can be any data type. The output is the same data type as the inputs u and IC. The reset R can be any data type.

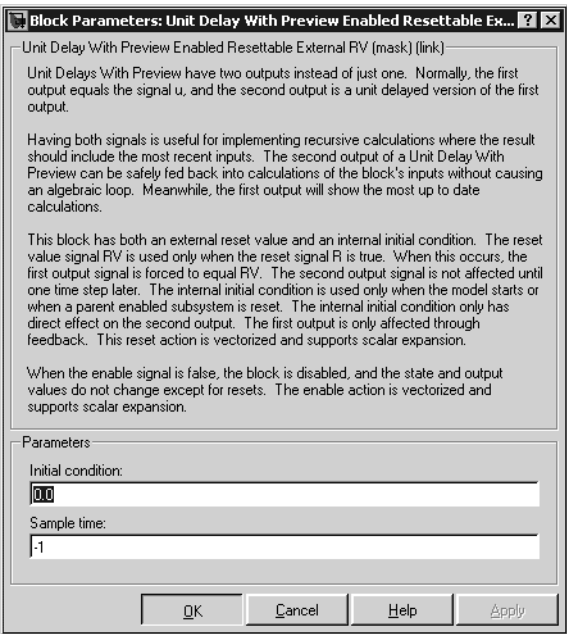
You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

## Data Type Support

The Unit Delay With Preview Enabled Resettable External RV block accepts signals of any data type supported by Simulink, including fixed-point data types.

# Unit Delay With Preview Enabled Resettable External RV

## Parameters and Dialog Box



### Initial condition

Specify the initial condition.

### Sample time

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

## Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the <b>Sample time</b> 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, Unit Delay With Preview Resettable  
External RV

# Unit Delay With Preview Resettable

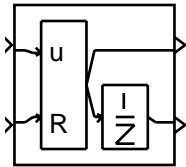
## Purpose

Output the signal and the signal delayed by one sample period, with an external Boolean reset

## Library

Additional Math & Discrete / Additional Discrete

## Description



The Unit Delay With Preview Resettable block supports calculations that have feedback and depend on the current input.

The block can reset its state based on an external reset signal **R**. The block has two output ports. When the reset **R** is false, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period.

When the reset **R** is true, the block resets the current state to the initial condition given by the **Initial condition** parameter. The block outputs that state delayed by one sample time through the lower output port, and outputs the state without a delay through the upper output port.

The input **u** and initial condition **IC** must be the same data type, but can be any data type. The output is the same data type as the inputs **u** and **IC**. The reset **R** can be any data type.

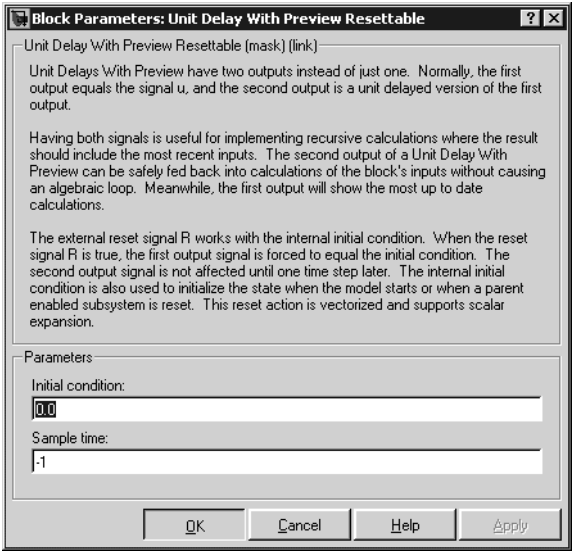
You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

## Data Type Support

The Unit Delay With Preview Resettable block accepts signals of any data type supported by Simulink, including fixed-point data types.

# Unit Delay With Preview Resettable

## Parameters and Dialog Box



### Initial condition

Specify the initial condition.

### Sample time

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

## Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the <b>Sample time</b> 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 Resetable External RV

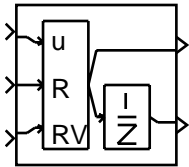
## Purpose

Output the signal and the signal delayed by one sample period, with an external RV reset

## Library

Additional Math & Discrete / Additional Discrete

## Description



The Unit Delay With Preview Resetable External RV block supports calculations that have feedback and depend on the current input.

The block can reset its state based on an external reset signal R. The block has two output ports. When the external reset R is false, the upper port outputs the signal and the lower port outputs the signal delayed by one sample period.

When the external reset R is true, the upper output signal is forced to equal the external reset signal RV. The lower output signal is not affected until one time step later, at which time it is equal to the external reset signal RV at the previous time step. The block uses the internal **Initial condition** only when the model starts or when a parent enabled subsystem is used. The internal **Initial condition** only affects the lower output signal. The first output is only affected through feedback.

The input u and initial condition IC must be the same data type, but can be any data type. The output is the same data type as the inputs u and IC. The reset R can be any data type.

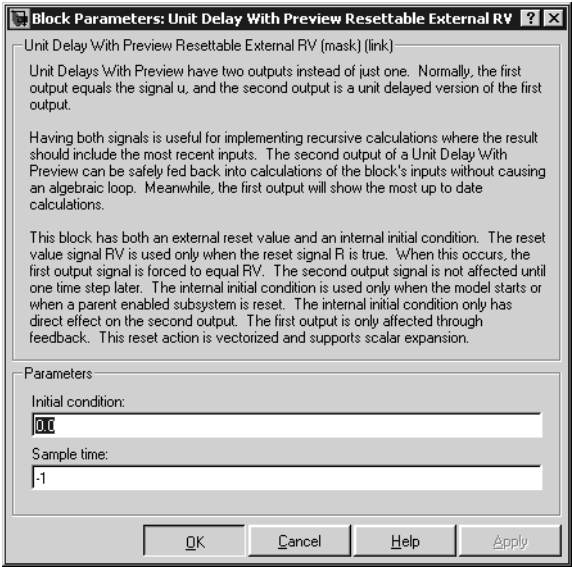
You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

## Data Type Support

The Unit Delay With Preview Resetable External RV block accepts signals of any data type supported by Simulink, including fixed-point data types.

# Unit Delay With Preview Resettable External RV

## Parameters and Dialog Box



### Initial condition

Specify the initial condition.

### Sample time

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

## Characteristics

Direct Feedthrough	Yes, to upper output port No, to lower output port
Sample Time	Specified in the <b>Sample time</b> 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 External RV

---

Unit Delay With Preview Enabled Resettable External RV, Unit Delay With Preview Resettable

# Variable Transport Delay

## Purpose

Delay the input by a variable amount of time

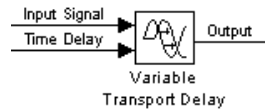
## Library

Continuous

## Description



The Variable Transport Delay block can be used to simulate a variable time delay. The block accepts two inputs: the first input is the signal that passes through the block; the second input is the time delay, as shown in this icon.



The **Maximum delay** parameter defines the largest value the time delay input can have. The block clips values of the delay that exceed this value. The **Maximum delay** must be greater than or equal to zero. If the time delay becomes negative, the block clips it to zero and issues a warning message.

During the simulation, the block stores time and input value pairs in an internal buffer. At the start of the simulation, the block outputs the **Initial input** parameter until the simulation time exceeds the time delay input. Then, at each simulation step the block outputs the signal at the time that corresponds to the current simulation time minus the delay time.

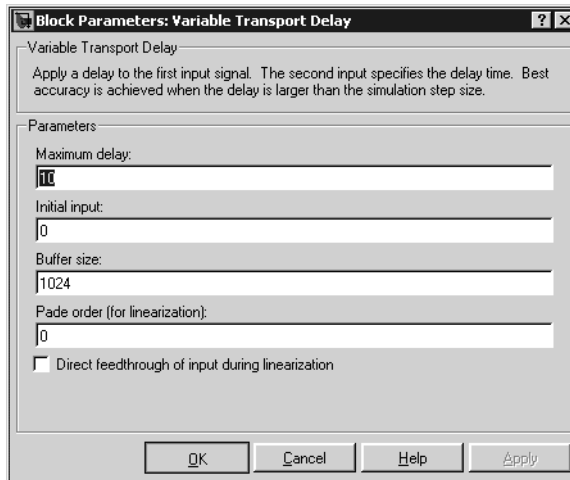
When output is required at a time that does not correspond to the times of the stored input values, the block interpolates linearly between points. If the time delay is smaller than the step size, the block extrapolates an output point. This can result in less accurate results. The block cannot use the current input to calculate its output value because the block does not have direct feedthrough at this port. To illustrate this point, consider a fixed-step simulation with a step size of 1 and the current time at  $t = 5$ . If the delay is 0.5, the block needs to generate a point at  $t = 4.5$ . Because the most recent stored time value is at  $t = 4$ , the block performs forward extrapolation.

The Variable Transport Delay block does not interpolate discrete signals. Instead, it returns the discrete value at  $t - t_{delay}$ .

## Data Type Support

The Variable Transport Delay block accepts and outputs real signals of type double.

## Parameters and Dialog Box



### Maximum delay

The maximum value of the time delay input. The value cannot be negative. The default is 10.

### Initial input

The output generated by the block until the simulation time first exceeds the time delay input. The default is 0.

### Buffer size

The number of points the block can store. The default is 1024.

### Pade order (for linearization)

The order of the Pade approximation for linearization routines. The default value is 0, which results in a unity gain with no dynamic states. Setting the order to a positive integer  $n$  adds  $n$  states to your model, but results in a more accurate linear model of the transport delay.

### 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.

# Variable Transport Delay

---

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.

<b>Characteristics</b>	Direct Feedthrough	Yes, of the time delay (second) input
	Sample Time	Continuous
	Scalar Expansion	Yes, of input and all parameters except <b>Buffer size</b>
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Implement a weighted moving average

**Library** Discrete

## Description



The Weighted Moving Average block samples and holds the  $N$  most recent inputs, multiplies each input by a specified value (given by the **Weights** parameter), and stacks them in a vector. This block supports both single-input/single-output (SISO) and single-input/multi-output (SIMO) modes.

For the SISO mode, the **Weights** parameter is specified as a row vector. For the SIMO mode, the weights are specified as a matrix where each row corresponds to a separate output.

The **Initial condition** parameter provides the initial values for all times preceding the start time. You specify the time interval between samples with the **Sample time** parameter.

You can choose whether or not to specify the data type and scaling of the weights in the dialog with the **Gain data type and scaling** parameter. If you select Specify via dialog for this parameter, the **Parameter data type**, **Parameter scaling**, and **Parameter scaling mode** parameters become visible.

You can specify the scaling for the weights with the **Parameter scaling** and **Parameter scaling mode** parameters. If **Parameter data type** is a generalized fixed-point number such as `sfix(16)`, the **Parameter scaling mode** list provides you with these scaling modes:

- **Use Specified Scaling**—This mode uses the [Slope Bias] or binary point-only scaling specified by the **Parameter scaling** parameter (for example,  $2^{-10}$ ).
- **Best Precision: Element-wise**—This mode produces binary points such that the precision is maximized for each element of the **Weights** parameter.
- **Best Precision: Row-wise**—This mode produces a common binary point for each element of the **Weights** row based on the best precision for the largest value of that row.
- **Best Precision: Column-wise**—This mode produces a common binary point for each element of the **Weights** column based on the best precision for the largest value of that column.

# Weighted Moving Average

- **Best Precision: Matrix-wise**—This mode produces a common binary point for each element of the **Weights** matrix based on the best precision for the largest value of the matrix.

If the weights are specified as a row vector, then scaling element-wise and column-wise produce the same result, while scaling matrix-wise and row-wise produce the same result.

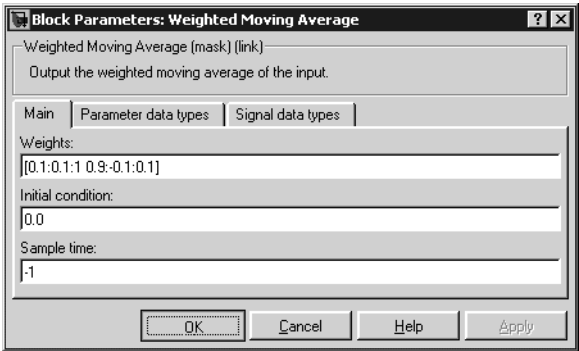
The Weighted Moving Average block first multiplies its inputs by the **Weights** parameter, converts those results to the output data type using the specified rounding and overflow modes, and then carries out the summation.

## Data Type Support

The Weighted Moving Average block supports all data types supported by Simulink, including fixed-point data types.

## Parameters and Dialog Box

The **Main** pane of the Weighted Moving Average block dialog appears as follows:



## Weights

Specify the weights of the moving average; one row per output. The **Weights** parameter is converted from doubles to the specified data type offline using round-to-nearest and saturation.

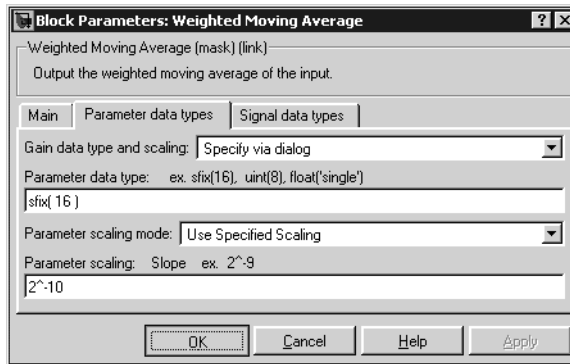
## Initial condition

Specify the initial values for all times preceding the start time. The **Initial condition** parameter is converted from doubles to the input data type offline using round-to-nearest and saturation.

## Sample time

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

The **Parameter data types** pane of the Weighted Moving Average block dialog appears as follows:



## Gain data type and scaling

Choose whether to specify the data type of the weights in the block dialog or via an internal rule. If you select **Specify via dialog**, the **Parameter data type**, **Parameter scaling**, and **Parameter scaling mode** parameters become visible.

## Parameter data type

Specify the data type of the weights. This parameter is only visible if you select **Specify via dialog** for the **Gain data type and scaling** parameter.

## Parameter scaling mode

This drop-down list enables you to specify the parameter scaling in the dialog or by an inherited rule. This parameter is only visible if you select **Specify via dialog** for the **Gain data type and scaling** parameter.

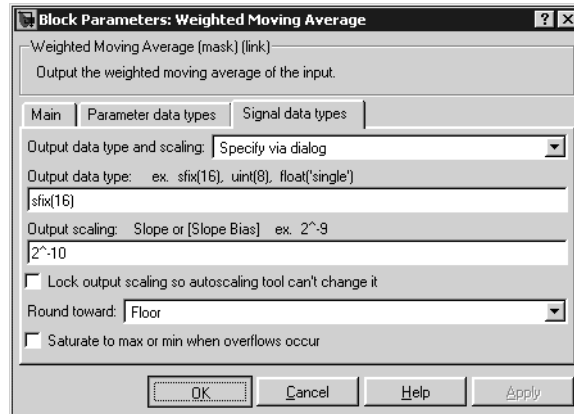
## Parameter scaling

Set the scaling of the weights using binary point-only or [Slope Bias] scaling. Additionally, the **Weights** vector or matrix can be scaled using the constant vector or constant matrix scaling modes for maximizing precision. These scaling modes are available only for generalized fixed-point data

# Weighted Moving Average

types. This parameter is only visible if you select **Specify** via dialog for the **Gain data type and scaling** parameter.

The **Signal data types** pane of the Weighted Moving Average block dialog appears as follows:



## Output data type and scaling

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

## Output data type

Specify the output data type.

## Output scaling

Set the output scaling using binary point-only or [Slope Bias] scaling. These scaling modes are available only for generalized fixed-point data types.

## Lock output scaling so autoscaling tool can't change it

If selected, **Output scaling** is locked. This feature is available only for generalized fixed-point output.

## Round toward

Rounding mode for the fixed-point output.



**Saturate to max or min when overflows occur**

If selected, fixed-point overflows saturate.

**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

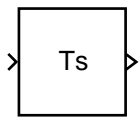
# Weighted Sample Time

---

**Purpose** Support calculations involving sample time

**Library** Signal Attributes

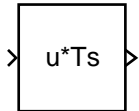
**Description** The Weighted Sample Time block is an implementation of the Weighted Sample Time Math block. See Weighted Sample Time Math on page 1-563 for more information.



**Purpose** Support calculations involving sample time

**Library** Math Operations

**Description** The Weighted Sample Time Math block adds, subtracts, multiplies, or divides the input signal,  $u$ , by a weighted sample time  $T_s$ .

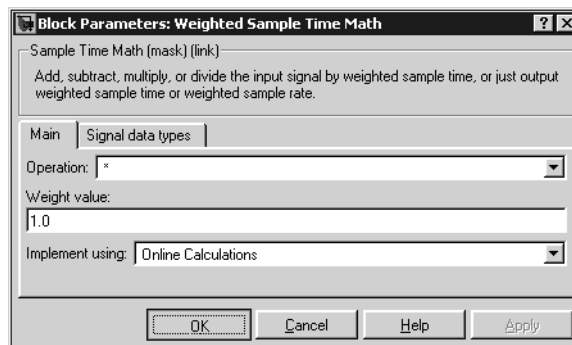


You specify the math operation with the **Operation** parameter. Additionally, you can specify to use only the weight with either the sample time or its inverse.

Enter the weighting factor in the **Weight value** parameter. If the weight is 1,  $w$  is removed from the equation.

**Data Type Support** The Weighted Sample Time Math block accepts signals of any data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box** The **Main** pane of the Weighted Sample Time Math block dialog appears as follows:



## Operation

Specify operation to use: +, -, \*, /,  $T_s$  only,  $1/T_s$  only.

## Weight value

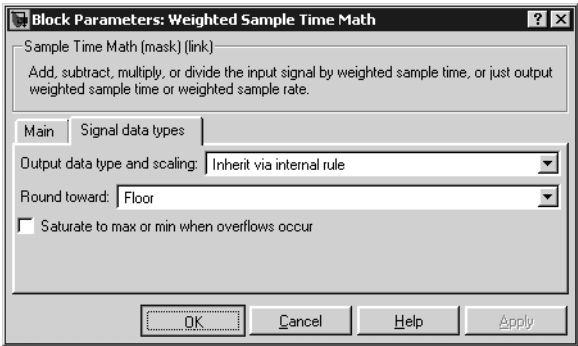
Enter weight of sample time.

# Weighted Sample Time Math

## Implement using

Specify online calculations or offline scaling adjustment. This parameter is only visible for some **Operation** parameter settings.

The **Signal data types** pane of the Weighted Sample Time Math block dialog appears as follows:



## Output data type and scaling

Specify whether the output data type and scaling are inherited by an internal rule or by backpropagation.

## Round toward

Select the rounding mode for fixed-point operations. This parameter is only visible if you select Online Calculations for the **Implement using** parameter.

## Saturate to max or min when overflows occur

If selected, fixed-point overflows saturate. This parameter is only visible if you select Online Calculations for the **Implement using** parameter.

<b>Characteristics</b>	Direct Feedthrough	For all math operations options except $T_s$ and $1/T_s$
	Scalar Expansion	No, the weight is always a scalar

## Purpose

Repeatedly execute the contents of a subsystem at the current time step while a condition is satisfied.

## Library

Ports & Subsystems / While Iterator Subsystem

## Description



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.

---

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.

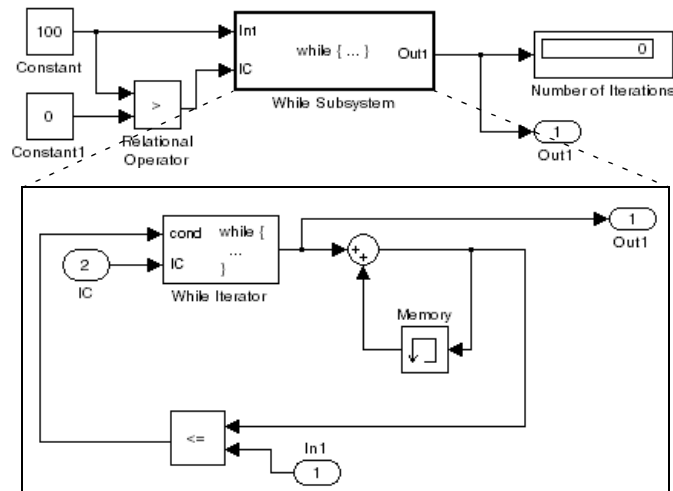
- while

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.

# While Iterator

**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.

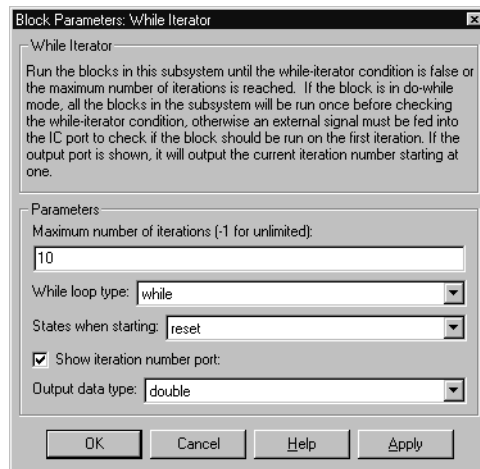
```
max_sum = 100;
sum = 0;
iteration_number = 0;
cond = (max_sum > 0);
while (cond != 0) {
    iteration_number = iteration_number + 1;
    sum = sum + iteration_number;
    if (sum > max_sum OR iteration_number > max_iterations)
        cond = 0;
}
```

## Data Type Support

Acceptable data inputs for the condition ports are any type supported by Simulink, as well as any fixed-point type, that includes a 0 value. For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using 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

# While Iterator

---

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 default), the While Iterator block outputs its iteration value. This value starts at 1 and is incremented by 1 for each succeeding iteration.

**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.

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	No
	Zero Crossing	No



**Purpose** Represent a subsystem that executes repeatedly while a condition is satisfied during a 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 “Control Flow Blocks” for more information.

```
> In1
  while { ... Out1 }
> It
```

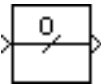
# Width

---

**Purpose**                      Output the width of the input vector

**Library**                     Signal Attributes

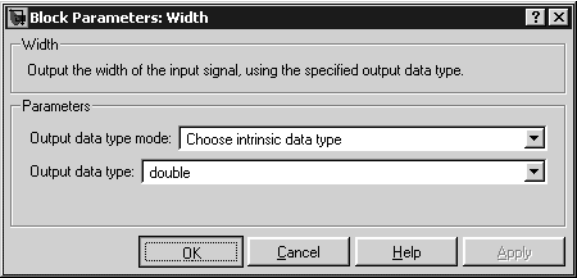
**Description**                The Width block generates as output the width of its input vector.



**Data Type Support**        The Width block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types. The Width block supports mixed-type signal vectors.

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

**Parameters and Dialog Box**



---

**Note** The Width block ignores the **Data Type Override** setting of the **Fixed-Point Settings** interface.

---

**Output data type mode**

Specify the output data type to be the same as the input, or inherit the data type by backpropagation. 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 for the **Output data type mode** parameter. Choose a built-in data type from the drop-down list.

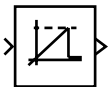
<b>Characteristics</b>	Sample Time	Constant
	Dimensionalized	Yes

# Wrap To Zero

**Purpose** Set output to zero if input is above threshold

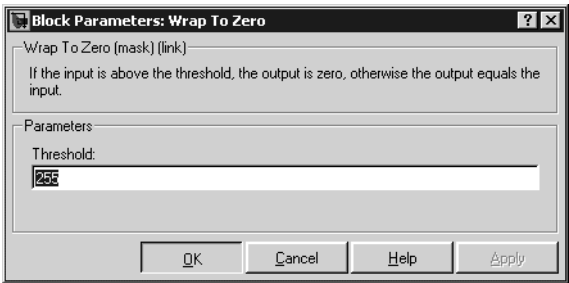
**Library** Discontinuities

**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 data type supported by Simulink, including fixed-point data types.

**Parameters and Dialog Box**



**Threshold** When the input exceeds the threshold, the output is set to zero.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Scalar Expansion	Yes

**Purpose** Display an X-Y plot of signals using a MATLAB figure window

**Library** Sinks

**Description** The XY Graph block displays an X-Y plot of its inputs in a MATLAB figure window.



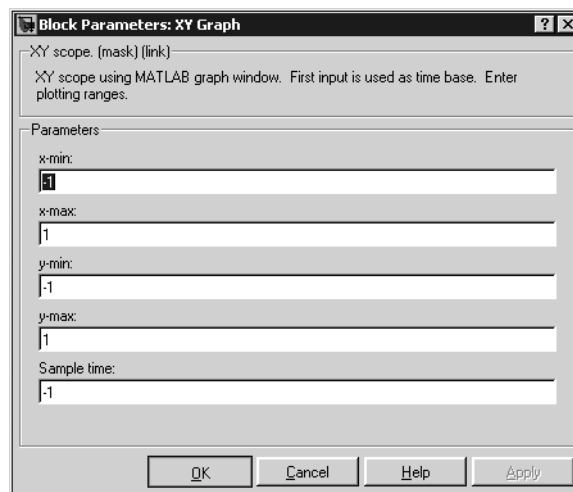
The block has two scalar inputs. The block plots data in the first input (the  $x$  direction) against data in the second input (the  $y$  direction). This block is useful for examining limit cycles and other two-state data. Data outside the specified range is not displayed.

Simulink opens a figure window for each XY Graph block in the model at the start of the simulation.

For a demo that illustrates the use of the XY Graph block, enter `lorenz` in the command window.

**Data Type Support** The XY Graph block accepts real signals of type `double`.

**Dialog box**



**x-min**

The minimum  $x$ -axis value. The default is `-1`.

# XY Graph

---

**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**

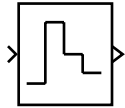
Specify the time interval between samples. To inherit the sample time, set this parameter to -1. See “Specifying Sample Time” in the online documentation for more information.

<b>Characteristics</b>	Sample Time	Specified in the <b>Sample time</b> parameter
	States	0

**Purpose** Implement a zero-order hold of one sample period

**Library** Discrete

**Description** The Zero-Order Hold block samples and holds its input for the specified sample period. 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 held for the same sample period.



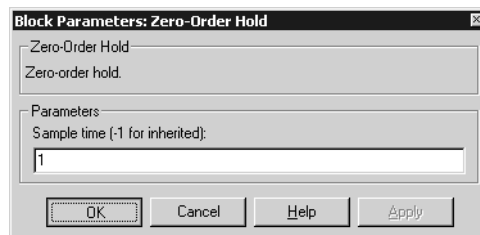
You specify the time between samples with the **Sample time** parameter. A setting of -1 means the **Sample time** is inherited.

This block provides a mechanism for discretizing one or more signals in time, or resampling the signal at a different rate. If your model contains multirate transitions, you must add Zero-Order Hold blocks between the fast-to-slow transitions. The sample rate of the Zero-Order Hold must be set to that of the slower block. For slow-to-fast transitions, use the Unit Delay block. For more information about multirate transitions, refer to the Simulink or the Real-Time Workshop documentation.

**Data Type Support** The Zero-Order Hold block accepts real or complex signals of any data type supported by Simulink, including fixed-point data types.

For a discussion on the data types supported by Simulink, refer to “Data Types Supported by Simulink” in the Using 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 “Specifying Sample Time” in the online documentation for more information.

# Zero-Order Hold

---

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Specified in the <b>Sample time</b> parameter
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No



**Purpose** Implement a transfer function specified in terms of poles and zeros

**Library** Continuous

**Description** The Zero-Pole block implements a system with the specified zeros, poles, and gain in terms of the Laplace operator  $s$ .

A transfer function can be expressed in factored or zero-pole-gain form, which, for a single-input single-output system in MATLAB, is

$$H(s) = K \frac{Z(s)}{P(s)} = K \frac{(s-Z(1))(s-Z(2))\dots(s-Z(m))}{(s-P(1))(s-P(2))\dots(s-P(n))}$$

where  $Z$  represents the zeros vector,  $P$  the poles vector, and  $K$  the gain.  $Z$  can be a vector or matrix,  $P$  must be a vector,  $K$  can be a scalar or vector whose length equals the number of rows in  $Z$ . 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.

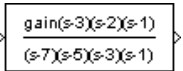
Block input and output widths are equal to the number of rows in the zeros matrix.

**Transfer Function Display on Block**

The Zero-Pole block displays the transfer function depending on how the parameters are specified:

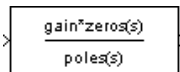
- If each is specified as an expression or a vector, the icon shows the transfer function with the specified zeros, poles, and gain. If you specify a variable in parentheses, the variable is evaluated.

For example, if you specify **Zeros** as [ 3, 2, 1 ], **Poles** as (poles), where poles is defined in the workspace as [ 7, 5, 3, 1 ], and **Gain** as gain, the icon looks like this:



# Zero-Pole

- If each is specified as a variable, the icon 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 icon looks like this.



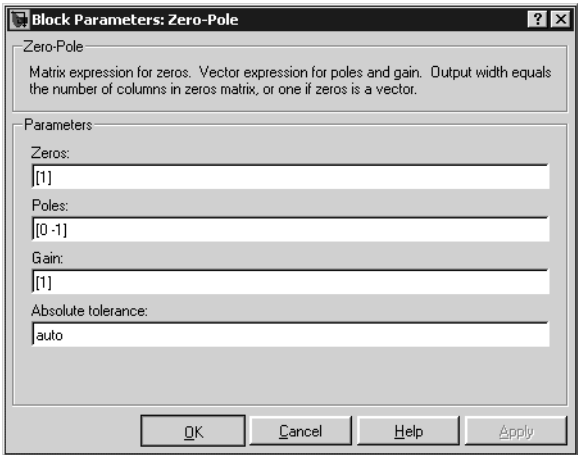
## Specifying the Absolute Tolerance for the Block's States

By default, Simulink uses the absolute tolerance value specified in the **Configuration Parameters** dialog box (see “Error Tolerances”) to solve the states of the Zero-Pole block. If this value does not provide sufficient error control, specify a more appropriate value in the **Absolute tolerance** field of the Zero-Pole block’s dialog box. The value that you specify is used to solve all the block’s states.

## Data Type Support

The Zero-Pole block accepts 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 -1 ].

Gain

The vector of gains. The default is [ 1 ].

Absolute tolerance

Absolute tolerance used to solve the block’s states. You can enter auto or a numeric value. If you enter auto, Simulink determines the absolute tolerance (see “Error Tolerances”). If you enter a numeric value, Simulink uses the specified value to solve the block’s states. Note that a numeric value overrides the setting for the absolute tolerance in the **Configuration Parameters** dialog box.

Characteristics	Direct Feedthrough	Only if the lengths of the <b>Poles</b> and <b>Zeros</b> parameters are equal
	Sample Time	Continuous
	Scalar Expansion	No
	States	Length of <b>Poles</b> vector
	Dimensionalized	No
	Zero Crossing	No



# Linearization and Trimming Commands

---

This section describes commands that you can use to linearize or trim a Simulink model. See “Analyzing Simulation Results” for more information on these commands.

## Linearization and Trimming Commands

linmod, dlinmod, linmod2  
trim

**Purpose** Extract the continuous- or discrete-time linear state-space model of a system around an operating point

**Syntax**

```
argout = linmod('sys');  
argout = linmod('sys',x,u);  
argout = linmod('sys', x, u, para);  
argout = linmod('sys', x, u, 'v5', para);  
argout = linmod('sys', x, u, 'v5', para, xpert, upert);  
  
argout = dlinmod('sys', x, u);  
argout = dlinmod('sys',Ts, x, u, 'v5', para);  
argout = dlinmod('sys', x, u, 'v5', para, xpert, upert);  
  
argout = linmod2('sys', x, u, para);
```

**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.
Ts	Sample time of the discrete-time linearized model
'v5'	An optional argument that invokes the perturbation algorithm created prior to MATLAB 5.3.
para	A 3-element vector of optional arguments: <ul style="list-style-type: none"><li>• 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.</li><li>• para(2) — Linearization time. For models that are functions of time, this parameter may be set with a nonnegative value of t at which the linear model is to be obtained. The default value is 0.</li><li>• 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.</li></ul>

# linmod, dlinmod, linmod2

---

xpert and upert	The perturbation values used to perform the perturbation of all the states and inputs of the model. The default values are $\text{xpert} = \text{para}(1) + 1\text{e-}3*\text{para}(1)*\text{abs}(x)$ $\text{upert} = \text{para}(1) + 1\text{e-}3*\text{para}(1)*\text{abs}(u)$
argout	linmod, dlinmod, and linmod2 all return state-space, 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: <ul style="list-style-type: none"><li>• <code>[A,B,C,D] = linmod('sys', x, u)</code> 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.</li><li>• <code>[num, den] = linmod('sys', x, u)</code> returns the linearized model in transfer function form.</li><li>• <code>sys_struc = linmod('sys', x, u)</code> returns a structure that contains the linearized model, including state names, input and output names, and information about the operating point.</li></ul>

## 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.

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 exact linearization for most blocks, and should be more accurate in most cases. 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  $T_s$  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  $T_s$  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  $T_s$ , provided that

- $T_s$  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  $A_d$  provides an indication of the stability of the system. The system is stable if  $T_s > 0$  and the eigenvalues are within the unit circle, as determined by this statement:

```
all(abs(eig(Ad))) < 1
```

Likewise, the system is stable if  $T_s = 0$  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  $A_d$  and  $B_d$  matrices, which can be complex. The eigenvalues of the  $A_d$  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.

## linmod, dlinmod, linmod2

---

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”.

**Purpose** Find a trim point of a 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] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options,t)
[x,u,y,dx,options] = trim('sys',...)
```

**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] = 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  $[x-x_0, u, y]$ . 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  $\text{abs}(dx-0)$ . You can obtain `x0` using this command.

```
[sizes,x0,xstr] = sys([],[],[],0)
```

`[x,u,y] = 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])
```

The command

```
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)])
```

Use the syntax

```
[x,u,y,dx] = trim('sys', x0, u0, y0, ix, iu, iy, dx0, idx)
```

to find 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.

The optional `options` argument is 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.

Five of the optimization array elements are particularly useful for finding trim points. The following table describes how each element affects the search for a trim point.

No.	Default	Description
1	0	Specifies display options. 0 specifies no display; 1 specifies tabular output; -1 suppresses warning messages.
2	0.0001	Precision the computed trim point must attain to terminate the search.
3	0.0001	Precision the trim search goal function must attain to terminate the search.

No.	Default	Description (Continued)
4	0.0001	Precision the state derivatives must attain to terminate the search.
10	N/A	Returns the number of iterations used to find a trim point.

See the Optimization Toolbox User’s Guide for a detailed description of the options array.

Examples

Consider a linear state-space model

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

The *A*, *B*, *C*, and *D* matrices are as follows in a system called sys.

$$\begin{aligned}A &= \begin{bmatrix} -0.09 & -0.01 & 1 & 0 \end{bmatrix}; \\ B &= \begin{bmatrix} 0 & -7 & 0 & -2 \end{bmatrix}; \\ C &= \begin{bmatrix} 0 & 2 & 1 & -5 \end{bmatrix}; \\ D &= \begin{bmatrix} -3 & 0 & 1 & 0 \end{bmatrix};\end{aligned}$$

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

```

# trim

---

## 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 documentation for the Optimization Toolbox for a description of this algorithm.



# Model Construction Commands

---

The following sections describe commands that you can use in programs that create or modify models.

Task-Oriented List of Commands  
(p. 3-2)

List of commands arranged by tasks to be performed

Specifying Parameters and Object  
Paths (p. 3-4)

How to specify parameters and object paths required by  
model construction commands

## Task-Oriented List of Commands

This table indicates the tasks performed by the commands described in this chapter. The reference section of this chapter lists the commands in alphabetical order.

Task	Command
Create a new Simulink system.	<code>new_system</code>
Open an existing system.	<code>open_system</code>
Invisibly load a model into memory.	<code>load_system</code>
Open or close the Library Browser.	<code>simulink</code>
Change MATLAB character encoding to be compatible with model character encoding.	<code>slCharacterEncoding</code>
Close a system window.	<code>close_system</code> , <code>bdclose</code>
Save a system.	<code>save_system</code>
Find a system, block, line, or annotation.	<code>find_system</code>
Find model references.	<code>find mdlrefs</code>
Get information about the library links in a model.	<code>libinfo</code>
Add a new block to a system.	<code>add_block</code>
Delete a block from a system.	<code>delete_block</code>
Replace a block in a system.	<code>replace_block</code>
Update obsolete versions of blocks	<code>slupdate</code>
Terminate unconnected ports in a system.	<code>addterms</code>
Add a line to a system.	<code>add_line</code>
Delete a line from a system.	<code>delete_line</code>
Add a parameter to a system.	<code>add_param</code>

Task	Command
Get a parameter value.	get_param
Set parameter values.	set_param
Delete a system parameter.	delete_param
Get the pathname of the current block.	gcb
Get the pathname of the current system.	gcs
Get the handle of the current block.	gcbh
Get the name of the root-level system.	bdroot
Display a graph of mode reference dependencies.	view_mdhrefs
Open the Simulink block library.	simulink
Discretize a model.	sldiscmdl
Open the Model Discretizer GUI.	slmdliscui

## Specifying Parameters and Object Paths

This section explains how to specify parameters and object paths required by model construction commands.

### How to Specify Parameters for the Commands

The commands described in this chapter require that you specify arguments that describe a system, block, or block parameter. Chapter 9, “Model and Block Parameters,” provides comprehensive tables of model and block parameters.

### How to Specify a Path for a Simulink Object

Many of the commands described in this chapter require that you identify a Simulink system or block. Identify systems and blocks by specifying their paths:

- To identify a system, specify its name, which is the name of the file that contains the system description, without the mdl extension.  
system
- To identify a subsystem, specify the system and the hierarchy of subsystems in which the subsystem resides.  
system/subsystem<sub>1</sub>/.../subsystem
- To identify a block, specify the path of the system that contains the block and specify the block name.  
system/subsystem<sub>1</sub>/.../subsystem/block

If the block name includes a newline or carriage return, specify the block name as a string vector and use `sprintf('\n')` as the newline character. For example, these lines assign the newline character to `cr`, then get the value for the Signal Generator block’s **Amplitude** parameter.

```
cr = sprintf('\n');
get_param(['untitled/Signal',cr,'Generator'],'Amplitude')
ans =
    1
```

If the block name includes a slash character (/), you repeat the slash when you specify the block name. For example, to get the value of the `Location` parameter for the block named `Signal/Noise` in the `mymodel` system, enter

```
get_param('mymodel/Signal//Noise','Location')
```

## Model Construction Commands

```
add_block  
add_line  
add_param  
addterms  
bdclose  
bdroot  
close_system  
delete_block  
delete_line  
delete_param  
find_mdhrefs  
find_system  
gcb  
gcbh  
gcs  
get_param  
libinfo  
load_system  
new_system  
open_system  
replace_block  
save_system  
set_param  
simulink  
signalbuilder  
slCharacterEncoding  
sldiscmdl  
slmdliscui  
slupdate  
view_mdhrefs
```

# add\_block

---

**Purpose** Add a block to a Simulink system

**Syntax**

```
add_block('src', 'dest')
add_block('src', 'dest', 'parameter1', value1, ...)
```

**Description** `add_block('src', 'dest')` copies the block with the full pathname 'src' to a new block with the full path name 'dest'. The block parameters of the new block are identical to those of the original. The name 'built-in' can be used as a source system name for all Simulink built-in blocks (blocks available in Simulink block libraries that are not masked blocks).

`add_block('src', 'dest_obj', 'parameter1', value1, ...)` creates a copy as above, in which the named parameters have the specified values. Any additional arguments must occur in parameter/value pairs.

**Examples** This command copies the Scope block from the Sinks subsystem of the simulink system to a block named Scope1 in the timing subsystem of the engine system.

```
add_block('simulink/Sinks/Scope', 'engine/timing/Scope1')
```

This command creates a new subsystem named controller in the F14 system.

```
add_block('built-in/SubSystem', 'F14/controller')
```

This command copies the built-in Gain block to a block named Volume in the mymodel system and assigns the **Gain** parameter a value of 4.

```
add_block('built-in/Gain', 'mymodel/Volume', 'Gain', '4')
```

**See Also** `delete_block`, `set_param`

# add\_line

---

## Purpose

Add a line to a Simulink system

## Syntax

```
h = add_line('sys','oport','iport')
h = add_line('sys','oport','iport', 'autorouting','on')
h = add_line('sys', points)
```

## Description

The `add_line` command adds a line to the specified system and returns a handle to the new line. You can define the line in two ways:

- By naming the block ports that are to be connected by the line
- By specifying the location of the points that define the line segments

`add_line('sys', 'oport', 'iport')` adds a straight line to a system from the specified block output port 'oport' to the specified block input port 'iport'. 'oport' and 'iport' are strings consisting of a block name and a port identifier in the form 'block/port'. Most block ports are identified by numbering the ports from top to bottom or from left to right, such as 'Gain/1' or 'Sum/2'. Enable, Trigger, State, and Action ports are identified by name, such as 'subsystem\_name/Enable', 'subsystem\_name/Trigger', 'Integrator/State', or 'if\_action\_subsystem\_name/Ifaction'.

`add_line('sys','oport','iport', 'autorouting','on')` works like `add_line('sys','oport','iport')` except that it routes the line around intervening blocks. The default value for autorouting is 'off'.

`add_line(system, points)` adds a segmented line to a system. Each row of the `points` array specifies the  $x$  and  $y$  coordinates of a point on a line segment. The origin is the top left corner of the window. The signal flows from the point defined in the first row to the point defined in the last row. If the start of the new line is close to the output of an existing block or line, a connection is made. Likewise, if the end of the line is close to an existing input, a connection is made.



## Examples

This command adds a line to the `mymodel` system connecting the output of the Sine Wave block to the first input of the Mux block.

```
add_line('mymodel','Sine Wave/1','Mux/1')
```

This command adds a line to the `mymodel` system extending from (20,55) to (40,10) to (60,60).

```
add_line('mymodel',[20 55; 40 10; 60 60])
```

## See Also

`delete_line`

# add\_param

---

<b>Purpose</b>	Add a parameter to a Simulink system
<b>Syntax</b>	<code>add_param('sys','parameter1',value1,'parameter2',value2,...)</code>
<b>Description</b>	The <code>add_param</code> 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, <code>set_param</code> and <code>get_param</code> can be used on the new parameters as if they were standard Simulink parameters.
<b>Examples</b>	<p>This command</p> <pre>add_param('vdp','Param1','Value1','Param2','Value2')</pre> <p>adds the parameters <code>Param1</code> and <code>Param2</code> with values <code>'Value1'</code> and <code>'Value2'</code> to the <code>vdp</code> system.</p>
<b>See Also</b>	<code>delete_param</code> , <code>get_param</code> , <code>set_param</code>

<b>Purpose</b>	Add terminators to unconnected ports in a model
<b>Syntax</b>	<code>addterms('sys')</code>
<b>Description</b>	<code>addterms('sys')</code> adds Terminator and Ground blocks to the unconnected ports in the Simulink block diagram <code>sys</code> .
<b>See Also</b>	<code>slupdate</code>

# bdclose

---

<b>Purpose</b>	Close any or all Simulink system windows unconditionally
<b>Syntax</b>	<code>bdclose</code> <code>bdclose('sys')</code> <code>bdclose('all')</code>
<b>Description</b>	<p><code>bdclose</code> 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.</p> <p><code>bdclose('sys')</code> closes the specified system window.</p> <p><code>bdclose('all')</code> closes all system windows.</p>
<b>Examples</b>	<p>This command closes the vdp system.</p> <pre>bdclose('vdp')</pre>
<b>See Also</b>	<code>close_system</code> , <code>new_system</code> , <code>open_system</code> , <code>save_system</code>

<b>Purpose</b>	Return the name of the top-level Simulink system
<b>Syntax</b>	<code>bdroot</code> <code>bdroot('obj')</code>
<b>Description</b>	<p><code>bdroot</code> with no arguments returns the top-level system name.</p> <p><code>bdroot('obj')</code>, where 'obj' is a system or block pathname, returns the name of the top-level system containing the specified object name.</p>
<b>Examples</b>	<p>This command returns the name of the top-level system that contains the current block.</p> <pre>bdroot(gcb)</pre>
<b>See Also</b>	<code>find_system</code> , <code>gcb</code>

# close\_system

---

**Purpose** Close a Simulink system window or a block dialog box

**Syntax**

```
close_system
close_system('sys')
close_system('sys', saveflag)
close_system('sys', 'newname')
close_system('blk')
```

**Description** `close_system` with no arguments closes the current system or subsystem window. If the current system is the top-level system and it has been modified, `close_system` asks if the changed system should be saved to a file before removing the system from memory. The current system is defined in the description of the `gcs` command.

`close_system('sys')` closes the specified system or subsystem window.

`close_system('sys', saveflag)` closes the specified top-level system window and removes it from memory:

- If `saveflag` is 0, the system is not saved.
- If `saveflag` is 1, the system is saved with its current name.

`close_system('sys', 'newname')` saves the specified top-level system to a file with the specified new name, then closes the system.

`close_system('blk')`, where `'blk'` is a full block pathname, closes the dialog box associated with the specified block or calls the block's `CloseFcn` callback parameter if one is defined. Any additional arguments are ignored.

**Examples** This command closes the current system.

```
close_system
```

This command closes the vdp system.

```
close_system('vdp')
```

This command saves the engine system with its current name, then closes it.

```
close_system('engine', 1)
```

This command saves the mymdl12 system under the new name testsys, then closes it.

```
close_system('mymdl12', 'testsys')
```

This command closes the dialog box of the Unit Delay block in the Combustion subsystem of the engine system.

```
close_system('engine/Combustion/Unit Delay')
```

### See Also

bdclose, gcs, new\_system, open\_system, save\_system

# delete\_block

---

<b>Purpose</b>	Delete a block from a Simulink system
<b>Syntax</b>	<code>delete_block('blk')</code>
<b>Description</b>	<code>delete_block('blk')</code> , where 'blk' is a full block pathname, deletes the specified block from a system.
<b>Example</b>	This command removes the Out1 block from the vdp system.  <code>delete_block('vdp/Out1')</code>
<b>See Also</b>	<code>add_block</code>



<b>Purpose</b>	Delete a line from a Simulink system
<b>Syntax</b>	<code>delete_line('sys', 'oport', 'iport')</code>
<b>Description</b>	<p><code>delete_line('sys', 'oport', 'iport')</code> 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'.</p> <p><code>delete_line('sys', [x y])</code> deletes one of the lines in the system that contains the specified point (x,y), if any such line exists.</p>
<b>Example</b>	<p>This command removes the line from the mymodel system connecting the Sum block to the second input of the Mux block.</p> <pre>delete_line('mymodel', 'Sum/1', 'Mux/2')</pre>
<b>See Also</b>	<code>add_line</code>

# delete\_param

---

<b>Purpose</b>	Delete a system parameter added via the add_param command
<b>Syntax</b>	<code>delete_param('sys','parameter1','parameter2',...)</code>
<b>Description</b>	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.
<b>Examples</b>	<p>The following example</p> <pre>delete_param('vdp','Param1') add_param('vdp','Param1','Value1','Param2','Value2')</pre> <p>adds the parameters Param1 and Param2 to the vdp system, then deletes Param1 from the system.</p>
<b>See Also</b>	<code>add_param</code>

<b>Purpose</b>	Find the Model blocks in a model and the models that the Model blocks reference.
<b>Syntax</b>	<pre>[refMdls, mdlBlks] = find_mdlsrefs(modelName) [refMdls, mdlBlks] = find_mdlsrefs(modelName, true) [refMdls, mdlBlks] = find_mdlsrefs(modelName, false)</pre>
<b>Description</b>	<p>[refMdls, mdlBlks] = find_mdlsrefs(modelName) or find_mdlsrefs(modelName, true) finds all Model blocks contained by and models referenced by <b>modelName</b> directly or indirectly. The commands output arguments are</p> <ul style="list-style-type: none"><li>• <b>refMdls</b> List of models. The last element in the list is <b>modelName</b>. The other elements are the names of models referenced by <b>modelName</b>.</li><li>• <b>mdlBlks</b> Names of Model blocks contained by <b>modelName</b> and the models that it references directly or indirectly.</li></ul> <p>[refMdls, mdlBlks] = find_mdlsrefs(modelName, false) finds only the Model blocks and models directly referenced by <b>modelName</b>.</p>
<b>Example</b>	<p>Open the mdlref_basic demo. Then execute</p> <pre>&gt;&gt; [r, b] = find_mdlsrefs('mdlref_basic')  r =     'mdlref_counter'     'mdlref_basic'  b =     'mdlref_basic/CounterA'     'mdlref_basic/CounterB'     'mdlref_basic/CounterC'</pre>
<b>See Also</b>	view_mdlsrefs

# 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 having the specified parameter values v1, v2, etc. sys can be a pathname (or cell array of pathnames), a handle (or vector of handles), or omitted. 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 9, “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.

Name	Value Type	Description
	'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.

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')
```

# find\_system

---

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\_subsystem 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'.
\$	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-zA-Z0-9].) For example, '^\\w' matches 'mu' but not '&mu'.
\d	Matches any digit (shorthand for [0-9]). For example, '\\d+' matches any integer.

# find\_system

Expression	Usage
\D	Matches any nondigit (shorthand for [ ^0-9]).
\s	Matches a white space (shorthand for [ \t\r\n\f]).
\S	Matches a non-white-space (shorthand for [ ^ \t\r\n\f]).
\<WORD\>	Matches WORD exactly, where WORD is a string of characters separated by white space from other words. For example, '\<to\>' matches 'to' but not 'today'.

To use regular expressions to search Simulink systems, specify the 'regexp' search constraint as 'on' in a find\_system command and use a regular expression anywhere you would use an ordinary search value string.

For example, the following command finds all the inport and outport blocks in the clutch model demo provided with Simulink.

```
find_system('clutch', 'regexp', 'on', 'blocktype', 'port')
```

## See Also

get\_param, set\_param



**Purpose** Get the pathname of the current block

**Syntax** `gcb`  
`gcb('sys')`

**Description** `gcb` returns the full block path name of the current block in the current system.

`gcb('sys')` returns the full block path name 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 on.
- 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`

# gcbh

---

**Purpose** Get the handle of the current block

**Syntax** gcbh

**Description** gcbh returns the handle of the current block in the current system.  
You can use this command to identify or address blocks that have no parent system. The command should be most useful to blockset authors.

**Examples** This command returns the handle of the most recently selected block.

```
gcbh  
  
ans =  
  
281.0001
```

**See Also** gcb

<b>Purpose</b>	Get the pathname of the current system
<b>Syntax</b>	<code>gcs</code>
<b>Description</b>	<p><code>gcs</code> returns the full pathname of the current system.</p> <p>The current system is one of these:</p> <ul style="list-style-type: none"><li>• During editing, the current system is the system or subsystem most recently clicked in.</li><li>• 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.</li><li>• During callbacks, the current system is the system containing any block whose callback routine is being executed.</li><li>• During evaluation of the <code>MaskInitialization</code> string, the current system is the system containing the block whose mask is being evaluated.</li></ul> <p>The current system is always the current model or a subsystem of the current model. Use <code>bdroot</code> to get the current model.</p>
<b>Examples</b>	<p>This example returns the path of the system that contains the most recently selected block.</p> <pre>gcs ans =     clutch/Locked</pre>
<b>See Also</b>	<code>gcb</code> , <code>bdroot</code>

# get\_param

---

**Purpose** Get system and block parameter values

**Syntax**

```
get_param('obj', 'parameter')  
get_param( { objects }, 'parameter')  
get_param(handle, 'parameter')  
get_param(0, 'parameter')  
get_param('obj', 'ObjectParameters')  
get_param('obj', 'DialogParameters')
```

**Description**

`get_param('obj', 'parameter')`, where 'obj' is a system or block path name, returns the value of the specified parameter. Case is ignored for parameter names.

`get_param( { objects }, 'parameter')` accepts a cell array of full path specifiers, enabling you to get the values of a parameter common to all objects specified in the cell array.

`get_param(handle, 'parameter')` returns the specified parameter of the object whose handle is handle.

`get_param(0, 'parameter')` returns the current value of a Simulink session parameter or the default value of a model or block parameter.

`get_param('obj', 'ObjectParameters')` returns a structure that describes obj's parameters. Each field of the returned structure corresponds to a particular parameter and has the parameter's name. For example, the Name field corresponds to the object's Name parameter. Each parameter field itself contains three fields, Name, Type, and Attributes, that specify the parameter's name (for example, "Gain"), data type (for example, string), and attributes (for example, read-only), respectively.

`get_param('obj', 'DialogParameters')` returns a cell array containing the names of the dialog parameters of the specified block.

Chapter 9, "Model and Block Parameters," contains lists of model and block parameters.

**Examples**

This command returns the value of the **Gain** parameter for the Inertia block in the Requisite Friction subsystem of the clutch system.

```
get_param('clutch/Requisite Friction/Inertia', 'Gain')
```

```
ans =
    1/(Iv+Ie)
```

These commands display the block types of all blocks in the `mx + b` system (the current system), described in “Masked Subsystem Example” in *Using Simulink*.

```
blks = find_system(gcs, 'Type', 'block');
listblks = get_param(blks, 'BlockType')

listblks =

    'SubSystem'
    'Inport'
    'Constant'
    'Gain'
    'Sum'
    'Outport'
```

This command returns the name of the currently selected block.

```
get_param(gcf, 'Name')
```

The following commands get the attributes of the currently selected block’s Name parameter.

```
p = get_param(gcf, 'ObjectParameters');
a = p.Name.Attributes

ans =

    'read-write'    'always-save'
```

The following command gets the dialog parameters of a Sine Wave block.

```
p = get_param('untitled/Sine Wave', 'DialogParameters')
p =

    'Amplitude'
    'Frequency'
    'Phase'
    'SampleTime'
```

## See Also

`find_system`, `set_param`

# libinfo

---

**Purpose** Get information about the library blocks referenced by a model.

**Syntax** `libdata = libinfo('sys')`

**Description** `libdata = libinfo('sys')` returns information about library blocks referenced by **sys** and all of the systems underneath it. The command returns an array of structures that describe each library block referenced by the model. Each structure has the following fields:

- **Block**  
Path of the link to the library block
- **Library**  
Name of the library containing the referenced block
- **ReferenceBlock**  
Path of the library block
- **LinkStatus**  
The value of the `LinkStatus` parameter for the link to the library block.

This command also accepts search constraints as additional arguments. For instance:

```
libdata=libinfo(Sys, 'FollowLinks', 'off')
```

See `find_system` for more information.

<b>Purpose</b>	Invisibly load a Simulink model.
<b>Syntax</b>	<code>load_system('sys')</code>
<b>Description</b>	<code>load_system('sys')</code> loads <b>sys</b> , where <b>sys</b> is the name of a Simulink model, into memory without making its model window visible.
<b>Examples</b>	<p>The command</p> <pre>load_system('vdp')</pre> <p>loads the vdp sample model into memory.</p>
<b>See Also</b>	<code>open_system</code> , <code>close_system</code>

# new\_system

---

**Purpose** Create an empty Simulink system

**Syntax**

```
new_system('sys')
new_system('sys', 'Model')
new_system('sys', 'Model', 'subsystem_path')
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.

`new_system('sys', 'Model', 'subsystem_path')` creates a new 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', '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 9, “Model and Block Parameters” for a list of the default parameter values for the new system.

**Example** This command creates a new system named 'mysys'.

```
new_system('mysys')
```

The commands

```
load_system('f14')
new_system('mycontroller', 'Model', 'f14/Controller')
```

creates 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`



<b>Purpose</b>	Open a Simulink system window or a block dialog box
<b>Syntax</b>	<pre>open_system('sys') open_system('blk') open_system('blk', 'force')</pre>
<b>Description</b>	<p><code>open_system('sys')</code> opens the specified system or subsystem window, where 'sys' is the name of a model on the MATLAB path, the fully qualified pathname of a model, or the relative pathname of a subsystem of an already open system (for example, engine/Combustion). On UNIX, the fully qualified pathname of a model can start with a tilde (~), signifying your home directory.</p> <p><code>open_system('blk')</code>, where 'blk' is a full block pathname, opens the dialog box associated with the specified block. If the block's <code>OpenFcn</code> callback parameter is defined, the routine is evaluated.</p> <p><code>open_system('blk', 'force')</code>, 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 <b>Look Under Mask</b> menu item.</p>
<b>Example</b>	<p>This command opens the controller system in its default screen location.</p> <pre>open_system('controller')</pre> <p>This command opens the block dialog box for the Gain block in the controller system.</p> <pre>open_system('controller/Gain')</pre>
<b>See Also</b>	<code>close_system</code> , <code>load_system</code> , <code>new_system</code> , <code>save_system</code>

# replace\_block

---

**Purpose** Replace blocks in a Simulink model

**Syntax** `replace_block('sys', 'blk1', 'blk2', 'noprompt')`  
`replace_block('sys', 'Parameter', 'value', 'blk', ...)`

**Description** `replace_block('sys', 'blk1', 'blk2')` replaces all blocks in 'sys' having the block or mask type 'blk1' with 'blk2'. If 'blk2' is a Simulink built-in block, only the block name is necessary. If 'blk' is in another system, its full block pathname is required. If 'noprompt' is omitted, Simulink displays a dialog box that asks you to select matching blocks before making the replacement. Specifying the 'noprompt' argument suppresses the dialog box from being displayed. If a return variable is specified, the paths of the replaced blocks are stored in that variable.

`replace_block('sys', 'Parameter', 'value', ..., 'blk')` replaces all blocks in 'sys' having the specified values for the specified parameters with 'blk'. You can specify any number of parameter name/value pairs.

---

**Note** Because it may be difficult to undo the changes this command makes, it is a good idea to save your system first.

---

**Example** This command replaces all Gain blocks in the f14 system with Integrator blocks and stores the paths of the replaced blocks in RepNames. Simulink lists the matching blocks in a dialog box before making the replacement.

```
RepNames = replace_block('f14', 'Gain', 'Integrator')
```

This command replaces all blocks in the Unlocked subsystem in the clutch system having a Gain of 'bv' with the Integrator block. Simulink displays a dialog box listing the matching blocks before making the replacement.

```
replace_block('clutch/Unlocked', 'Gain', 'bv', 'Integrator')
```

This command replaces the Gain blocks in the f14 system with Integrator blocks but does not display the dialog box.

```
replace_block('f14', 'Gain', 'Integrator', 'noprompt')
```

### See Also

`find_system`, `set_param`

# save\_system

---

**Purpose** Save a Simulink system

**Syntax**

```
save_system
save_system('sys')
save_system('sys', 'newname')
save_system('sys', 'newname', 'BreakLinks')
```

**Description**

`save_system` saves the current top-level system to a file with its current name.

`save_system('sys')` saves the specified top-level system to a file with its current name. The system must be open.

`save_system('sys', 'newname')` saves the specified top-level system to a file with the specified new name. The new name can be a file name, in which case Simulink saves the system in the working directory, or a fully qualified pathname. On UNIX, the fully qualified pathname can start with a tilde (~), signifying your home directory. The system to be saved must be open.

`save_system('sys', 'newname', 'BreakLinks')` saves the specified top-level system to a file with the specified new name, replacing links to library blocks with copies of the library blocks in the saved file.

**Example** This command saves the current system.

```
save_system
```

This command saves the vdp system.

```
save_system('vdp')
```

This command saves the vdp system to a file with the name 'myvdp'.

```
save_system('vdp', 'myvdp')
```

**See Also** `close_system`, `new_system`, `open_system`

<b>Purpose</b>	Set Simulink system and block parameters
<b>Syntax</b>	<code>set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)</code>
<b>Description</b>	<p><code>set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)</code>, where 'obj' is a system or block path or 0, sets the specified parameters to the specified values. Use 0 to set the default value of a parameter or the values of session parameters. Case is ignored for parameter names. Value strings are case sensitive. Any parameters that correspond to dialog box entries have string values. Model and block parameters are listed in “Model and Block Parameters” on page 9-1</p> <p>You can change block parameter values in the workspace during a simulation and update the block diagram with these changes. To do this, make the changes in the command window, then make the model window the active window, then choose <b>Update Diagram</b> from the <b>Edit</b> menu.</p>

---

**Note** Most block parameter values must be specified as strings. Two exceptions are the Position and UserData parameters, common to all blocks.

---

<b>Examples</b>	<p>This command sets the Solver and StopTime parameters of the vdp system.</p> <pre>set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000')</pre> <p>This command sets the <b>Gain</b> parameter of block Mu in the vdp system to 1000 (stiff).</p> <pre>set_param('vdp/Mu', 'Gain', '1000')</pre> <p>This command sets the position of the Fcn block in the vdp system.</p> <pre>set_param('vdp/Fcn', 'Position', [50 100 110 120])</pre> <p>This command sets the Zeros and Poles parameters for the Zero-Pole block in the mymodel system.</p> <pre>set_param('mymodel/Zero-Pole', 'Zeros', '[2 4]', 'Poles', '[1 2 3]')</pre>
-----------------	--

## set\_param

---

This command sets the **Gain** parameter for a block in a masked subsystem. The variable `k` is associated with the **Gain** parameter.

```
set_param('mymodel/Subsystem', 'k', '10')
```

This command sets the `OpenFcn` callback parameter of the block named `Compute` in system `mymodel`. The function `'my_open_fcn'` executes when the user double-clicks on the `Compute` block (see “Using Callback Routines”).

```
set_param('mymodel/Compute', 'OpenFcn', 'my_open_fcn')
```

### See Also

`get_param`, `find_system`

**Purpose** Create and access Signal Builder blocks.

**Syntax**

```
[time, data] = signalbuilder(block)
[time, data, siglabels] = signalbuilder(block)
[time, data, siglabels, grouplabels] = signalbuilder(block)
block = signalbuilder([], 'create', time, data, siglabels,
    grouplabels)
block = signalbuilder(block, 'append', time, data, siglabels,
    grouplabels)
[time, data] = signalbuilder(block, 'get', signal, group)
signalbuilder(block, 'set', signal, group, time, data)
```

**Description** [time, data] = signalbuilder(block) returns the time (x coordinate) and amplitude (y coordinate) data of the Signal Builder block.

The output arguments, time and data, take different formats depending on the block configuration:

Configuration	Time/Data Format
1 signal, 1 group	Row vector of break points
>1 signal, 1 group	Column cell vector where each element corresponds to a separate signal and contains a row vector of breakpoints.
1 signal, >1 group	Row cell vector where each element corresponds to a separate group and contains a row vector of breakpoints
>1 signal, >1 group	Cell matrix where each element (i,j) corresponds to signal i and group j

[time, data, siglabels] = signalbuilder(block) returns the signal labels, siglabels, in a string or a cell array of strings.

[time, data, siglabels, grouplabels] = signalbuilder(block) returns the group labels, grouplabels, in a string or a cell array of strings.

**block** = `signalbuilder([], 'create', time, data, siglabels, grouplabels)` creates a Signal Builder block in a new Simulink model using the specified values. If **data** is a cell array and **time** is a vector, the **time** values are duplicated for each element of **data**. Each vector in **time** and **data** must be the same length and have at least two elements. If **time** is a cell array, all elements in a column must have the same initial and final value. Signal labels, **siglabels**, and group labels, **grouplabels**, can be omitted to use default values. The function returns the path to the new block, **BLOCK**.

**block** = `signalbuilder(block, 'append', time, data, siglabels, grouplabels)` appends new groups to the Signal Builder block, **block**. The **time** and **data** arguments must have the same number of signals as the existing block.

## Get/Set Methods for Specific Signals and Groups

`[time, data] = signalbuilder(block, 'get', signal, group)` gets the time and data values for the specified signal(s) and group(s). The **signal** argument can be the name of a signal, a scalar index of a signal, or an array of signal indices. The **group** argument can be a group label, a scalar index, or an array of indices.

`signalbuilder(block, 'set', signal, group, time, data)` sets the time and data values for the specified signal(s) and group(s). Use empty values of **time** and **data** to remove groups and signals.



**Purpose** Open the Simulink block library

**Syntax**

```
simulink  
simulink('open')  
simulink('close')
```

**Description** On Microsoft Windows, the `simulink` or `simulink('open')` opens the Simulink block library browser. On UNIX, the command opens the Simulink library window. `simulink('close')` closes the library window.

# slCharacterEncoding

---

**Purpose** Change the MATLAB character set encoding.

**Syntax** `slCharacterEncoding()`  
`slCharacterEncoding(encoding)`

**Description** This command allows you to change the current MATLAB character set encoding to be compatible with the encoding of a model that you want to open. `slCharacterEncoding()` returns the current MATLAB character set encoding. `slCharacterEncoding(encoding)` change the MATLAB character set encoding to the specified encoding. Valid values are

- 'US-ASCII'
- 'UTF-8'
- 'Shift\_JIS'
- 'ISO-8859-1'

---

**Note** You must close all open models or libraries before changing the MATLAB character set encoding except when changing from 'US-ASCII' to another encoding.

---

**Purpose** Discretize a Simulink model containing continuous blocks

**Syntax**

```
sldiscmdl('sys',sampletime)
sldiscmdl('sys',sampletime,'method')
sldiscmdl('sys',sampletime,{options})
sldiscmdl('sys',sampletime,'method',cf)
sldiscmdl('sys',sampletime,'method',{options})
sldiscmdl('sys',sampletime,'method',cf,{options})
```

**Description** sldiscmdl('sys',sampletime) discretizes the model specified by 'sys' and sampletime. You can enter a sample time and an offset as a 2-element vector for sampletime. The units for sampletime are seconds.

sldiscmdl('sys',sampletime,'method') discretizes the model with the transform method specified by 'method'. Available values for 'method' are shown below:

Value	Description
'zoh'	Zero-order hold on the inputs
'foh'	First-order hold on the inputs
'tustin'	Bilinear (Tustin) approximation
'prewarp'	Tustin approximation with frequency prewarping
'matched'	Matched pole-zero method (for SISO systems only)

sldiscmdl('sys',sampletime,{options}) discretizes the model with the criteria specified by {options}, where {options} is a cell array containing the following string elements:

```
{'target','ReplaceWith','PutInto','prompt'}
```

Available values for 'target' are shown below:

Value	Description
'all'	Discretize all continuous blocks
'selected'	Discretize selected blocks only
'<full path name of block>'	Discretize specified block

Available values for 'ReplaceWith' are shown below:

Value	Description
'parammask'	Create discrete blocks whose parameters are retained from the corresponding continuous block
'hardcoded'	Create discrete blocks whose parameters are “hard_coded” values placed directly into the block’s dialog

Available values for 'PutInto' are shown below:

Value	Description
'current'	Apply discretization to current model
'configurable'	Create discretization candidate in a configurable subsystem
'untitled'	Create discretization in a new untitled window
'copy'	Create discretization in copy of the original model

Available values for 'prompt' are shown below:

Value	Description
'on'	Show the discretization information
'off'	Do not show the discretization information

sldiscmdl('sys',sampletime,'method',cf) discretizes the model with the critical frequency specified by cf. The units for cf are Hz. This is only used when the transform method is 'prewarp'.

Examples

This command discretizes all of the continuous blocks in the f14 model with a 1 second sample time.

```
sldiscmdl('f14',1.0)
```

This command discretizes the Controller subsystem in the f14 model using a first-order hold transform method with a 1 second sample time and a 0.1 second sample time offset. The discretized block has “hard-coded” parameters that are placed directly into the block’s dialog box.

```
sldiscmdl('f14',[1.0 0.1],'foh',{'f14/Controller',...
'hardcoded','copy','on'})
```

This command discretizes the Controller subsystem in the f14 model using a zero-order hold transform method with a 1 second sample time and a 0.1 second sample time offset. It returns to the command window a cell array for the original continuous blocks in the system and a cell array for the discretized blocks in the system.

```
[a, b] = sldiscmdl('f14',[1.0 0.1],'zoh', {'f14/Controller',...
'hardcoded', 'copy', 'on'})
a =

    [1x43 char]    [1x37 char]    [1x53 char]    [1x30 char]

b =
```

```
[1x43 char]    [1x37 char]    [1x53 char]    [1x30 char]
```

You can index into the cell arrays to get the new names of the discretized blocks and the original names of the continuous blocks.

For example, this command returns the name of the second discretized block.

```
b{2}
```

```
ans =
```

```
f14_disc_copy/Controller/Pitch Rate  
Lead Filter
```

<b>Purpose</b>	Open the Model Discretizer GUI
<b>Syntax</b>	<code>slmdliscui('name')</code>
<b>Description</b>	<code>slmdliscui('name')</code> Opens the Model Discretizer with the library or model specified by 'name'.
<b>Examples</b>	<p>This command opens the Model Discretizer with the f14 model.</p> <pre>slmdliscui('f14')</pre> <p>This command opens the Model Discretizer with the library named Test.</p> <pre>slmdliscui('Test')</pre>

# slupdate

**Purpose** Replace obsolete versions of blocks in a model with the latest versions

**Syntax** `slupdate('sys')`  
`slupdate('sys', prompt)`

**Description** `slupdate('sys')` replaces versions of blocks in the model `sys` made obsolete by Simulink 5 with the latest versions. `slupdate('sys', prompt)` specifies whether to prompt you before replacing a block. If `prompt` equals 1, the command prompts you before replacing the block. The prompt asks whether you want to replace the block. Valid responses are

- y  
Replace the block (the default)
- n  
Do not replace the block
- a  
Replace this and all subsequent obsolete blocks without further prompting

If `prompt` equals 0, the command replaces all obsolete blocks without prompting you.

**Note** The model to be updated must have been created by a version of Simulink earlier than Simulink 5 and must be open when you call `slupdate`.

Blocks updated by this command include:

Block	Comments
Pulse Generator	New implementation
Hit Crossing	S-function replaced by built-in implementation
Memory	S-function replaced by built-in implementation
Quantizer	S-function replaced by built-in implementation
Graph scope	Replaced by much-improved built-in Scope



Block	Comments
2-D Table Lookup	S-function replaced by built-in implementation
Elementary Math	Replaced by either the Trigonometry, Rounding, or Math block
To Workspace	Three-element version of Maximum rows parameter is separated into individual fields
Outport	Replace initial output of width []

In addition to replacing obsolete blocks, slupdate

- Adds terminator blocks to any unconnected input and output ports, i.e., Ground and Terminator blocks, respectively
- Converts blocks to links in the appropriate block libraries
- Replaces any masked built-in block that is not a subsystem or S-functions with a masked subsystems containing the built-in block

# view\_mdhrefs

---

**Purpose** Display a graph of model reference dependencies

**Syntax** `view_mdhrefs('model_name')`

**Description** `view_mdhrefs('model_name')` displays a graph of model reference dependencies for the model specified by **model\_name**. The nodes in the graph represent Simulink models. The directed lines indicate model dependencies. For more information, see the `mdlref_depgraph` demo.

**See Also** `find_mdhrefs`

# Simulation Commands

---

The following section describes commands that you can use to run simulations manually.

# Task-Oriented List of Commands

This table indicates the tasks performed by the commands described in this chapter. The reference section of this chapter lists the commands in alphabetical order.

Task	Command
Simulate a dynamic system represented by a Simulink model.	sim
Get simulation options.	simget
Set simulation options.	simset
Plot simulation output.	simplot
Execute a particular phase of the simulation of a model.	model
Build simulation targets for models referenced by this model.	slbuild
Unpack a signal log.	unpack
List the names of signal logging objects in a signal log container object.	who
List the names and types of signal logging objects in a signal log container object.	whos

## Simulation Commands

- model
- sim
- simplot
- simset
- simget
- slbuild
- unpack
- who
- whos

# model

---

## Purpose

Execute a particular phase of the simulation of a model

## Syntax

```
[sys,x0,str,ts] = model([],[],[],'sizes');  
[sys,x0,str,ts] = model([],[],[],'compile');  
outputs = model(t,x,u,'outputs');  
derivs= model(t,x,u,'derivs');  
dstates = model(t,x,u,'update');  
model([],[],[],'term');
```

## Description

The `model` command executes a specific phase of the simulation of a Simulink model whose name is `model`. The command's last (`flag`) argument specifies the phase of the simulation to be executed. See “Simulating Dynamic Systems” for a description of the steps that Simulink uses to simulate a model.

This command is intended to allow linear analysis and other M-file program-based tools to run a simulation step by step, gathering information about the model's states and outputs at each step. It is not intended to be used to run a model step by step, for example, to debug a model. Use the Simulink debugger if you need to examine intermediate results to debug a model.

## Arguments

<code>sys</code>	Vector of model size data: <ul style="list-style-type: none"><li>• <code>sys(1)</code> = number of continuous states</li><li>• <code>sys(2)</code> = number of discrete states</li><li>• <code>sys(3)</code> = number of outputs</li><li>• <code>sys(4)</code> = number of inputs</li><li>• <code>sys(5)</code> = reserved</li><li>• <code>sys(6)</code> = direct-feedthrough flag (1 = yes, 0 = no)</li><li>• <code>sys(7)</code> = number of sample times (= number of rows in <code>ts</code>)</li></ul>
<code>x0</code>	Vector containing the initial conditions of the system's states
<code>str</code>	Vector of names of the blocks associated with the model's states. The state names and initial conditions appear in the same order in <code>str</code> and <code>x0</code> , respectively.
<code>ts</code>	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: <ul style="list-style-type: none"> <li>• 'sizes' executes the size computation phase of the simulation. This phase determines the sizes of the model's inputs, outputs, state vector, etc.</li> <li>• 'compile' executes the compilation phase of the simulation. The compilation phase propagates signal and sample time attributes. It is equivalent to selecting the <b>Update Diagram (Ctrl-D)</b> option from the Simulink <b>Edit</b> menu.</li> <li>• 'update' computes the next values of the model's discrete states.</li> <li>• 'outputs' computes the outputs of the model's blocks at time <math>t</math>.</li> <li>• 'derivs' computes the derivatives of the model's continuous states at time step <math>t</math>.</li> <li>• 'term' causes Simulink to terminate simulation of the model.</li> </ul>

## Examples

This command executes the compilation phase of the vdp model that comes with Simulink.

```
vdp([], [], [], 'compile')
```

The following command terminates the simulation initiated in the previous example.

```
vdp([], [], [], 'term')
```

# model

---

---

**Note** You must always terminate simulation of the model by invoking the model command with the 'term' command. Simulink does not let you close the model until you have terminated the simulation.

---

## See Also

`sim`



<b>Purpose</b>	Simulate a dynamic system	
<b>Syntax</b>	<pre>[t,x,y] = sim(model,timespan,options,ut); [t,x,y1, y2, ..., yn] = sim(model,timespan,options,ut);</pre>	
<b>Description</b>	<p>The <code>sim</code> command executes a Simulink model, using all simulation parameter dialog settings including Workspace I/O options.</p> <p>You can supply a null (<code>[]</code>) matrix for any right-side argument except the first (the model name). The <code>sim</code> command uses default values for unspecified arguments and arguments specified as null matrices. The default values are the values specified by the model. You can set optional simulation parameters, using the <code>sim</code> command's <code>options</code> argument. Parameters set in this way override parameters specified by the model.</p> <p>If you do not specify the left side arguments, the command logs the simulation data specified by the <b>Workspace I/O</b> pane of the <b>Simulation parameters</b> dialog box (see “The Workspace I/O Pane” in the online documentation for Simulink).</p> <p>If you want to simulate a continuous system, you must specify the solver parameter, using <code>simset</code>. The solver defaults to <code>VariableStepDiscrete</code> for purely discrete models.</p>	
<b>Arguments</b>	t	Returns the simulation's time vector.
	x	Returns the simulation's state matrix consisting of continuous states followed by discrete states.
	y	Returns the simulation's output matrix. Each column contains the output of a root-level Outport block, in port number order. If any Outport block has a vector input, its output takes the appropriate number of columns.
	y1,...,yn	Each $y_i$ returns the output of the corresponding root-level Outport block for a model that has $n$ such blocks.
	model	Name of a block diagram.

timespan	<p>Simulation start and stop time. Specify as one of these:</p> <ul style="list-style-type: none"><li>tFinal to specify the stop time. The start time is 0.</li><li>[tStart tFinal] to specify the start and stop times.</li><li>[tStart OutputTimes tFinal] to specify the start and stop times and time points to be returned in t. Generally, t will include more time points. OutputTimes is equivalent to choosing <b>Produce additional output</b> on the dialog box. For a single-rate discrete system, the additional output times specified by OutputTimes must be integer multiples of the fundamental time step. For such a system, you must use an expression of the form</li></ul> $T_s * [\text{vector of integers}]$ <p>where <math>T_s</math> is the fundamental time step to specify the additional output times. Do not use an expression of the form <math>0:T_s:N*T_s</math>.</p>
options	Optional simulation parameters specified as a structure created by the simset command (see simset on page 4-12).
ut	Optional external inputs to top-level Inport blocks. ut can be a MATLAB function (expressed as a string) that specifies the input $u = UT(t)$ at each simulation time step, a table of input values versus time for all input ports, or a comma-separated list of tables, ut1, ut2, ..., each of which corresponds to a specific port. Tabular input for all ports can be in the form of a MATLAB array or a structure. Tabular input for individual ports must be in the form of a structure. See “Loading Input from the Base Workspace” in the online documentation for a description of the array and structure input formats.

## Examples

This command simulates the Van der Pol equations, using the vdp model that comes with Simulink. The command uses all default parameters.

```
[t,x,y] = sim('vdp')
```

This command simulates the Van der Pol equations, using the parameter values associated with the vdp model, but defines a value for the Refine parameter.

```
[t,x,y] = sim('vdp', [], simset('Refine',2));
```

This command simulates the Van der Pol equations for 1,000 seconds, saving the last 100 rows of the return variables. The simulation outputs values for t and y only, but saves the final state vector in a variable called xFinal.

```
[t,x,y] = sim('vdp', 1000, simset('MaxRows', 100,  
    'OutputVariables', 'ty', 'FinalStateName', 'xFinal'));
```

## See Also

simset, simget

# simplot

---

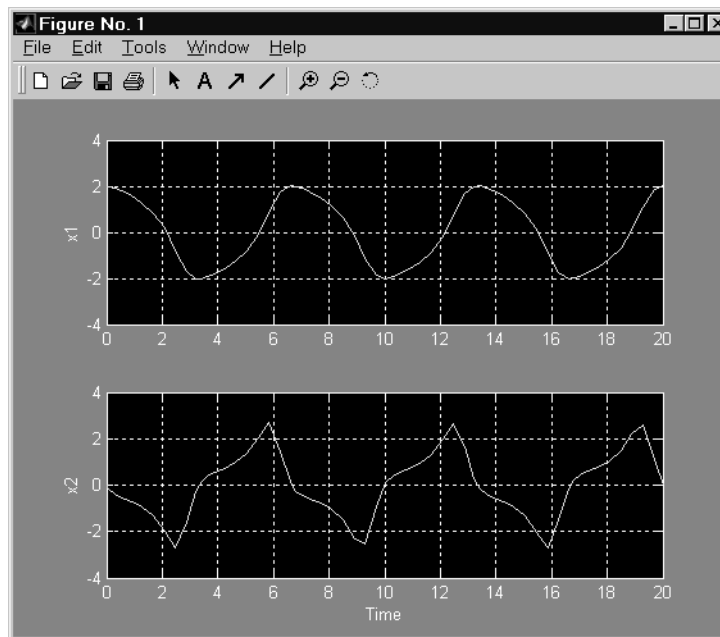
<b>Purpose</b>	Plot simulation data in a figure window	
<b>Syntax</b>	<pre>simplot(data); simplot(time, data);</pre>	
<b>Description</b>	The <code>simplot</code> command plots output from a simulation in a Handle Graphics® figure window. The plot looks like the display on the screen of a Scope block. Plotting the output on a figure window allows you to annotate and print the output.	
<b>Arguments</b>	<code>data</code>	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: <b>Array</b> , <b>Structure</b> , <b>Structure with time</b> (see “The Workspace I/O Pane” in the online documentation for Simulink).
	<code>time</code>	The vector of sample times produced by an output block when you have selected <b>Array</b> or <b>Structure</b> as the simulation’s output format. The <code>simplot</code> command ignores this argument if the format of the data is <b>Structure with time</b> .

## 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`

# simset

---

<b>Purpose</b>	Create or edit simulation parameters and solver properties for the <code>sim</code> command
<b>Syntax</b>	<pre>options = simset(property, value, ...); options = simset(old_opstruct, property, value, ...); options = simset(old_opstruct, new_opstruct); simset</pre>
<b>Description</b>	<p>The <code>simset</code> command creates a structure called <code>options</code>, in which the named simulation parameters and solver properties have specified values. All unspecified parameters and properties take their default values. It is only necessary to enter enough leading characters to uniquely identify the parameter or property. Case is ignored for parameters and properties.</p> <p><code>options = simset(property, value, ...)</code> sets the values of the named properties and stores the structure in <code>options</code>.</p> <p><code>options = simset(old_opstruct, property, value, ...)</code> modifies the named properties in <code>old_opstruct</code>, an existing structure.</p> <p><code>options = simset(old_opstruct, new_opstruct)</code> combines two existing options structures, <code>old_opstruct</code> and <code>new_opstruct</code>, into <code>options</code>. Any properties defined in <code>new_opstruct</code> overwrite the same properties defined in <code>old_opstruct</code>.</p> <p><code>simset</code> with no input arguments displays all property names and their possible values.</p> <p>You cannot obtain or set values of these properties and parameters using the <code>get_param</code> and <code>set_param</code> commands.</p>
<b>Parameters</b>	<p><code>AbsTol</code>                      positive scalar {1e-6}</p> <p><i>Absolute error tolerance.</i> This scalar applies to all elements of the state vector. <code>AbsTol</code> applies only to the variable-step solvers.</p> <p><code>Debug</code>                      'on'   {'off'}   cmds</p> <p><i>Debug.</i> Starts the simulation in debug mode (see “Starting the Debugger” in the online Simulink documentation for more information). The value of this option can be a cell array of commands to be sent to the debugger after it starts, e.g.,</p>

```

opts = simset('debug', ...
             {'strace 4', ...
              'diary solvertrace.txt', ...
              'cont', ...
              'diary off', ...
              'cont'})
sim('vdp',[], opts);

```

Decimation                      positive integer {1}

*Decimation for output variables.* Decimation factor applied to the return variables *t*, *x*, and *y*. A decimation factor of 1 returns every data logging time point, a decimation factor of 2 returns every other data logging time point, etc.

DstWorkspace                  base | {current} | parent

*Where to assign variables.* This property specifies the workspace in which to assign any variables defined as return variables or as output variables on the To Workspace block.

FinalStateName                string {''}

*Name of final states variable.* This property specifies the name of a variable in which Simulink saves the model's states at the end of the simulation.

FixedStep                      positive scalar

*Fixed step size.* This property applies only to the fixed-step solvers. If the model contains discrete components, the default is the fundamental sample time; otherwise, the default is one-fiftieth of the simulation interval.

InitialState                  vector {[]}

*Initial continuous and discrete states.* The initial state vector consists of the continuous states (if any) followed by the discrete states (if any). InitialState supersedes the initial states specified in the model. The default, an empty matrix, causes the initial state values specified in the model to be used.

InitialStep                    positive scalar {auto}

*Suggested initial step size.* This property applies only to the variable-step solvers. The solvers try a step size of InitialStep first. By default, the solvers determine an initial step size automatically.

MaxOrder                      1 | 2 | 3 | 4 | {5}

*Maximum order of ode15s.* This property applies only to ode15s.

MaxDataPoints            nonnegative integer {0}

*Limit number of output data points.* This property limits the number of data points returned in t, x, and y to the last MaxDataPoints data logging time points. If specified as 0, the default, no limit is imposed.

MaxStep                   positive scalar {auto}

*Upper bound on the step size.* This property applies only to the variable-step solvers and defaults to one-fiftieth of the simulation interval.

OutputPoints            {specified} | all

*Determine output points.* When set to specified, the solver produces outputs t, x, and y only at the times specified in timespan. When set to all, t, x, and y also include the time steps taken by the solver.

OutputVariables        {txy} | tx | ty | xy | t | x | y

*Set output variables.* If 't', 'x', or 'y' is missing from the property string, the solver produces an empty matrix in the corresponding output t, x, or y.

Refine                    positive integer {1}

*Output refine factor.* This property increases the number of output points by the specified factor, producing smoother output. Refine applies only to the variable-step solvers. It is ignored if output times are specified.

RelTol                    positive scalar {1e-3}

*Relative error tolerance.* This property applies to all elements of the state vector. The estimated error in each integration step satisfies

$$e(i) \leq \max(\text{RelTol} \cdot \text{abs}(x(i)), \text{AbsTol}(i))$$

This property applies only to the variable-step solvers and defaults to 1e-3, which corresponds to accuracy within 0.1%.

Solver                    VariableStepDiscrete |  
ode45 | ode23 | ode113 | ode15s | ode23s |  
FixedStepDiscrete |  
ode5 | ode4 | ode3 | ode2 | ode1

*Method to advance time.* This property specifies the solver that is used to advance time.



SrcWorkspace {base} | current | parent

*Where to evaluate expressions.* This property specifies the workspace in which to evaluate MATLAB expressions defined in the model.

Trace 'minstep', 'siminfo', 'compile' {''}

*Tracing facilities.* This property enables simulation tracing facilities (specify one or more as a comma-separated list):

- The 'minstep' trace flag specifies that simulation stops when the solution changes so abruptly that the variable-step solvers cannot take a step and satisfy the error tolerances. By default, Simulink issues a warning message and continues the simulation.
- The 'siminfo' trace flag provides a short summary of the simulation parameters in effect at the start of simulation.
- The 'compile' trace flag displays the compilation phases of a block diagram model.

ZeroCross {on} | off

*Enable/disable location of zero crossings.* This property applies only to the variable-step solvers. If set to off, variable-step solvers do not detect zero crossings for blocks having intrinsic zero-crossing detection. The solvers adjust their step sizes only to satisfy error tolerance.

## Examples

This command creates an options structure called myopts that defines values for the MaxDataPoints and Refine parameters, using default values for other parameters.

```
myopts = simset('MaxDataPoints', 100, 'Refine', 2);
```

This command simulates the vdp model for 10 seconds and uses the parameters defined in myopts.

```
[t,x,y] = sim('vdp', 10, myopts);
```

## See Also

sim, simget

# simget

---

**Purpose** Get options structure properties and parameters

**Syntax**

```
struct = simget(model)
value = simget(model, property)
value = simget(OptionStructure, property)
```

**Description** The `simget` command gets simulation parameter and solver property values for the specified Simulink model. If a parameter or property is defined using a variable name, `simget` returns the variable's value, not its name. If the variable does not exist in the workspace, Simulink issues an error message.

`struct = simget(model)` returns the current options structure for the specified Simulink model. The options structure is defined using the `sim` and `simset` commands.

`value = simget(model, property)` extracts the value of the named simulation parameter or solver property from the model.

`value = simget(OptionStructure, property)` extracts the value of the named simulation parameter or solver property from `OptionStructure`, returning an empty matrix if the value is not specified in the structure. `property` can be a cell array containing the list of parameter and property names of interest. If a cell array is used, the output is also a cell array.

You need to enter only as many leading characters of a property name as are necessary to uniquely identify it. Case is ignored for property names.

**Examples** This command retrieves the options structure for the `vdp` model.

```
options = simget('vdp');
```

This command retrieves the value of the `Refine` property for the `vdp` model.

```
refine = simget('vdp', 'Refine');
```

**See Also** `sim`, `simset`

**Purpose** Build model targets.

**Syntax**

```
slbuild('model', 'ModelReferenceSimTarget')  
slbuild('model', 'ModelReferenceRTWTarget')
```

**Description**

`slbuild('model', 'ModelReferenceSimTarget')` builds simulation targets for models referenced by **model** in the current working directory. The command also rebuilds any existing targets, depending on the rebuild option specified in the model's active Model Referencing configuration pane (see “Rebuild options for all referenced models”).

`slbuild('model', 'ModelReferenceRTWTarget')` builds a stand-alone executable from **model**, using its Real-Time Workshop configuration settings. This form of the command requires that you have Real-Time Workshop installed on your system.

# unpack

---

**Purpose** Extract signal logging objects from a Simulink.ModelDataLogs object and write them into the MATLAB workspace.

**Syntax**

```
log.unpack  
unpack('log')  
log.unpack('systems')  
unpack('log', 'systems')  
log.unpack('all')  
unpack('log', '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 are not expanded into individual timeseries 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 or Simulink.SubsysDataLogs object named **log**.

**See Also** whos, who

<b>Purpose</b>	List the contents of a <code>Simulink.ModelDataLogs</code> or <code>Simulink.SubsysDataLogs</code> object.
<b>Syntax</b>	<code>log.who</code> <code>log.who('systems')</code> <code>logout.who('all')</code>
<b>Description</b>	<p><b>log.who</b> or <b>who(log)</b> lists the names of the top-level signal logging objects (i.e., objects of type <code>Simulink.Timeseries</code>, <code>Simulink.TsArray</code>, <code>Simulink.ModelDataLogs</code>, or <code>Simulink.SubsysDataLogs</code>) contained by <b>log</b> where <b>log</b> is the handle of a <code>Simulink.ModelDataLogs</code> object name.</p> <p><b>log.who('systems')</b> or <b>who(log, 'systems')</b> lists the names of all signal logging objects contained by <b>log</b> except for <code>Simulink.Timeseries</code> objects stored in <code>Simulink.TsArray</code> objects contained by <b>log</b>.</p> <p><b>log.who('all')</b> or <b>who(log, 'all')</b> lists the names of all signal logging objects contained by <b>log</b></p>
<b>See Also</b>	<code>whos</code> , <code>unpack</code>

# whos

---

<b>Purpose</b>	List the names and types of simulink data logging objects contained by a <code>Simulink.ModelDataLogs</code> or <code>Simulink.SubsysDataLogs</code> object.
<b>Syntax</b>	<pre><b>log</b>.whos <b>log</b>.whos('systems') <b>log</b>.whos('all')</pre>
<b>Description</b>	<p><b>log.who</b> or <b>who(log)</b> lists the names and types of the top-level signal logging objects (i.e., objects of type <code>Simulink.Timeseries</code>, <code>Simulink.TsArray</code>, <code>Simulink.ModelDataLogs</code>, or <code>Simulink.SubsysDataLogs</code>) contained by <b>log</b> where <b>log</b> is the handle of a <code>Simulink.ModelDataLogs</code> object name.</p> <p><b>log.who('systems')</b> or <b>who(log, 'systems')</b> lists the names and types of all signal logging objects contained by <b>log</b> except for <code>Simulink.Timeseries</code> objects stored in <code>Simulink.TsArray</code> objects contained by <b>log</b>.</p> <p><b>log.who('all')</b> or <b>who(log, 'all')</b> lists the names and types of all signal logging objects contained by <b>log</b></p>
<b>See Also</b>	<code>who</code> , <code>unpack</code>

# Mask Icon Drawing Commands

---

This section describes commands that you can use to create programs that create or modify models.

Command Summary (p. 5-2)

Brief descriptions of commands.

## Command Summary

This table summarizes the commands that you can use to create icons for masked subsystems.

Command	Usage
disp	Display text centered on a mask icon.
dpoly	Display a transfer function on a mask icon.
fprintf	Display variable text on a mask icon.
image	Display an image on a mask icon.
patch	Draws a color patch of a specified shape on a mask icon.
plot	Display graphics on a mask icon.
port_label	Display a port label on a mask icon.
text	Display text at a specified location on a mask icon.



---

## Specifying Parameters and Object Paths

This section explains how to specify parameters and object paths required by model construction commands.

### How to Specify Parameters for the Commands

The commands described in this chapter require that you specify arguments that describe a system, block, or block parameter. Chapter 9, “Model and Block Parameters” provides comprehensive tables of model and block parameters.

### How to Specify a Path for a Simulink Object

Many of the commands described in this chapter require that you identify a Simulink system or block. Identify systems and blocks by specifying their paths:

- To identify a system, specify its name, which is the name of the file that contains the system description, without the mdl extension.  
system
- To identify a subsystem, specify the system and the hierarchy of subsystems in which the subsystem resides.  
system/subsystem<sub>1</sub>/.../subsystem
- To identify a block, specify the path of the system that contains the block and specify the block name.  
system/subsystem<sub>1</sub>/.../subsystem/block

If the block name includes a newline or carriage return, specify the block name as a string vector and use `sprintf('\n')` as the newline character. For example, these lines assign the newline character to `cr`, then get the value for the Signal Generator block’s **Amplitude** parameter.

```
cr = sprintf('\n');  
get_param(['untitled/Signal',cr,'Generator'],'Amplitude')  
ans =  
    1
```

If the block name includes a slash character (/), you repeat the slash when you specify the block name. For example, to get the value of the `Location` parameter for the block named `Signal/Noise` in the `mymodel` system.

```
get_param('mymodel/Signal//Noise','Location')
```

## Mask Icon Drawing Commands

- disp
- dpoly
- fprintf
- image
- patch
- plot
- port\_label
- text

# disp

---

**Purpose** Display text on the icon of a 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.

**Examples** The following command

```
disp('{\itEquation:} \alpha^2 + \beta^2 \rightarrow \gamma^2,
\chi, \phi_3 = {\bf cool}', 'texmode', 'on')
```

draws the equation that appears on this masked block icon.

Equation

**See Also** `fprintf`, `port_label`, `text`

## Purpose

Display a transfer function on the icon of a masked subsystem

## Syntax

```
dpoly(num, den)
dpoly(num, den, 'character')
```

## Description

`dpoly(num, den)` displays the transfer function whose numerator is *num* and denominator is *den*.

`poly(num, den, 'character')` allows you to specify the name of the transfer function's independent variable. The default is *s*.

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:

$$\frac{1}{s^2+2s+1}$$

- 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:

$$\frac{1}{z^2+2z+1}$$

- To display a discrete transfer function in ascending powers of *1/z*, enter `dpoly(num, den, 'z-')`

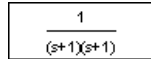
For example, for `num` and `den` as defined previously, the icon looks like this:

$$\frac{z^2}{1+2z^{-1}+z^{-2}}$$

- 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;
```


$$\frac{1}{(s+1)(s+1)}$$

You can add a fourth argument ( ' z ' or ' z - ' ) to express the equation in terms of  $z$  or  $1/z$ .

If the parameters are not defined or have no values when you create the icon, Simulink displays three question marks ( ? ? ? ) in the icon. When the parameter values are entered in the mask dialog box, Simulink evaluates the transfer function and displays the resulting equation in the icon.

## See Also

`disp`, `port_label`, `text`

**Purpose** Display variable text centered on the icon of a 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 commands is identical in name to its corresponding MATLAB function, it provides only the functionality described above.

---

**See Also** `disp`, `port_label`, `text`

# image

---

**Purpose** Display an image on the icon of a masked subsystem

**Syntax**

```
image(a)
image(a, [x, y, w, h])
image(a, [x, y, w, h], 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 to the necessary matrix format.

`image(a, [x, y, w, h])` creates the image at the specified position relative to the lower left corner of the mask.

`image(a, [x, y, w, h], 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.

**See Also** `patch`, `plot`



**Purpose** Draw a color patch of a specified shape on the icon of a masked subsystem

**Syntax** `patch(x, y)`

**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,

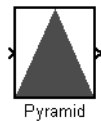
```
patch([0 .5 1], [0 1 0], [1 0 0])
```

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`

# plot

---

**Purpose** Draw a graph connecting a series of points

**Syntax** `plot(Y)`  
`plot(X1,Y1,X2,Y2,...)`

**Description** `plot(Y)` plots, for a vector `Y`, each element against its index. If `Y` is a matrix, it plots each column of the matrix as though it were a vector.

`plot(X1,Y1,X2,Y2,...)` plots the vectors `Y1` against `X1`, `Y2` against `X2`, and so on. Vector pairs must be the same length and the list must consist of an even number of vectors.

Plot commands can include NaN and inf values. When NaNs or infs are encountered, Simulink stops drawing, then begins redrawing at the next numbers that are not NaN or inf.

The appearance of the plot on the icon depends on the value of the **Drawing coordinates** parameter. For more information, see “Icon Options” in the online Simulink documentation.

Simulink displays three question marks (? ? ?) in the block icon and issues warnings in these situations:

- When the values for the parameters used in the drawing commands are not yet defined (for example, when the mask is first created and values have not yet been entered in the mask dialog box)
- When a masked block parameter or drawing command is entered incorrectly

## Examples

This command

```
plot([0 1 5], [0 0 4])
```

generates the plot that appears on the icon for the Ramp block, in the Sources library.



**See Also** `image`

**Purpose** Draw a port label on the icon of a masked subsystem

**Syntax** `port_label(port_type, port_number, label)`  
`port_label(port_type, port_number, label, 'texmode', 'on')`

**Description** `port_label(port_type, port_number, label)` draws a label on a port where `port_type` is either 'input' or 'output', `port_number` is an integer, and `label` is a string specifying the port's label.

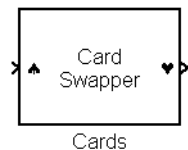
`port_label(port_type, port_number, label, 'texmode', 'on')` lets you use TeX formatting commands in `label`. The TeX formatting commands allow you to include symbols and Greek letters in the port label. See “Mathematical Symbols, Greek Letters, and Tex Characters” in the MATLAB documentation for information on the TeX formatting commands supported by Simulink.

**Examples** The command  
`port_label('input', 1, 'a')`  
defines `a` as the label of input port 1.

The commands

```
disp('Card\nSwapper');  
port_label('input',1,'\spadesuit','texmode','on');  
port_label('output',1,'\heartsuit','texmode','on');
```

draw playing card symbols as the labels of the ports on a masked subsystem.



**See Also** `disp`, `fprintf`, `text`

# text

**Purpose**

Display text at a specific location on the icon of a masked subsystem

**Syntax**

```
text(x, y, text)
text(x, y, text, 'horizontalAlignment', halign,
      'verticalAlignment', valign)
text(x, y, text, 'texmode', 'on')
```

**Description**

The `text` command places a character string (`text` or the contents of `stringvariablename`) at a location specified by the point  $(x,y)$ . The units depend on the **Drawing coordinates** parameter. For more information, see “Icon Options” in the online Simulink documentation.

`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.

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 commands is identical in name to its corresponding MATLAB function, it provides only the functionality described above.

Examples

The command

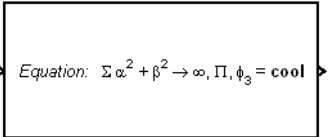
```
text(0.5, 0.5, 'foobar', 'horizontalAlignment', 'center')
```

centers foobar in the icon.

The command

```
text(.05,.5,'\itEquation:} \Sigma \alpha^2 + \beta^2 \rightarrow \infty, \Pi, \phi_3 = {\bf cool}', 'hor','left','texmode','on')
```

draws a left-aligned equation on the icon.



Equation

See Also

disp, fprintf, port\_label

**text**

---

# Simulink Debugger Commands

---

This section describes commands that you can use to create programs that create or modify models.

Command Summary (p. 6-2)

Brief descriptions of commands.

## Command Summary

The following table lists the debugger commands. The table's Repeat column specifies whether pressing the **Enter** key at the command line repeats the command. Detailed descriptions of the commands follow the table.

Command	Short Form	Repeat	Description
animate	ani	No	Enable/disable animation mode.
ashow	as	No	Show an algebraic loop.
atrace	at	No	Set algebraic loop trace level.
bafter	ba	No	Insert a breakpoint after a method.
break	b	No	Insert a breakpoint before a method.
bshow	bs	No	Show a specified block.
clear	cl	No	Clear breakpoints from a model.
continue	c	Yes	Continue the simulation.
disp	d	Yes	Display a block's I/O when the simulation stops.
ebreak	eb	No	Break at recoverable solver errors.
elist	el	No	Display method execution order.
emode	em	No	Toggle between accelerated and normal mode
etrace	et	No	Enable or disable method tracing.
help	? or h	No	Display help for debugger commands.
minor	m	No	Enable or disable breakpoints in minor time steps.
nanbreak	na	No	Set or clear nonfinite value break mode



<b>Command</b>	<b>Short Form</b>	<b>Repeat</b>	<b>Description</b>
next	n	Yes	Go to start of the next time step.
probe	p	No	Display block data.
quit	q	No	Abort simulation.
rbreak	rb	No	Toggle solver reset breakpoint.
run	r	No	Run the simulation to completion.
stimes	sti	No	Display a model's sample times.
slist	sli	No	Display a model's sorted lists.
states	state	No	Display current state values.
status	stat	No	Display debugging options in effect.
step	s	Yes	Advance the simulation by one or more methods.
stop	sto	No	Stop the simulation.
strace	i	No	Set solver trace level.
systems	sys	No	List a model's nonvirtual systems.
tbreak	tb	No	Set or clear a time breakpoint.
trace	tr	Yes	Display a block's I/O each time the block executes.
undisp	und	Yes	Remove a block from the debugger's list of display points.
untrace	unt	Yes	Remove a block from the debugger's list of trace points.
where	w	No	Display the current location of the simulation in the simulation loop.

Command	Short Form	Repeat	Description
xbreak	x	No	Break when the debugger encounters a step-size-limiting state.
zcbreak	zcb	No	Toggle breaking at nonsampled zero-crossing events.
zclist	zcl	No	List blocks containing nonsampled zero crossings.

## Simulink Debugger Commands

animate  
ashow  
atrace  
bafter  
break  
bshow  
clear  
continue  
disp  
ebreak  
elist  
emode  
etrace  
help  
minor  
nanbreak  
next  
probe  
quit  
rbreak  
run  
slist  
states  
status  
step  
stimes  
stop  
strace  
systems  
tbreak  
trace  
undisp  
untrace  
where  
xbreak  
zcbreak  
zclist



<b>Purpose</b>	Enable or disable animation mode				
<b>Syntax</b>	<code>animate [delay   <b>stop</b>]</code>				
<b>Arguments</b>	<table><tr><td><code>delay</code></td><td>Length in seconds between method calls (1 second by default)</td></tr><tr><td><code><b>stop</b></code></td><td>Disable animation mode.</td></tr></table>	<code>delay</code>	Length in seconds between method calls (1 second by default)	<code><b>stop</b></code>	Disable animation mode.
<code>delay</code>	Length in seconds between method calls (1 second by default)				
<code><b>stop</b></code>	Disable animation mode.				
<b>Description</b>	<code>animate</code> without any arguments enables animation mode. <code>animate delay</code> enables animation mode and specifies <code>delay</code> as the time delay in seconds between method calls. <code>animate <b>stop</b></code> disables animation mode.				
<b>See Also</b>	<code>continue</code>				

# ashow

---

<b>Purpose</b>	Show an algebraic loop	
<b>Syntax</b>	ashow <gcb   s:b   s#n   clear>	
<b>Arguments</b>	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.
<b>Description</b>	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 nth algebraic loop in system s. The ashow <b>clear</b> command removes algebraic loop highlights from the model diagram.	
<b>See Also</b>	atrace, slist	

**Purpose** Set algebraic loop trace level

**Syntax** `atrace level`

**Arguments** `level` Trace level (0 = none, 4 = everything).

**Description** The `atrace` command sets the algebraic loop trace level for a simulation.

Command	Displays for Each Algebraic Loop
<code>atrace 0</code>	No information
<code>atrace 1</code>	The loop variable solution, the number of iterations required to solve the loop, and the estimated solution error
<code>atrace 2</code>	Same as level 1
<code>atrace 3</code>	Level 2 plus Jacobian matrix used to solve loop
<code>atrace 4</code>	Level 3 plus intermediate solutions of the loop variable

**See Also** `systems`, `states`

# bafter

---

<b>Purpose</b>	Insert a breakpoint after a specified method	
<b>Syntax</b>	<pre>bafter bafter <b>m</b>:mid bafter &lt;sid:bid   <b>gcb</b>&gt; [mth] [<b>tid</b>:TID] bafter &lt;<b>s</b>:sid   <b>gcs</b>&gt; [mth] [<b>tid</b>:TID] bafter <b>mdl</b> [mth] [<b>tid</b>:TID]</pre>	
<b>Arguments</b>	mid	method ID
	sid:bid	block ID
	<b>gcb</b>	currently selected block
	sid	system ID
	<b>gcs</b>	currently selected system
	<b>mdl</b>	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” on page 13-11).

`bafter sid:bid` inserts a breakpoint after each invocation of the method of the block specified by `sid:bid` (see “Block ID” on page 13-11) in major time steps.

`bafter gcb` inserts a breakpoint after each invocation of a method of the currently selected block (see `gcb`) in major times steps. To break at invocation of a block’s methods in minor time steps, execute `minor before` or `after` executing this form of the `bafter` command.

`bafter s:sid` inserts a breakpoint after each method of the root system or nonvirtual subsystem specified by the system ID: `sid`.

---

**Note** The `systems` command displays the system IDs for all nonvirtual systems in the currently selected model.

---



**bafter gcs** inserts a breakpoint after each method of the currently selected nonvirtual system.

**bafter mdl** inserts a breakpoint after each method of the currently selected model.

The optional **mtb** parameter allow you to set a breakpoint after a particular block, system, or model method and task. For example, **bafter gcb Outputs** sets a breakpoint after the Outputs method of the currently selected block.

The optional **TID** parameter allows you to set a breakpoint after invocation of a method by a particular task. For example, suppose that the currently selected nonvirtual subsystem operates on task 2 and 3. Then **bafter gcs Outputs tid:2** sets a breakpoint after the invocation of the subsystem's Outputs method that occurs when task 2 is active.

## **See Also**

**break**, **ebreak**, **tbreak**, **xbreak**, **nanbreak**, **zcbreak**, **rbreak**, **clear**, **where**, **slist**, **systems**

# break

---

**Purpose** Insert a breakpoint before a specified method.

**Syntax**

```
break
break m:mid
break <sid:bid | gcb> [mth] [tid:TID]
break <s:sid | gcs> [mth] [tid:TID]
break mdl [mth] [tid:TID]
```

**Arguments**

<b>mid</b>	method ID
<b>sid:bid</b>	block ID
<b>gcb</b>	currently selected block
<b>sid</b>	system ID
<b>gcs</b>	currently selected system
<b>mdl</b>	currently selected model
<b>mth</b>	a method name, e.g., Outputs.Major
<b>TID</b>	task ID

**Description**

`break` inserts a breakpoint before the current method.

`break m:mid` inserts a breakpoint before the method specified by `mid` (see “Method ID”).

`break sid:bid` inserts a breakpoint before each invocation of the method of the block specified by `sid:bid` (see “Block ID”) in major time steps. `break gcb` inserts a breakpoint before each invocation of a method of the currently selected block (see `gcb`) in major times steps. To break before invocation of a block’s methods in minor time steps, execute `minor before` or after executing this form of the `bafter` command.

`break s:sid` inserts a breakpoint at each method of the root system or nonvirtual subsystem specified by the system ID: `sid`.

---

**Note** The `systems` command displays the system IDs for all nonvirtual systems in the currently selected model.

---

`break gcs` inserts a breakpoint at each method of the currently selected nonvirtual system.

`break mdl` inserts a breakpoint at each method of the currently selected model.

The optional *meth* 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`, `ebreak`, `tbreak`, `xbreak`, `nanbreak`, `zcbreak`, `rbreak`, `clear`, `where`, `slist`, `systems`

# bshow

---

<b>Purpose</b>	Show a specified block
<b>Syntax</b>	bshow s:b
<b>Arguments</b>	s:b            The block whose system index is s and block index is b.
<b>Description</b>	The bshow command opens the model window containing the specified block and selects the block.
<b>See Also</b>	slist

**Purpose** Clear breakpoints from a model.

**Syntax**

```
clear  
clear m:mid  
clear id  
clear <sid:bid | gcb>
```

**Arguments**

<b>mid</b>	method ID
<b>id</b>	breakpoint ID
<b>sid:bid</b>	block ID
<b>gcb</b>	currently selected block

**Description**

`clear` clears a breakpoint from the current method.

`clear m:mid` clears a breakpoint from the method specified by **mid**.

`clear id` clears the breakpoint specified by the breakpoint ID **id**.

`clear sid:bid` clears any breakpoints set on the methods of the block specified by **sid:bid**.

`clear gcb` clears any breakpoints set on the methods of the currently selected block.

**See Also** `break`, `bafter`, `slist`

# continue

---

**Purpose** Continue the simulation

**Syntax** `continue`

**Description** The `continue` command continues the simulation from the current breakpoint. If animation mode is not enabled, the simulation continues until it reaches another breakpoint or its final time step. If animation mode is enabled, the simulation continues in animation mode to the first method of the next major time step, ignoring breakpoints.

**See Also** `run`, `stop`, `quit`, `animate`

<b>Purpose</b>	Display a block's I/O when the simulation stops	
<b>Syntax</b>	<code>disp</code> <code>disp <b>gcb</b></code> <code>disp s:b</code>	
<b>Arguments</b>	<code>s:b</code>	The block whose system index is s and block index is b.
	<code>gcb</code>	Current block.
<b>Description</b>	The <code>disp</code> 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 <code>disp</code> without arguments shows a list of display points. Use <code>undisp</code> to unregister a block.	
<b>See Also</b>	<code>undisp</code> , <code>slist</code> , <code>probe</code> , <code>trace</code>	

# ebreak

---

<b>Purpose</b>	Enable (or disable) a breakpoint on solver errors.
<b>Syntax</b>	ebreak
<b>Description</b>	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.
<b>See Also</b>	break, bafter, tbreak, xbreak, nanbreak, zcbreak, rbreak, clear, where, slist, systems



**Purpose** List simulation methods in the order in which they are executed during a simulation

**Syntax**

```
elist m:mid [tid:TID]
elist <gcs | s:sid> [mth] [tid:TID]
elist <gcb | sid:bid> [mth] [tid:TID]
```

**Description** elist m:mid lists the methods invoked by the system or nonvirtual subsystem method corresponding to the method id mid (see the where command for information on method IDs), e.g.,

```
sldebug @19): elist n:19

RootSystem.Outputs 'vdp' [tid=0] : ← Calling method
0:0 Integrator.Outputs 'x1' [tid=0]
0:1 Outport.Outputs 'Out1' [tid=0]
0:2 Integrator.Outputs 'x2' [tid=0]
...

Block id      Method      Block      Task id
```

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 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':
  0:4 Scope.Start 'Scope'
  0:5 Fcn.Start 'Fcn'
  0:6 Product.Start 'Product'
  0:7 Gain.Start 'Mu'
  0:8 Sum.Start 'Sum'

RootSystem.Initialize 'vdp':
  0:0 Integrator.Initialize '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':
  0:4 Scope.Start 'Scope'
  0:5 Fcn.Start 'Fcn'
  0:6 Product.Start 'Product'
  0:7 Gain.Start 'Mu'
  0:8 Sum.Start 'Sum'
```

Use `elist gcb` to list the methods invoked by the subsystem currently selected in the model.

## See Also

`where`, `slist`, `systems`

# emode

---

<b>Purpose</b>	Toggle model execution between accelerated and normal mode
<b>Syntax</b>	emode
<b>Description</b>	Toggles the simulation between accelerated and normal mode when using the Simulink Accelerator. See “Using the Simulink Accelerator with the Simulink Debugger” in <i>Using Simulink</i> for more information.

**Purpose** Enable or disable method tracing

**Syntax** etrace **level** level-number

**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

---

<b>Purpose</b>	Display help for debugger commands
<b>Syntax</b>	help
<b>Description</b>	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.

<b>Purpose</b>	Enable or disable breakpoints in minor time steps
<b>Syntax</b>	<code>minor</code>
<b>Description</b>	The <code>minor</code> command causes the debugger to enable or disable minor step breakpoint mode. By default, executing <code>break &lt;sid:bid   gcb&gt;</code> sets breakpoints only on invocations of a block's methods, e.g., Outputs, in major time steps. To enable breakpoints at invocations of the methods in minor time steps, you must execute <code>minor</code> before or after executing the <code>break</code> command. To disable breaking in minor steps, execute <code>minor</code> again.
<b>See Also</b>	<code>break</code> , <code>bafter</code>

# nanbreak

---

<b>Purpose</b>	Set or clear nonfinite value break mode
<b>Syntax</b>	nanbreak
<b>Description</b>	The nanbreak command causes the debugger to break whenever the simulation encounters a <i>nonfinite</i> (NaN or Inf) value. If nonfinite break mode is set, nanbreak clears it.
<b>See Also</b>	break, bafter, ebreak, xbreak, tbreak, zcbreak, rbreak



<b>Purpose</b>	Advance the simulation to the start of the next method at the current level in the model's execution list
<b>Syntax</b>	<code>next</code>
<b>Description</b>	<p>The <code>next</code> command advances the simulation to the start of the next method at the current level in the model's method execution list.</p> <hr/> <p><b>Note</b> The <code>next</code> command has the same effect as the <code>step over</code> command. See the <code>step</code> documentation for more information.</p> <hr/>
<b>See Also</b>	<code>step</code>

# probe

---

<b>Purpose</b>	Display block data.								
<b>Syntax</b>	<code>probe [&lt;s:b   gcb&gt;] [<b>level</b> io   (all)]</code>								
<b>Arguments</b>	<table><tr><td><code>s:b</code></td><td>The block whose system index is s and block index is b.</td></tr><tr><td><code>gcb</code></td><td>Current block.</td></tr><tr><td><code>level io</code></td><td>Display block's I/O.</td></tr><tr><td><code>level all</code></td><td>Display all information regarding a block's current state, including inputs and outputs, states, and zero crossings.</td></tr></table>	<code>s:b</code>	The block whose system index is s and block index is b.	<code>gcb</code>	Current block.	<code>level io</code>	Display block's I/O.	<code>level all</code>	Display all information regarding a block's current state, including inputs and outputs, states, and zero crossings.
<code>s:b</code>	The block whose system index is s and block index is b.								
<code>gcb</code>	Current block.								
<code>level io</code>	Display block's I/O.								
<code>level all</code>	Display all information regarding a block's current state, including inputs and outputs, states, and zero crossings.								
<b>Description</b>	<p>probe causes the debugger to enter or exit probe mode. In probe mode, the debugger displays the I/O of any block you select. To exit probe mode, enter any command. <code>probe gcb</code> displays the I/O of the currently selected block. <code>probe s:b</code> displays the I/O of the block whose index is s:b.</p>								
<b>See Also</b>	<code>disp</code> , <code>trace</code>								

<b>Purpose</b>	Abort simulation
<b>Syntax</b>	<code>quit</code>
<b>Description</b>	The <code>quit</code> command terminates the current simulation.
<b>See Also</b>	<code>stop</code>

# rbreak

---

<b>Purpose</b>	Break when the simulation requires a solver reset.
<b>Syntax</b>	<code>rbreak</code>
<b>Description</b>	This command enables (or disables) a solver reset breakpoint if the breakpoint is disabled (or enabled). The breakpoint causes the debugger to halt the simulation whenever an event that requires a solver reset occurs. The halt occurs before the solver is reset.
<b>See Also</b>	<code>break</code> , <code>bafter</code> , <code>ebreak</code> , <code>xbreak</code> , <code>tbreak</code> , <code>zcbreak</code> , <code>nanbreak</code>

---

<b>Purpose</b>	Run the simulation to completion
<b>Syntax</b>	run
<b>Description</b>	The run command runs the simulation from the current breakpoint to its final time step. It ignores breakpoints and display points.
<b>See Also</b>	continue, stop, quit

<b>Purpose</b>	Display the sorted list of a model's root system and of each of its nonvirtual subsystems
<b>Syntax</b>	slist
<b>Description</b>	The slist command displays the sorted list of a model's root system and each of its nonvirtual subsystems. For example, the sorted list for the vdp model's root system is

```
---- Sorted list for 'vdp' [9 nonvirtual blocks, directFeed=0]
0:0    'vdp/x1' (Integrator)
0:1    'vdp/Out1' (Outport)
0:2    'vdp/x2' (Integrator)
0:3    'vdp/Out2' (Outport)
0:4    'vdp/Scope' (Scope)
0:5    'vdp/Fcn' (Fcn)
0:6    'vdp/Product' (Product)
0:7    'vdp/Mu' (Gain)
0:8    'vdp/Sum' (Sum)
```

For each system (root or nonvirtual), the slist command displays a title line followed by an entry for each block in the order in which the blocks appear in the sorted list. The title line specifies the name of the system, the number of nonvirtual blocks that the system contains, and the number of blocks in the system that have direct feedthrough ports. Each block entry lists the block's id and the name and type of the block. The block id consists of a system index and a block index separated by a colon (s:b). The block index is the position of the block in the sorted list. The system index is the order in which Simulink generated the system's sorted list. The system index has no special significance. It simply allows blocks that appear in the same position in different sorted lists to have unique identifiers.

A sorted list is a list of a root system or nonvirtual subsystem's blocks sorted according to data dependencies and other criteria. Simulink uses sorted lists to create block method execution lists (see elist) for root system and nonvirtual subsystem methods that invoke the corresponding methods of the blocks that the root system or subsystem contains. In general, root system and nonvirtual subsystem methods invoke the block methods in the same order as the blocks appear in the sorted list. However, significant exceptions occur. For example, execution lists for multitask models group all blocks operating at the same rate

(i.e., in the same task) together with slower groups appearing later than faster groups. The grouping of methods by task can result in an order of block method execution that differs from the order in which blocks appear in the sorted list. However, within groups, methods execute in the same order as the corresponding blocks appear in the sorted list.

**See Also**

systems, elist

# states

---

<b>Purpose</b>	Display current state values
<b>Syntax</b>	states
<b>Description</b>	The states command displays a list of the current states of the model. The display lists the value, index, and name of each state.



<b>Purpose</b>	Display debugging options in effect
<b>Syntax</b>	<code>status</code>
<b>Description</b>	The <code>status</code> command displays a list of the debugging options in effect.

# step

**Purpose** Advance the simulation by one or more methods

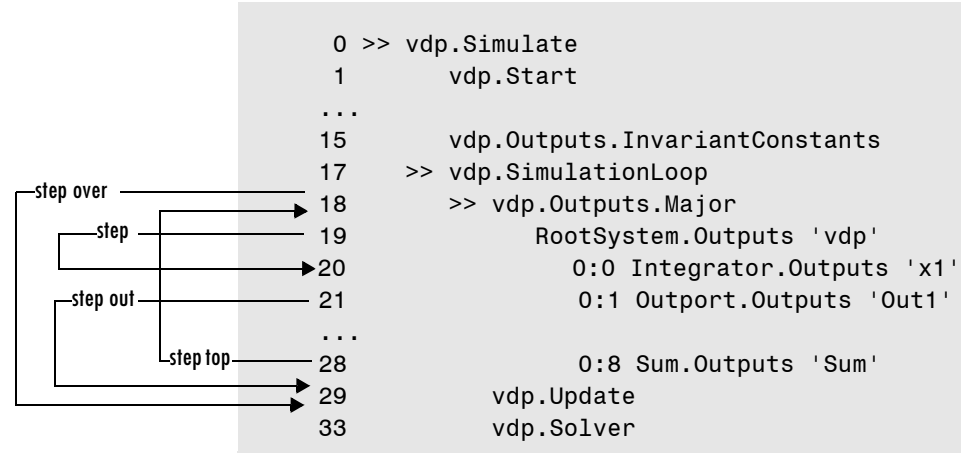
**Syntax**

```
step [in into]
step over
step out
step top
step blockmth
```

**Description** This command advances the simulation

- into (step [in into]), over (step over), or out of the method at which the simulation is currently stopped (step out)
- to the top of the simulation loop (step top), i.e., to the start of the first method executed at the start of the next time step
- to the next method that operates on a block (step blockmth)

The following diagram illustrates the effect of various forms of the step command.



If this command advances the simulation to the start of a block method, the debugger points the debug pointer at the block on which the method operates.

**See Also** next, where, elist

<b>Purpose</b>	Display the sample times defined by the model being debugged.
<b>Syntax</b>	<code>stimes</code>
<b>Description</b>	This command displays information about the sample times defined by this model, including the sample time's period, offset, and task ID.
<b>Example</b>	<p>The following command displays the sample times for the f14 demo</p> <pre>(sldebug @0): &gt;&gt; stimes  --- Sample times for 'f14' [Number of sample times = 3]   1. [0      , 0      ] tid=0 (continuous sample time)   2. [0      , 1      ] tid=1 (continuous but fixed in minor step)   3. [0.1    , 0      ] tid=2</pre>

# stop

---

<b>Purpose</b>	Stop the simulation
<b>Syntax</b>	stop
<b>Description</b>	The stop command stops the simulation.
<b>See Also</b>	continue, run, quit

<b>Purpose</b>	Set solver trace level.
<b>Syntax</b>	<code>strace [0   1   2   3   4]</code>
<b>Description</b>	The <code>strace</code> command causes the solver to display diagnostic information at the command line after each debugger command that involves the solver. The command's numeric argument specifies the level of detail of the diagnostic information. Valid values range from 0 (no information) to 4 (maximum detail).
<b>See Also</b>	<code>atrace</code> , <code>trace</code>

# systems

---

<b>Purpose</b>	List a model's nonvirtual systems
<b>Syntax</b>	<code>systems</code>
<b>Description</b>	The <code>systems</code> command lists a model's nonvirtual systems in the MATLAB Command Window.
<b>See Also</b>	<code>slist</code>

<b>Purpose</b>	Set or clear a time breakpoint
<b>Syntax</b>	<code>tbreak</code> <code>tbreak t</code>
<b>Description</b>	The <code>tbreak</code> command sets a breakpoint at the specified time step. If a breakpoint already exists at the specified time, <code>tbreak</code> clears the breakpoint. If you do not specify a time, <code>tbreak</code> toggles a breakpoint at the current time step.
<b>See Also</b>	<code>break</code> , <code>bafter</code> , <code>ebreak</code> , <code>xbreak</code> , <code>nanbreak</code> , <code>zcbreak</code> , <code>rbreak</code>

# trace

---

<b>Purpose</b>	Display a block's I/O each time the block executes	
<b>Syntax</b>	<code>trace gcb</code> <code>trace s:b</code>	
<b>Arguments</b>	<code>s:b</code>	The block whose system index is s and block index is b.
	<code>gcb</code>	Current block.
<b>Description</b>	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.	
<b>See Also</b>	<code>disp</code> , <code>probe</code> , <code>untrace</code> , <code>slist</code> , <a href="#"><code>strace</code></a>	



<b>Purpose</b>	Remove a block from the debugger's list of display points	
<b>Syntax</b>	<code>undisp gcb</code> <code>undisp s:b</code>	
<b>Arguments</b>	<code>s:b</code>	The block whose system index is s and block index is b.
	<code>gcb</code>	Current block.
<b>Description</b>	The undisp command removes the specified block from the debugger's list of display points.	
<b>See Also</b>	<code>disp</code> , <code>slist</code>	

# untrace

---

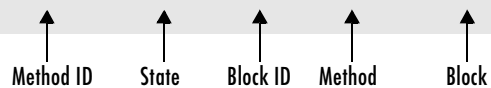
<b>Purpose</b>	Remove a block from the debugger's list of trace points	
<b>Syntax</b>	<code>untrace gcb</code> <code>untrace s:b</code>	
<b>Arguments</b>	<code>s:b</code>	The block whose system index is s and block index is b.
	<code>gcb</code>	Current block.
<b>Description</b>	The <code>untrace</code> command removes the specified block from the debugger's list of trace points.	
<b>See Also</b>	<code>trace</code> , <code>slist</code>	

**Purpose** Display the current location of the simulation in the simulation loop

**Syntax** where [detail]

**Description** The where command displays the current location of the simulation in the simulation loop, for example,

```
sldebug @7): where
  0 >> vdp.Simulate
  1   >> vdp.Start
  2   >> RootSystem.Start 'vdp'
  7   >| 0:8 Sum.Start 'Sum'
```



Method ID      State      Block ID      Method      Block

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

---

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

<b>Purpose</b>	Break when the debugger encounters a step-size-limiting state
<b>Syntax</b>	<code>xbreak</code>
<b>Description</b>	The <code>xbreak</code> command pauses execution of the model when the debugger encounters a state that limits the size of the steps that the solver takes. If <code>xbreak</code> mode is already on, <code>xbreak</code> turns the mode off.
<b>See Also</b>	<code>break</code> , <code>bafter</code> , <code>ebreak</code> , <code>zcbreak</code> , <code>tbreak</code> , <code>nanbreak</code> , <code>rbreak</code>

# zcbreak

---

<b>Purpose</b>	Toggle breaking at nonsampled zero-crossing events
<b>Syntax</b>	<code>zcbreak</code>
<b>Description</b>	The <code>zcbreak</code> command causes the debugger to break when a nonsampled zero-crossing event occurs. If zero-crossing break mode is already on, <code>zcbreak</code> turns the mode off.
<b>See Also</b>	<code>break</code> , <code>bafter</code> , <code>xbreak</code> , <code>tbreak</code> , <code>nanbreak</code> , <code>zclist</code>

<b>Purpose</b>	List blocks containing nonsampled zero crossings
<b>Syntax</b>	<code>zclist</code>
<b>Description</b>	The <code>zclist</code> command displays a list of blocks in which nonsampled zero crossings can occur. The command displays the list in the MATLAB Command Window.
<b>See Also</b>	<code>zcbreak</code>





# Data Type Functions

---

This section describes functions that create MATLAB structures or Simulink objects that define data types. You can use these functions in Simulink models to specify user-defined data types.

## Data Type Functions

```
fixdt  
fixptbestexp  
fixptbestprec  
fixpt_evenspace_cleanup  
fixpt_interp1  
fixpt_look1_func_approx  
fixpt_look1_func_plot  
fixpt_set_all  
float  
fxptdlg  
num2fixpt  
sfix  
sfrac  
sint  
ufix  
uint
```

**Purpose**

Create a `Simulink.NumericType` object describing a 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.

`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.

`[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**

`sfix`, `ufix`, `sint`, `uint`, `sfrac`, `ufrac`, `float`

# fixptbestexp

---

**Purpose** Determine the exponent that gives the best precision fixed-point representation of a value

**Syntax**

```
out = fixptbestexp(RealWorldValue,TotalBits,IsSigned)
out = fixptbestexp(RealWorldValue,FixPtDataType)
```

**Description** `out = fixptbestexp(RealWorldValue,TotalBits,IsSigned)` determines the exponent that gives the best 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 exponent is returned to `out`.

`out = fixptbestexp(RealWorldValue,FixPtDataType)` determines the exponent that gives the best precision based on the data type specified by `FixPtDataType`.

**Examples** The following command returns 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, you can specify the fixed-point data type:

```
out = fixptbestexp(4/3,sfix(16))
out =
    -14
```

This value means that the maximum precision representation of 4/3 is obtained by placing 14 bits to the right of the binary point:

```
01.01010101010101
```

You would 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`

**Purpose** Determine the maximum precision available for the fixed-point representation of a 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 parameter 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%:

# fixptbestprec

---

```
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 **Output scaling value** parameter in a block mask in your model. Be sure to select the **Lock output scaling against changes by the autoscaling tool** parameter in the same block to prevent the scaling from being overridden by the **Fixed-Point Settings** interface. 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 the `autofixexp` reference page.

## See Also

`fixptbestexp`

<b>Purpose</b>	Modify lookup table input data to be evenly spaced
<b>Syntax</b>	<pre>xdata_adjusted = fixpt_evenspace_cleanup(xdata_original, xdt,  xscale)</pre>
<b>Description</b>	<p><code>xdata_adjusted = fixpt_evenspace_cleanup(xdata_original, xdt, xscale)</code> modifies lookup table input data to be evenly spaced if it is not quite evenly spaced after quantization. For example, 0:0.005:1 appears evenly spaced, but if it is quantized with scaling <math>2^{-12}</math>, it is not evenly spaced. Loss of even spacing can make a significant impact on the efficiency of your implementation. Code generated by Real-Time Workshop to implement an uneven lookup table is more complicated. In addition, unevenly spaced input data is stored in data memory. If you modify the input data to remain evenly spaced after quantization, Real-Time Workshop generates simpler code and excludes the input data from memory, thereby saving significant amounts of data memory.</p> <p>The modifications to the lookup table input data are likely to change the numerical behavior of the table. The numerical changes may or may not be trivial, so you should test the model using simulation, rapid prototyping, or other appropriate methods. This function is intended for use with nontunable data. Tunable data is always treated as if it were unevenly spaced. Even if tunable data starts out evenly spaced, it may later be tuned to values that are unevenly spaced.</p> <p>It is important to note that the data is judged to be “almost” evenly spaced relative to the scaling slope. Consider the data vector [0 2 5], which has spacing value 2 and 3. A natural first impression is that the data has significantly 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, then a spacing variation of 1 represents a one bit change or less. A spacing variation of one bit or less is judged to be “almost” evenly spaced, and this function will adjust the data to force it to be evenly spaced.</p>

# fixpt\_evenspace\_cleanup

The required input parameters of this function are as follows.

Input	Value	Example
xdata_original	Input lookup data	0:0.005:1
xdt	Input data type	sfix(16)
xscale	Input scaling	2^-12

## See Also

fixdt, fixpt\_interp1, fixpt\_look1\_func\_approx, sfix, ufix



<b>Purpose</b>	Implement a 1-D lookup table
<b>Syntax</b>	<code>y = fixpt_interp1(xdata,ydata,x,xdt,xscale,ydt,yscale,rndmeth)</code>
<b>Description</b>	<p><code>fixpt_interp1(xdata,ydata,x,xdt,xscale,ydt,yscale,rndmeth)</code> implements a lookup table to find output(s) <code>y</code> for input(s) <code>x</code>. If <code>x</code> falls between two <code>xdata</code> values, then <code>y</code> is found by interpolating between the corresponding <code>ydata</code> pair. If <code>x</code> falls above the range given by <code>xdata</code>, <code>y</code> is given as the maximum <code>ydata</code> value. If <code>x</code> falls below the range given by <code>xdata</code>, <code>y</code> is given as the minimum <code>ydata</code> value.</p> <p>If either the input data type, <code>xdt</code>, or the output data type, <code>ydt</code>, is floating point, then floating-point calculation is used to perform the interpolation. Otherwise, integer-only calculation is used. This calculation handles the input scaling, <code>xscale</code>, and the output scaling, <code>yscale</code>, appropriately, and obeys the designated rounding method, <code>rndmeth</code>.</p>
<b>Examples</b>	<p>Define <code>xdata</code> as a vector of 33 evenly spaced points between 0 and 8, and <code>ydata</code> as the sinc of <code>xdata</code>.</p> <pre>xdata = linspace(0,8,33).'; ydata = sinc(xdata);</pre> <p>Now define your input <code>x</code> as a vector of 201 evenly spaced points between -1 and 9.</p> <pre>x = linspace(-1,9,201).';</pre> <p>Notice that <code>x</code> includes some values that are both lower and higher than the range of <code>xdata</code>.</p> <p>You can now use <code>fixpt_interp1</code> to interpolate outputs for <code>x</code>.</p> <pre>y = fixpt_interp1(xdata,ydata,x,sfix(8),2^-3,sfix(16),2^-14,...     'Floor')</pre>
<b>See Also</b>	<code>fixpt_look1_func_approx</code> , <code>fixpt_look1_func_plot</code>

# fixpt\_look1\_func\_approx

**Purpose** Optimize for a fixed-point function, the x values, or breakpoints, that are generated for a lookup table

**Syntax**

```
[xdata,ydata,errworst]=fixpt_look1_func_approx('funcstr',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax)

[xdata,ydata,errworst]=fixpt_look1_func_approx('funcstr',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,[],nptsmax)

[xdata,ydata,errworst]=fixpt_look1_func_approx('funcstr',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,nptsmax)

[xdata,ydata,errworst]=fixpt_look1_func_approx('funcstr',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,nptsmax,spacing)
```

**Description** fixpt\_look1\_func\_approx('funcstr',xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax) optimizes the breakpoints of a lookup table over a specified range. The lookup table satisfies the maximum acceptable error, maximum number of points, and spacing requirements given by the optional parameters. The breakpoints refer to the x values of the lookup table. The command

```
[xdata,ydata,errworst]=fixpt_look1_func_approx('funcstr',...
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax)
```

returns the *x* and *y* coordinates of the lookup table as vectors *xdata* and *ydata*, respectively. It also returns the maximum absolute error of the lookup table as a variable *errworst*.

The fixed-point approximation is found by interpolating between the lookup table data points. The required input parameters are as follows.

Input	Value
'funcstr'	Function of x funcstr is the function for which breakpoints are approximated.
xmin	Minimum value of x
xmax	Maximum value of x
xdt	Data type of x

Input	Value
xscale	Scaling for the x values
ydt	Data type of y
yscale	Scaling for the y values
rndmeth	Rounding mode supported by fixed-point Simulink blocks: 'Toward Zero', 'Nearest', 'Floor' (default value), 'Ceiling'

- xmin and xmax specify the range over which the breakpoints are approximated.
- xdt, xscale, ydt, yscale, and rndmeth follow conventions used by fixed-point Simulink blocks.
- rndmeth has a default value listed in the input table.

In addition to the required parameters, there are three optional inputs, as follows.

Input	Value
errmax	Maximum acceptable error
nptsmax	Maximum number of points
errworst	Spacing: 'even', 'pow2' (even power of 2), 'unrestricted' (default value)

Of these, you must use at least one of the parameters errmax and nptsmax. If you omit one of these, use brackets, [ ], in place of the omitted parameter. The function will then ignore that requirement for the lookup table.

# fixpt\_look1\_func\_approx

The outputs of the function are as follows.

Output	Value
xdata	The breakpoints for the lookup table
ydata	The ideal function applied to the breakpoints
errworst	The worst case error, which is the maximum absolute error between the ideal function and the approximation given by the lookup table

## Criteria For Optimizing the Breakpoints: errmax, nptsmx, and spacing

The approximation produced from the lookup table must satisfy the requirements for the maximum acceptable error, errmax, the maximum number of points, nptsmx, and the spacing, spacing. The requirements are

- The maximum absolute error is less than errmax.
- The number of points required is less than nptsmx.
- The spacing is specified as unrestricted, even or even power of 2.

## Modes for errmax and nptsmx

- If both errmax and nptsmx are specified  
The returned breakpoints will meet both criteria if possible. The errmax parameter is given priority, and nptsmx is ignored, if both criteria cannot be met with the specified spacing.
- If only errmax is specified  
The breakpoints that meet the error criteria, and have the least number of points are returned
- If only nptsmx is specified  
The breakpoints that require nptsmx or fewer, and give the smallest worst case error are returned

## Modes for Spacing

If no spacing is specified, and more than one spacing method meets the requirements given by errmax and nptsmx, power of 2 spacing is chosen over

even spacing, which in turn is chosen over uneven spacing. This case occurs when the `errmax` and `nptsmax` are both specified, but typically does not occur when only one is specified:

- If `unrestricted` is entered, the function chooses the spacing that provides the best optimization.
- If `even` is entered, the function chooses an evenly spaced set of points, including the `pow2` spacing.
- If `pow2` spacing is entered, the function chooses an even power of 2 spaced set of points.

---

**Note** The global optimum may not be found. The worst case error can depend on fixed-point calculations, which are highly nonlinear. Furthermore, the optimization approach is heuristic.

---

The spacing you choose depends on the parameters you want to optimize: execution speed, function approximation error, ROM usage, and RAM usage:

- The execution speed depends on the bisection search, and the interpolation method.
- The error depends on how accurately the method approximates the nonuniform curvature of the function.
- The ROM usage depends on the amount of data and command ROM used.
- The RAM usage depends on how much global and stack RAM is used.

When the lookup table has even power of two spacing, division is replaced by a bit shift. As a result, the execution speed is faster than for evenly spaced data.

### Using the Approximation Function

- 1 Choose a function and use the `eval('funcstr');` command to view the function before creating the lookup table.
- 2 Define the remaining inputs.
- 3 Run the `fixpt_look1_func_approx` function.

- 4 Use the `fixpt_look1_func_plot` function to plot the function from the selected breakpoints, and to calculate the error and the number of points used.
- 5 Vary the inputs to produce sets of breakpoints that generate functions with varying number of points required and worst case error.
- 6 Compare the number of points required and worst case error from various runs to choose the best set of breakpoints.

### Calculating the Output Function

To calculate the function, use the returned breakpoints with

- The `eval` function
- A function lookup table. The x values are the breakpoints from the `fixpt_look1_func_approx` function, and the y values can be supplied using the `eval` function.

See Chapter 6, “Tutorial: Producing Lookup Table Data” for a tutorial on using `fixpt_look1_func_approx`.

The following table summarizes the effect of spacing on the execution speed, error, and memory used.

Parameter	Even Power of 2 Spaced Data	Evenly Spaced Data	Unevenly Spaced Data
Execution Speed	The execution speed is the fastest. The position search and interpolation are the same as for evenly spaced data. However, to increase the speed more, the position search is replaced by a bit shift, and the interpolation is replaced with a bit mask.	The execution speed is faster than that for unevenly spaced data because the position search is faster and the interpolation requires a simple division.	The execution speed is the slowest of the different spacings because the position search is slower, and the interpolation requires more operations.
Error	The error can be larger than that for unevenly spaced data because approximating a function with nonuniform curvature requires more points to achieve the same accuracy.	The error can be larger than that for unevenly spaced data because approximating a function with nonuniform curvature requires more points to achieve the same accuracy.	The error can be smaller because approximating a function with nonuniform curvature requires fewer points to achieve the same accuracy.
ROM Usage	Uses less command ROM, but more data ROM.	Uses less command ROM, but more data ROM.	Uses more command ROM, and less data ROM.
RAM Usage	Not significant.	Not significant.	Not significant.

## Examples

This example produces a lookup table for a sine function. The inputs for the example are as follows:

## fixpt\_look1\_func\_approx

---

```
funcstr = 'sin(2*pi*x)';  
xmin = 0;  
xmax = 0.25;  
xdt = ufix(16);  
xscale = 2^-16;  
ydt = sfix(16);  
yscale = 2^-14;  
rndmeth = 'Floor';  
errmax = 2^-10;  
spacing = 'pow2';
```

To create the lookup table, type

```
[xdata, ydata, errWorst]=fixpt_look1_func_approx(funcstr,  
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth,errmax,[],spacing);
```

The brackets [] are a place holder for the nptsmax parameter, which is not used in this example.

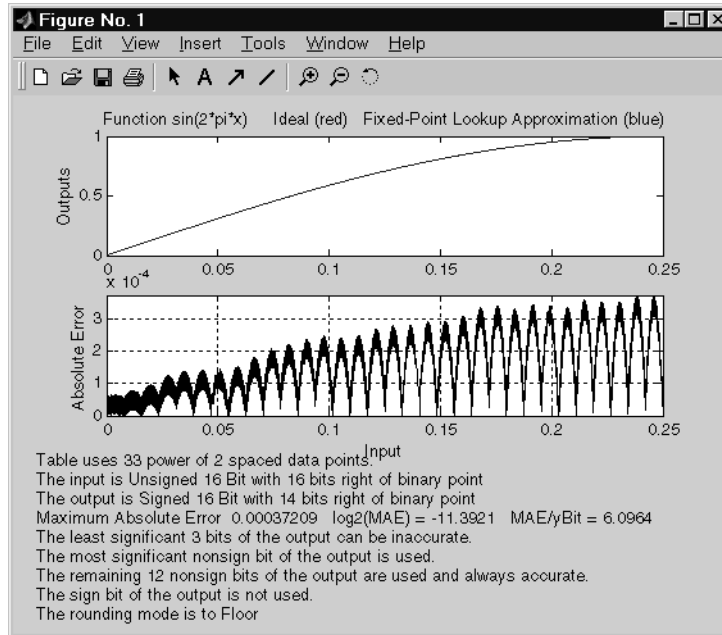
You can then plot the ideal function, the approximation, and the errors by typing

```
fixpt_look1_func_plot(xdata,ydata,funcstr,xmin,xmax,xdt,...  
xscale,ydt,yscale,rndmeth);
```

The `fixpt_look1_func_plot` function produces a plot of the fixed-point sine function, using these breakpoints, and a plot of the error between the ideal function and the fixed-point function. The maximum absolute error and the number of points required are listed with the plot. The error drops to zero at a breakpoint, and increases between breakpoints due to the difference in curvature of the ideal function and the line drawn between breakpoints.

The resulting plots are shown.





The lookup table requires 33 points to achieve a maximum absolute error of  $2^{-11.3922}$ .

## See Also

`fixpt_look1_func_plot`

# fixpt\_look1\_func\_plot

**Purpose** Plot a function with x values generated by the `fixpt_look1_func_approx` function

**Syntax** `errworst=fixpt_look1_func_plot(xdata,ydata,'funcstr',...  
xmin,xmax,xdt,xscale,ydt,yscale,rndmeth)`

**Description** `fixpt_look1_func_plot(xdata,ydata,'funcstr',xmin,xmax,xdt,xscale,ydt,yscale,rndmeth)` plots a lookup table approximation function and its error from the ideal function. You can use the `fixpt_look1_func_approx` function to generate `xdata` and `ydata`, the *x* and *y* data points for the lookup table. The function returns the maximum absolute error as a variable `errworst`. The inputs are as follows.

Input	Value
<code>xdata</code>	x values for the lookup table
<code>ydata</code>	y values for the lookup table
<code>'funcstr'</code>	Function of x
<code>xmin</code>	Minimum input of interest
<code>xmax</code>	Maximum input of interest
<code>xdt</code>	Data type of x
<code>xscale</code>	Scaling for the x values
<code>ydt</code>	Data type of y
<code>yscale</code>	Scaling for the y values
<code>rndmeth</code>	Rounding mode supported by the blockset: 'Toward Zero', 'Nearest', 'Floor', 'Ceiling'

The `fixpt_look1_func_approx` function applies the ideal function to the points in `xdata` to produce `ydata`. While this is the easiest way to generate `ydata`, you are not required to use these values for `ydata` as input for the `fixpt_look1_func_approx` function. Choosing different values for `ydata` can, in some cases, produce a lookup table with a smaller maximum absolute error.

See Chapter 6, “Tutorial: Producing Lookup Table Data” for a tutorial on using the function `fixpt_look1_func_plot`. For an example of the function, see the reference page for the `fixpt_look1_func_approx` function.

**See Also**

`fixpt_look1_func_approx`

# fixpt\_set\_all

---

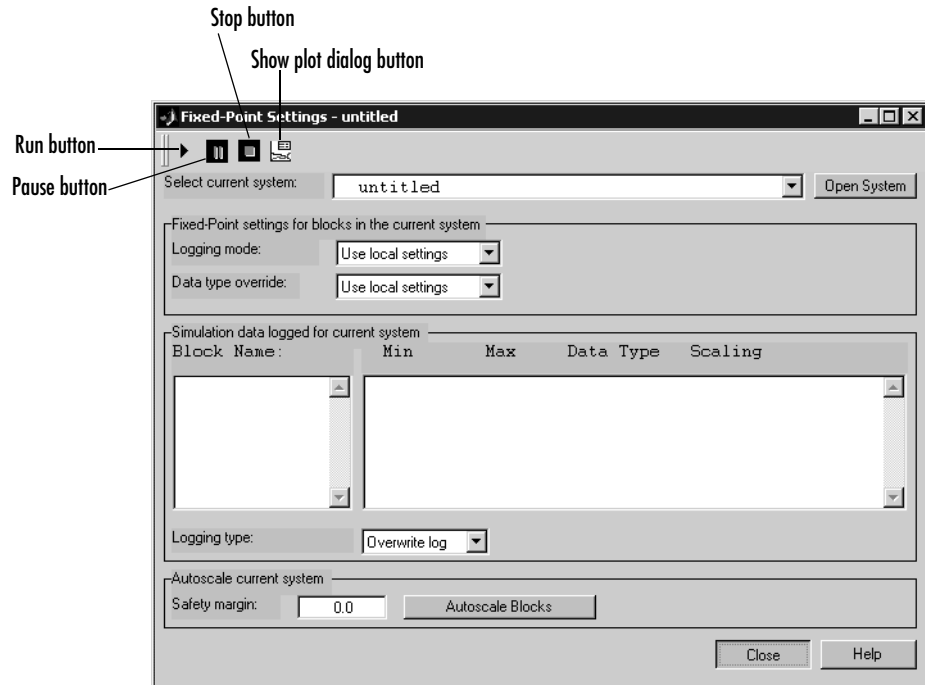
<b>Purpose</b>	Set a property for every fixed-point block in a subsystem
<b>Syntax</b>	<code>fixpt_set_all(SystemName,fixptPropertyName,fixptPropertyValue)</code>
<b>Description</b>	<code>fixpt_set_all</code> sets the property <code>fixptPropertyName</code> of every applicable block in the model or subsystem <code>SystemName</code> to the value <code>fixptPropertyValue</code> .
<b>Examples</b>	<p>To set every fixed-point block in a model called <code>Filter_1</code> to round toward the floor and to saturate upon overflow, type</p> <pre>fixpt_set_all('Filter_1','RndMeth','Floor') fixpt_set_all('Filter_1','DoSatur','on')</pre>

<b>Purpose</b>	Create a MATLAB structure describing a floating-point data type
<b>Syntax</b>	<pre>a = float('single') a = float('double') a = float(TotalBits, ExpBits)</pre>
<b>Description</b>	<p><code>float('single')</code> returns a MATLAB structure that describes the data type of an IEEE single (32 total bits, 8 exponent bits).</p> <p><code>float('double')</code> returns a MATLAB structure that describes the data type of an IEEE double (64 total bits, 11 exponent bits).</p> <p><code>float(TotalBits, ExpBits)</code> 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.</p> <p><code>float</code> is automatically called when a floating-point number is specified in a block dialog box.</p>
<b>Examples</b>	<p>Define a nonstandard, IEEE-style, floating-point data type with 31 total bits (excluding the hidden leading one) and 9 exponent bits:</p> <pre>a = float(31,9) a =     Class: 'FLOAT'   MantBits: 21    ExpBits: 9</pre>
<b>See Also</b>	<code>fixdt</code> , <code>sfix</code> , <code>sfrac</code> , <code>sint</code> , <code>ufix</code> , <code>ufrac</code> , <code>uint</code>

<b>Purpose</b>	Invokes the <b>Fixed-Point Settings</b> interface
<b>Syntax</b>	<code>fxptdlg('model')</code>
<b>Description</b>	<p><code>fxptdlg('model')</code> brings up the <b>Fixed-Point Settings</b> interface for the MDL-file <code>model</code>. You can also invoke this interface by</p> <ul style="list-style-type: none"><li>• Selecting <b>Fixed-Point Settings</b> in the <b>Tools</b> menu in the model window</li><li>• Right-clicking in any subsystem and selecting <b>Fixed-Point Settings</b> from the menu that pops up</li></ul> <p>With the Simulink Fixed Point product, the <b>Fixed-Point Settings</b> interface provides convenient access to global data type overrides and logging settings, the logged data, the automatic scaling script, and the Plot System interface. You can invoke the <b>Fixed-Point Settings</b> interface for any system or subsystem, and it controls the model specified by the <b>Select current system</b> parameter.</p> <p>If Simulink Fixed Point is installed, the <b>Fixed-Point Settings</b> interface displays the name, minimum simulation value, maximum simulation value, data type, and scaling of each block in the model that logs data. Additionally, if a signal saturates or overflows, a message is displayed for the associated block indicating how many times saturation or overflow occurred. You can display a block's dialog box by double-clicking on the appropriate block entry in this pane.</p> <p>Most of the functionality in the Fixed-Point Settings interface is for use with the Simulink Fixed Point product. However, even if you do not have Simulink Fixed Point, you can use the Fixed-Point Settings interface to perform a data type override that allows you to work with a fixed-point model.</p> <p>If you do not have Simulink Fixed Point, you can work with a model containing Simulink blocks with fixed-point settings by doing the following:</p> <ol style="list-style-type: none"><li>1 Access the <b>Fixed-Point Settings</b> interface from the model by selecting <b>Tools -&gt; Fixed-Point Settings</b>.</li><li>2 Set the <b>Logging mode</b> parameter to Force off model wide.</li><li>3 Set the <b>Data type override</b> parameter to True doubles or True singles model wide.</li></ol>

This procedure allows you to share fixed-point Simulink models among people in your company who may or may not have Simulink Fixed Point.

## Parameters and Dialog Box



### Select current system

Displays the names of all systems and subsystems in currently opened models in a hierarchical format. The menu can be expanded and collapsed using the + and - signs. The information displayed in the rest of the **Fixed-Point Settings** interface applies to the subsystem designated by this parameter.

### Logging mode

Controls which blocks log data. The value of this parameter for parent systems controls logging for all child subsystems, unless `Use local settings` is selected:

- **Use local settings** — Data is logged according to the value of this parameter set for each subsystem. Otherwise, settings for parent systems always override those of child systems.
- **Min, max and overflow** — Minimum value, maximum value, and overflow data is logged for all blocks in the current system or subsystem.
- **Overflow** — Only overflow data is logged for all blocks in the current system or subsystem.
- **Force off** — No data is logged 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.

## Data type override

Controls data type override of blocks that allow you to specify data types in their block masks. 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** — Data types are overridden according to the value of this parameter set for each subsystem. Otherwise, settings for parent systems override those of child systems.
- **Scaled doubles** — The output data type of all blocks in the current system or subsystem is overridden with doubles; however, the scaling and bias specified in the mask of each block is maintained.
- **True doubles** — The output data type of all blocks in the current system or subsystem is overridden with true doubles. The overridden values have no scaling or bias.
- **True singles** — The output data type of all blocks in the current system or subsystem is overridden with true singles. 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.

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.



---

**Note** The following Simulink blocks allow you to set data types in their block masks, but ignore the **Data Type Override** setting: Probe, Trigger, Width.

---

### Block Name

Displays blocks that log data in the selected system or subsystem. The block path is described in terms of the blockset model name. The minimum value, maximum value, data type, and scaling are shown opposite each block name when the simulation is run.

### Logging type

Controls the logging type:

- **Overwrite log** — Information in the **Simulation data logged for current system** pane is completely cleared before new logging data is entered.
- **Merge log** — New logging data is merged with any information previously appearing in the **Simulation data logged for current system** pane.

### Safety margin

The **Safety margin** parameter is used as part of the automatic scaling procedure. Before automatic scaling is performed, you must run the simulation to collect min/max data. To learn how to do this, refer to Chapter 5, “Tutorial: Feedback Controller Simulation.”

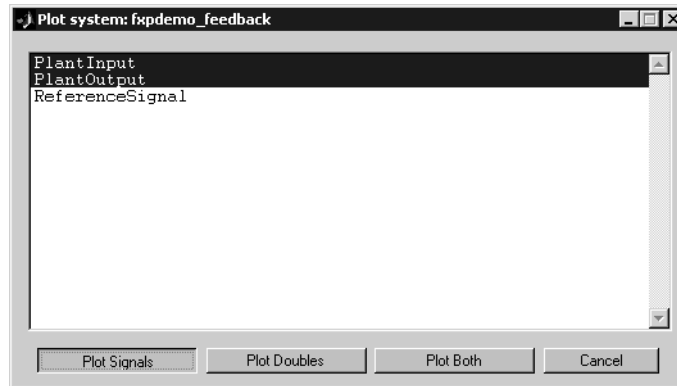
Simulation values are multiplied by the factor designated by this parameter, allowing you to specify a range different from that defined by the maximum and minimum values logged to the workspace. For example, a value of 55 specifies that a range *at least* 55 percent larger is desired. A value of -15 specifies that a range *up to* 15 percent smaller is acceptable.

The **Fixed-Point Settings** interface contains eight buttons:

- **Run** runs the model and updates the display with the latest simulation information.
- **Pause** pauses the simulation.
- **Stop** stops the simulation from running.

- **Show plot dialog** invokes the **Plot systems** interface, which displays any To Workspace, Outport, or Scope blocks found in the model.
- **Open System** invokes the **Fixed-Point Settings** interface for the system or subsystem displayed in the **Select current system** parameter.
- **Autoscale Blocks** invokes the automatic scaling script `autofixexp`.
- **Close** closes the interface.
- **Help** displays the HTML-based help for the `fxptdlg` function.

The **Plot systems** interface is shown below. In this example, the interface is displaying variable names that correspond to Scope block outputs from the `fxpdemo_feedback` demo.



To plot the simulation results, select one or more variable names, and then select the appropriate plot button:

- **Plot Signals** plots the raw signal data for the selected variable(s).
- **Plot Doubles** plots doubles data for the selected variable(s). Doubles are generated when the **Data type override** parameter is set to `True doubles`.
- **Plot Both** plots both raw signal data and doubles data for the selected signal(s). Note that the doubles override does not overwrite the raw data.
- **Cancel** allows you to exit the interface without plotting.

## Examples

To learn how to use the **Fixed-Point Settings** interface, refer to “Tutorial: Feedback Controller Simulation” in the Simulink Fixed Point documentation.

## See Also

`showfixptsimerrors`, `showfixptsimranges`

# num2fixpt

---

## Purpose

Quantize a value using a Simulink Fixed Point representation

## Syntax

```
outValue = num2fixpt(OrigValue,FixPtDataType,FixPtScaling,...  
    RndMeth, DoSatur)
```

## Description

num2fixpt(OrigValue,FixPtDataType,FixPtScaling, RndMeth, DoSatur) casts a real-world value represented in floating-point doubles, OrigValue, as a fixed-point number, outValue.

OrigValue      Identifies the real-world value to be cast to a fixed-point value.

FixPtDataType   Designates the desired fixed-point data type of outValue.

FixPtScaling    Indicates the scaling of the output in either Slope or [Slope Bias] format.

RndMeth          Specifies the rounding technique to be used on the output. If FixPtDataType is FLOAT, then RndMeth is ignored.

DoSatur          Indicates whether the output should be saturated to the minimum or maximum representable value upon underflow or overflow. If FixPtDataType is FLOAT, then DoSatur is ignored.

## Examples

The command

```
num2fixpt(pi,sfix(8),2^-5,'Nearest','on')
```

returns Pi as a signed 8-bit fixed-point number with scaling of  $2^{-5}$ . Rounding is toward the nearest representable value, and overflows saturate.

## See Also

fixptbestexp, fixptbestprec

<b>Purpose</b>	Create a MATLAB structure describing a signed generalized fixed-point data type
<b>Syntax</b>	<code>a = sfix(TotalBits)</code>
<b>Description</b>	<p><code>sfix(TotalBits)</code> returns a MATLAB structure that describes the data type of a signed generalized fixed-point number with a word size given by <code>TotalBits</code>.</p> <p><code>sfix</code> is automatically called when a signed generalized fixed-point data type is specified in a block dialog box.</p>

---

**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`

# sfrac

---

<b>Purpose</b>	Create a MATLAB structure describing a signed fractional data type
<b>Syntax</b>	<pre>a = sfrac(TotalBits) a = sfrac(TotalBits, GuardBits)</pre>
<b>Description</b>	<p><code>sfrac(TotalBits)</code> returns a MATLAB structure that describes the data type of a signed fractional number with a word size given by <code>TotalBits</code>.</p> <p><code>sfrac(TotalBits, GuardBits)</code> returns a MATLAB structure that describes the data type of a signed fractional number. The total word size is given by <code>TotalBits</code> with <code>GuardBits</code> bits located to the left of the sign bit.</p> <p><code>sfrac</code> is automatically called when a signed fractional data type is specified in a block dialog box.</p> <p>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 in addition to the sign bit.</p>
<b>Examples</b>	<p>Define an 8-bit signed fractional data type with 4 guard bits. Note that the range of this number is <math>-2^4 = -16</math> to <math>(1 - 2^{(1-8)}) \cdot 2^4 = 15.875</math>:</p> <pre>a = sfrac(8,4) a =     Class: 'FRAC'   IsSigned: 1    MantBits: 8   GuardBits: 4</pre>
<b>See Also</b>	<code>fixdt</code> , <code>float</code> , <code>sfix</code> , <code>sint</code> , <code>ufix</code> , <code>ufrac</code> , <code>uint</code>

<b>Purpose</b>	Create a MATLAB structure describing a signed integer data type
<b>Syntax</b>	<code>a = sint(TotalBits)</code>
<b>Description</b>	<p><code>sint(TotalBits)</code> returns a MATLAB structure that describes the data type of a signed integer with a word size given by <code>TotalBits</code>.</p> <p><code>sint</code> is automatically called when a signed integer is specified in a block dialog box.</p> <p>The default binary point for this data type is assumed to lie to the right of all bits.</p>
<b>Examples</b>	<p>Define a 16-bit signed integer data type:</p> <pre>a = sint(16) a =     Class: 'INT'   IsSigned: 1   MantBits: 16</pre>
<b>See Also</b>	<code>fixdt</code> , <code>float</code> , <code>sfix</code> , <code>sfrac</code> , <code>ufix</code> , <code>ufrac</code> , <code>uint</code>

# ufix

---

<b>Purpose</b>	Create a MATLAB structure describing an unsigned generalized fixed-point data type
<b>Syntax</b>	<code>a = ufix(TotalBits)</code>
<b>Description</b>	<p><code>ufix(TotalBits)</code> returns a MATLAB structure that describes the data type of an unsigned generalized fixed-point data type with a word size given by <code>TotalBits</code>.</p> <p><code>ufix</code> is automatically called when an unsigned generalized fixed-point data type is specified in a block dialog box.</p>

---

**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.

---

<b>Examples</b>	<p>Define a 16-bit unsigned generalized fixed-point data type:</p> <pre>a = ufix(16) a =     Class: 'FIX'     IsSigned: 0     MantBits: 16</pre>
-----------------	--

<b>See Also</b>	<code>fixdt</code> , <code>float</code> , <code>sfix</code> , <code>sfrac</code> , <code>sint</code> , <code>ufrac</code> , <code>uint</code>
-----------------	---



**Purpose** Create a MATLAB structure describing an 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  $(1 - 2^{-8}) \cdot 2^4 = 15.9375$ :

```
a = ufrac(8,4)
a =
    Class: 'FRAC'
   IsSigned: 0
   MantBits: 8
  GuardBits: 4
```

**See Also** `fixdt`, `float`, `sfix`, `sfrac`, `sint`, `ufix`, `uint`

# uint

---

<b>Purpose</b>	Create a MATLAB structure describing an unsigned integer data type
<b>Syntax</b>	<code>a = uint(TotalBits)</code>
<b>Description</b>	<p><code>uint(TotalBits)</code> returns a MATLAB structure that describes the data type of an unsigned integer with a word size given by <code>TotalBits</code>.</p> <p><code>uint</code> is automatically called when an unsigned integer is specified in a block dialog box.</p> <p>The default binary point for this data type is assumed to lie to the right of all bits.</p>
<b>Examples</b>	<p>Define a 16-bit unsigned integer:</p> <pre>a = uint(16) a =     Class: 'INT'    IsSigned: 0   MantBits: 16</pre>
<b>See Also</b>	<code>fixdt</code> , <code>float</code> , <code>sfix</code> , <code>sfrac</code> , <code>sint</code> , <code>ufix</code> , <code>ufrac</code>

# Data Object Classes

---

This section describes the properties and usage of the following classes of Simulink data objects (see “Working with Data Objects” for general information on creating and using Simulink data objects).

Class Summary (p. 8-2)

Brief description of the purpose of each class.

Classes — Alphabetical List (p. 8-3)

Classes listed alphabetically.

# Class Summary

The following table briefly describes the purpose of each Simulink data object class..

Class	Purpose
Simulink.AliasType	Specifies an alternate name for an existing data type.
Simulink.Bus	Describes a signal bus.
Simulink.ModelDataLogs	Stores a model’s signal logs.
Simulink.ModelWorkspace	Accesses a model’s workspace.
Simulink.NumericType	Describes a numeric data type.
Simulink.Parameter	Describes the value of a block parameter.
Simulink.Signal	Describes the value of a block output.
Simulink.StructElement	Describes an element of a data structure.
Simulink.StructType	Describes a data structure.
Simulink.SubsysDataLogs	Stores a subsystem’s signal logs.
Simulink.Timeseries	Log for an elementary signal.
Simulink.TsArray	Log for a composite signal.

## Classes — Alphabetical List

Simulink.AliasType  
Simulink.Bus  
Simulink.BusElement  
Simulink.ModelDataLogs  
Simulink.ModelWorkspace  
Simulink.NumericType  
Simulink.Parameter  
Simulink.ParamRTWInfo  
Simulink.Signal  
Simulink.StructElement  
Simulink.StructType  
Simulink.SubsysDataLogs  
Simulink.Timeseries  
Simulink.TsArray

# Simulink.AliasType

---

## Purpose

Create an alias for a signal and/or parameter data type

## Description

This class allows you to designate MATLAB variables as aliases for signal and parameter data types. You do this by creating instances of this class and assigning them to variables in the MATLAB or model workspaces (see “Creating a Data Type Alias” on page 8-4). The MATLAB variable to which a Simulink.AliasType object is assigned is called a data type alias. The data type to which an alias refers is called its base type. Simulink allows you to set the BaseType property of the object that the variable references, thereby designating the data type for which it is an alias.

Simulink lets you use aliases instead of actual type names in dialog boxes and set\_param commands to specify the data types of Simulink block outputs and parameters. Using aliases to specify signal and parameter data types can greatly simplify global changes to the signal and parameter data types that a model specifies. In particular, changing the data type of all signals and parameters whose data type is specified by an alias requires only changing the base type of the alias. By contrast, changing the data types of signals and parameters whose data types are specified by an actual type name requires respecifying the data type of each signal and parameter individually, a tedious and time-consuming process.

## Creating a Data Type Alias

You can use either the Model Explorer or MATLAB commands (see “MATLAB Commands for Creating Data Type Aliases” on page 8-5) to create a data type alias.

To use the Model Explorer to create an alias,

- 1 Select the MATLAB workspace in the Model Explorer’s **Model Hierarchy** pane.  
You must create data type aliases in the MATLAB workspace. If you attempt to create an alias in a model workspace, Simulink displays an error.
- 2 Select **Simulink.AliasType** from the Model Explorer’s **Add** menu.  
Simulink creates an instance of a Simulink.AliasType object and assigns it to a variable named Alias in the MATLAB workspace.

- 3 Rename the variable to a more appropriate name, for example, a name that reflects its intended usage.

To change the name, edit the name displayed in the **Name** field in the Model Explorer's **Contents** pane.

- 4 Enter the name of the data type that this alias represents in the **Base type** field in the Model Explorer's **Dialog** pane.

You can specify the name of any existing standard or user-defined data type in this field. Skip this step if the desired base type is `double` (the default).

- 5 Save the newly created alias in a MAT-file loaded by the models in which it is used.

## MATLAB Commands for Creating Data Type Aliases

Use the following syntax to create a data type alias at the MATLAB command line or in a MATLAB program

```
ALIAS = Simulink.Alias;
```

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.Alias;
```

The following notations get and set the properties of a data type alias, respectively,

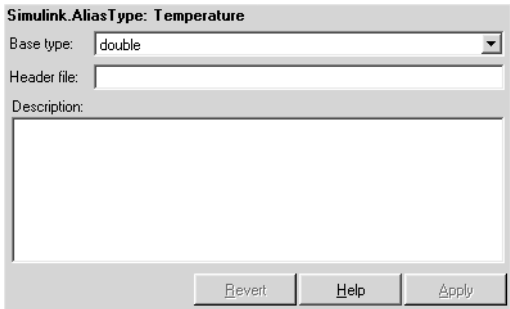
```
PROPVALUE = ALIAS.PROPNAME;  
ALIAS.PROPNAME = PROPVALUE;
```

where ALIAS is the name of the alias, PROPNAME is the name of the alias object's properties, and PROPVALUE is the property's value. For example, the following code saves the current value of `MyFloat`'s `BaseType` property and assigns it a new value.

```
save = MyFloat.BaseType;  
MyFloat.BaseType = 'single';
```

See the "Properties" section below for information on the names, permitted values, and usage of the properties of data type alias objects.

## Property Dialog Box



### Base type

The data type to which this alias refers. The default is `double`. To specify another data type, select the data type from the adjacent pull-down list of standard data types or enter the data type's name in the edit field. Note that you can, with one exception, specify a nonstandard data type, e.g, a data type defined by a `Simulink.NumericType` object, by entering the data type's name in the edit field. The exception is a `Simulink.NumericType` whose Category is Fixed-point: unspecified scaling.

### Header file

Name of a user-supplied C header file that defines a data type having the same name as this alias (i.e., as the MATLAB variable that references this alias object). If this field is not empty, code generated from this model defines the alias type by including the specified header file. If this field is empty, the generated code defines the alias type itself.

### Description

Describes the usage of the data type referenced by this alias.

## Properties

Name	Description
BaseType	A string specifying the name of a standard or custom data type. ( <b>Base type</b> )



Name	Description
Description	A string that describes the usage of the data type. May be a null string. ( <b>Description</b> )
HeaderFile	A string that specifies the name of a C header file that defines a data type having the same name as the alias. ( <b>Header file</b> )

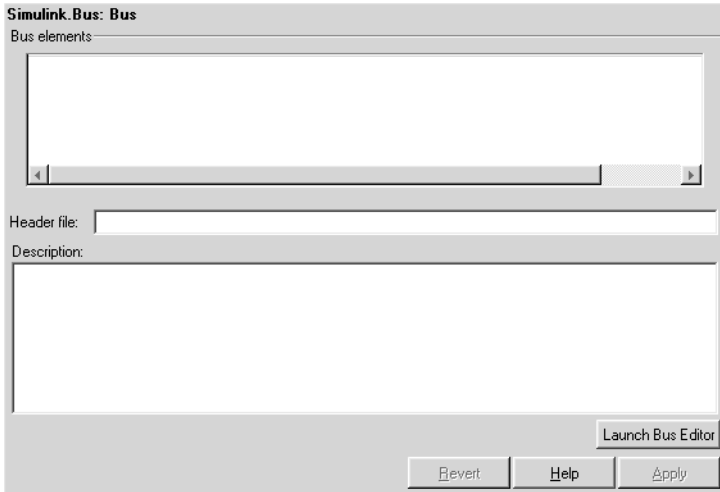
# Simulink.Bus

**Purpose** Specify the properties of a signal bus

**Description** Objects of this class (in conjunction with objects of the `Simulink.BusElement` class) specify the properties of a signal bus. You can use these objects to enable Simulink to check the validity of buses connected to the inputs of blocks in your model. You do this by entering the name of a bus object defining a bus in the **Bus object** field of a block’s parameter dialog box. When you update the model’s diagram or start a simulation of the model, Simulink checks whether the buses connected to the blocks have the properties specified by the bus objects. If not, Simulink halts and displays an error message.

You can use the Model Explorer (see “The Model Explorer”), the Simulink Bus Editor (see “Bus Editor”), or MATLAB commands (see “Working with Data Objects”) to create bus objects. You must use the Bus Editor or the MATLAB command line to set the properties of a bus object.

## Property Dialog Box



### Bus elements

Table that displays the properties of the bus’s elements. You cannot edit this table. You must use either the Simulink **Bus Editor** (see “Bus Editor” in the online Simulink documentation) 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. Simulink ignores it.

**Description**

Description of this structure. This field is intended for you to use to document this bus. Simulink itself does not use this field.

**Properties**

Name	Access	Description
Description	RW	String that describes this bus. This property is intended for user use. Simulink itself does not use it. ( <b>Description</b> )
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. ( <b>Bus elements</b> )
HeaderFile	RW	String that specifies the name of a C header file that declares the structure of this bus. This property is intended for use by Real-Time Workshop. Simulink does not use it. ( <b>Header file</b> )

**See Also**

Simulink.BusElement

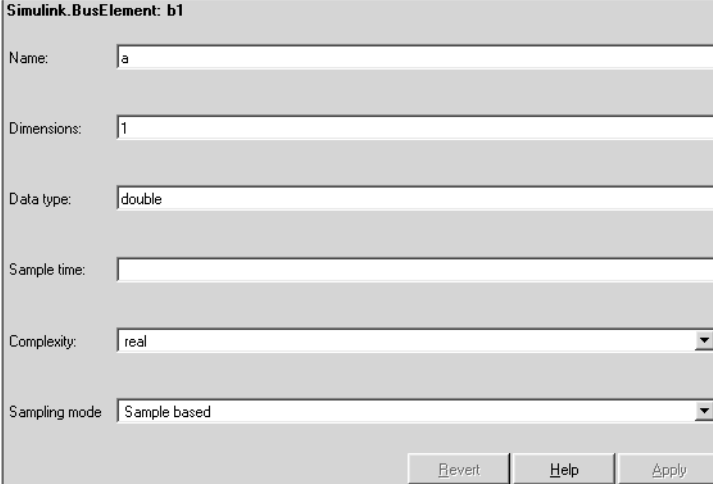
# Simulink.BusElement

---

**Purpose** Describe an element of a signal bus

**Description** Objects of this class describe elements of buses described by objects of the `Simulink.Bus` class.

## Property Dialog Box



The image shows a MATLAB/Simulink property dialog box for a `Simulink.BusElement` object named `b1`. The dialog has a title bar that says "Simulink.BusElement: b1". It contains several input fields and dropdown menus:

- Name:** A text field containing the value "a".
- Dimensions:** A text field containing the value "1".
- Data type:** A text field containing the value "double".
- Sample time:** An empty text field.
- Complexity:** A dropdown menu with "real" selected.
- Sampling mode:** A dropdown menu with "Sample based" selected.

At the bottom right of the dialog are three buttons: "Evert", "Help", and "Apply".

**Name**  
Name of the element.

**Dimensions**  
A vector specifying the dimensions of the element.

**Data type**  
Name of the data type of this element. The value of this field can, with one exception, be a `Simulink.NumericType` object. The exception is a `Simulink.NumericType` whose `Category` is `Fixed-point: unspecified scaling`. The value of this field can also be the name of a base workspace variable that references a `Simulink.Bus` object. This allows you to create bus objects that specify hierarchical buses, i.e., buses that contain other buses.

**Sample time**  
Size of the interval between times when this signal's value must be recomputed. See "Specifying Sample Time" for more information.

Complexity

Numeric type (i.e., real or complex) of this element.

Sampling mode

Sampling mode of this element. This field is intended to be used by applications based on Simulink. Simulink itself ignores it.

Properties

Name	Access	Description
Complexity	RW	String that specifies the numeric type ('real' or 'complex') of this element. ( <b>Complexity</b> )
DataType	RW	String that specifies the name of the data type of this element or the name of a bus object to specify an element that is itself a bus. ( <b>Data type</b> )
Dimensions	RW	A vector specifying the dimensions of this element. ( <b>Dimensions</b> )
Name	RW	String specifying the name of this element. ( <b>Name</b> )
SampleTime	RW	Scalar or vector that specifies the sample time of this bus element. ( <b>Sample time</b> )
SamplingMode	RW	String that specifies the sampling mode of this element. ( <b>Sampling mode</b> )

See Also

Simulink.Bus

# Simulink.ModelDataLogs

---

**Purpose** Container for a model's signal data logs.

**Description** Simulink creates instances of this class to contain signal logs that it creates while simulating a model (see “Logging Signals”). In particular, Simulink creates an instance of this class for a top-level model and for each model referenced by the top-level model that contains signals to be logged. Simulink assigns the ModelDataLogs object for the top-level model to a variable in the MATLAB workspace. The name of the variable is the name specified in the **Signal logging name** field on the **Data Import/Export** pane of the model's **Configuration Parameters** dialog box. The default value is `logout`.

A ModelDataLogs object has a variable number of properties. The first property, named `Name`, specifies the name of the model whose signal data the object contains or, if the model is a referenced model, the name of the Model block that references the model. The remaining properties reference objects that contain signal data logged during simulation of the model. The objects may be instances of any of the following types of objects:

- `Simulink.Timeseries`  
Log for a signal in this model.
- `Simulink.TsArray`  
Container for the logs of the elements of a root-level composite signal (e.g., a Mux or Bus Creator signal) in this model.
- `Simulink.ModelDataLogs`  
Container for the logs of a model referenced by this model.
- `Simulink.SubsysDataLogs`  
Container for the signal logs of a subsystem of this model.

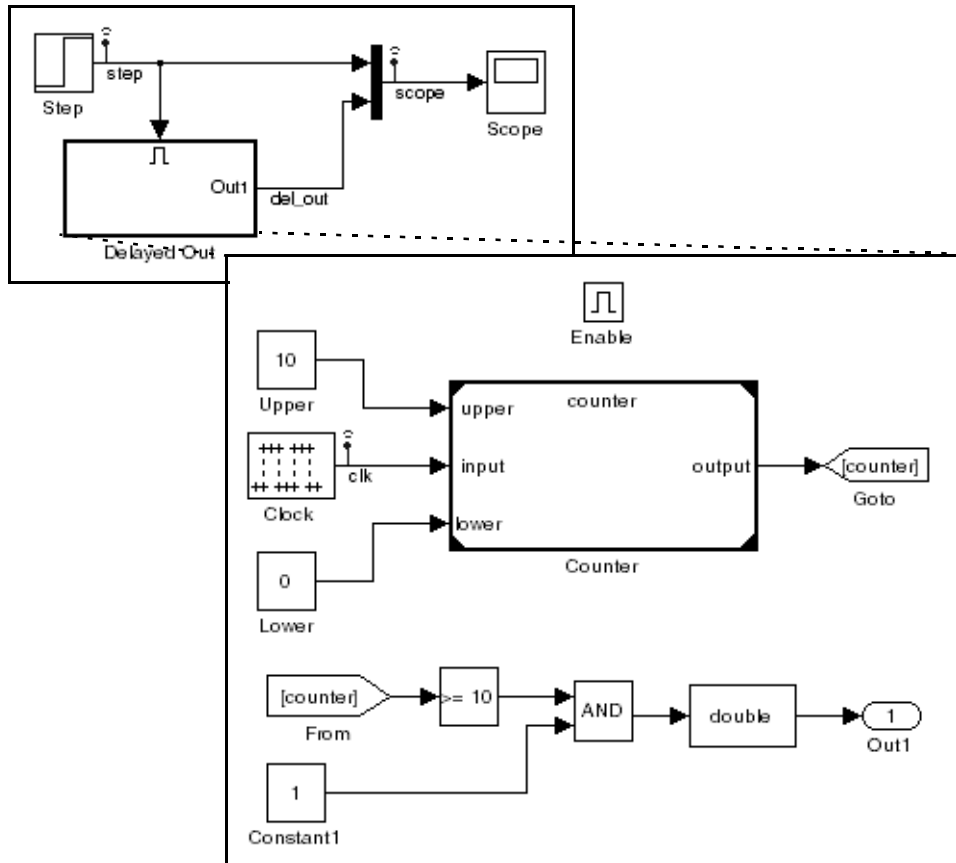
The names of the properties are the log names of the signals in the cases of signals or of a subsystem or a referenced model in the case of objects that contain data for a subsystem or referenced model.

---

**Note** If a name contains spaces, the ModelDataLogs objects specifies its name as `('name')` where **name** is the actual name, e.g., `('Brake Subsystem')`.

---

Consider, for example, the following model.



As indicated by the testpoint icons, this model specifies that Simulink should log the signals named `step` and `scope` in the model's root system and the signal named `clk` in the subsystem named `Delayed Out`. After simulation of this model, the MATLAB workspace contains the following variable:

```
>> logout
```

```
logout =
```

```
Simulink.ModelDataLogs (siglgex):
```

# Simulink.ModelDataLogs

---

Name	Elements	Simulink Class
scope	2	TsArray
step	1	Timeseries
('Delayed Out')	2	SubsysDataLogs

The logsout variable contains the signal data logged during the simulation. You can use fully qualified object names or the Simulink unpack command to access the signal data stored in logsout. For example, to access the amplitudes of the clk signal in the Delayed Out subsystem, enter

```
>> data = logsout.('Delayed Out').clk;
```

or

```
>> logsout.unpack('all');  
>> data = clk;
```

**See Also**

Simulink.Timeseries, Simulink.TsArray, Simulink.SubsysDataLogs, unpack



**Purpose** Describe a model workspace.

**Description** Instances of this class describe model workspaces. Simulink creates an instance of this class for each model that you open during a Simulink session. See “Working with Model Workspaces” for more information.

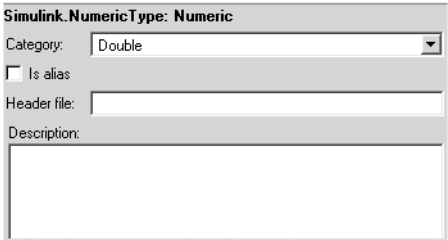
Properties	Name	Access	Description
	DataSource	RW	Specifies the source used to initialize this workspace. Valid values are <ul style="list-style-type: none"><li>• 'MDL-File'</li><li>• 'MAT-File'</li><li>• 'M-Code'</li></ul>
	FileName	RW	Specifies the name of the MAT-file used to initialize this workspace. Simulink ignores this property if DataSource is not 'MAT-File'.
	MCode	RW	A string specifying M code used to initialize this workspace. Simulink ignores this property if DataSource is not 'M-Code'.

# Simulink.NumericType

**Purpose** Specify a data type

**Description** This class lets you specify a data type. To do this, create an instance of this class in the MATLAB workspace and set its properties to the properties of the custom data type. Then assign this data type to all signals and parameters of your model that you want to conform to the data type. Assigning the data type in this way allows you to change the data types of the signals and parameters in your model by changing the properties of the object that describe them. You do not have to change the model itself.

**Property  
Dialog Box**



**Category** Category to which this data type belongs. 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 to disable other controls on the dialog box (see below) that apply to the category and to disable controls that do not apply. Selecting a fixed-point category may, depending on the other dialog box options that you select, cause the model to run only on systems that have a Simulink Fixed Point option installed.

## Is alias

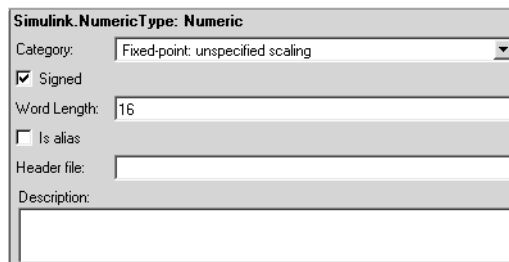
If this option is selected, Simulink uses the name of the workspace variable that references this object as the name of the data type. Otherwise, Simulink uses the category of the data type as its name, or, if the category is a fixed-point category, Simulink generates a name that encodes the type's properties, using the encoding specified by the Simulink Fixed Point product.

## Header file

Storage class of this parameter. See “Storage Classes of Tunable Parameters” in the Real-Time Workshop documentation for more information.

## Description

Description of this data type. This field is intended for use in documenting this data type. Simulink ignores it.

The image shows a dialog box titled "Simulink.NumericType: Numeric". It contains several fields: "Category:" with a dropdown menu showing "Fixed-point: unspecified scaling"; "Signed" with a checked checkbox; "Word Length:" with a text field containing "16"; "Is alias" with an unchecked checkbox; "Header file:" with an empty text field; and "Description:" with a large empty text area.

## Signed

Specifies whether the data type is signed or unsigned. This option is enabled only for fixed-point data type categories.

## Word Length

Word length in bits of the fixed-point data type. This option is enabled only for fixed-point data type categories.

Simulink.NumericType: Numeric

Category:

Fixed-point: binary point scaling

☒ Signed

Word Length:

16

Fraction Length:

0

☐ Is alias

Header file:

Description:

## 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.

Simulink.NumericType: Numeric

Category:

Fixed-point: slope and bias scaling

☒ Signed

Word Length:

16

Slope:

$2^0$

Bias:

0

☐ Is alias

Header file:

Description:

## 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.

## 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. ( <b>Bias</b> )
Category	RW	String that specifies the category of this data type. Valid values are 'Double', 'Boolean', 'Single', 'Fixed-point: unspecified scaling', 'Fixed-point: binary point scaling', and 'Fixed-point: slope and bias scaling'. ( <b>Category</b> )
Description	RW	Description of this data type. ( <b>Description</b> )
FixedExponent		Exponent used for binary point scaling. This property equals -FractionLength. Setting this property causes Simulink to set the FractionLength and Slope properties accordingly, and vice versa. This property appears only if the data type category is Fixed-point: binary point scaling or Fixed-point: slope and bias scaling.
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 to set the FixedExponent property accordingly, and vice versa. This field is intended for use by the Simulink Fixed Point product. ( <b>Fraction length</b> )
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). ( <b>Is alias</b> )

# Simulink.NumericType

Name	Access	Description
Signed	RW	Integer that specifies whether this data type is signed or unsigned. Valid values are 1 (yes) or 0 (no). ( <b>Signed</b> )
Slope		Slope for slope and bias scaling of fixed-point numbers. This property equals $\text{SlopeAdjustmentFactor} * 2^{\text{FixedExponent}}$ . If $\text{SlopeAdjustmentFactor}$ is 1.0, Simulink displays the value of this field as $2^{\text{SlopeAdjustmentFactor}}$ . Otherwise, it displays it as a numeric value. Setting this property causes Simulink to set the $\text{FixedExponent}$ and $\text{SlopeAdjustmentFactor}$ properties accordingly, and vice versa. This property appears only if $\text{Category}$ is Fixed-point: slope and bias scaling. ( <b>Slope</b> )
SlopeAdjustment Factor	RW	Slope for slope and bias scaling of fixed-point numbers. Setting this property causes Simulink to adjust the $\text{Slope}$ property accordingly, and vice versa. This property appears only if $\text{Category}$ is Fixed-point: slope and bias scaling.
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 $\text{Category}$ is Fixed-point. ( <b>Word Length</b> )

<b>Purpose</b>	Specify the value, value range, data type, and other properties of a block parameter.
<b>Description</b>	This class enables you to create workspace objects that you can then use as the values of block parameters, for example, as the value of a Gain block's Gain parameter. The advantage? 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 make use of this information. For example, Simulink and Real-Time Workshop use information specified by Simulink.Parameter objects to determine whether the parameter is tunable (see "Tuning Parameters").

**Property  
Dialog Box**

Simulink.Parameter: gain

Value:

Data type:

Units:

Dimensions:

Complexity:

Minimum:

Maximum:

Code generation options

Storage class:

Alias:

Description:

Revert

Help

Apply

<b>Value</b>	Value of the parameter. You can use MATLAB expressions to specify the numeric type, dimensions, and data type of the parameter, e.g., <code>single([2+5i -1+3.14159i])</code> .
<b>Data type</b>	Data type of the parameter. Simulink determines the data type from the entry in the parameter's <b>Value</b> field. You cannot set this field yourself.
<b>Units</b>	

Measurement units in which this value is expressed, e.g., inches. This field is intended for use in documenting this parameter. Simulink ignores it.

### Dimensions

Dimensions of the parameter. Simulink determines the dimensions from the entry in the **Value** field of this parameter. You cannot set this field yourself.

### Complexity

Numeric type (i.e., real or complex) of the parameter. Simulink determines the numeric type of this parameter from the entry in the **Value** field of this parameter. You cannot set this field yourself.

### Minimum

Minimum value that the parameter can have when tuned. This field is intended for use in documenting this parameter. Simulink ignores it.

### Maximum

Maximum value that the parameter can have. This field is intended for use in documenting this parameter. Simulink ignores it.

### Storage class

Storage class of this parameter. See “Storage Classes of Tunable Parameters” in the Real-Time Workshop documentation for more information.

### Alias

Alternate name for this parameter. Simulink ignores this setting.

### Description

Description of this parameter. This field is intended for use in documenting this parameter. Simulink ignores it.

## Properties

Name	Access	Description
Value	RW	Value of this parameter. ( <b>Value</b> )
DataType	RO	String specifying the name of the data type of this parameter. This field is read-only. ( <b>Data type</b> )



Name	Access	Description
Dimensions	RO	Vector specifying the dimensions of this parameter. ( <b>Dimensions</b> )
Complexity	RO	String specifying the numeric type of this paramter. Valid values are 'real' or 'complex'. ( <b>Complexity</b> )
Min	RW	Minimum value that this parameter can have. Simulink ignores this property. ( <b>Minimum</b> )
Max	RW	Maximum value that this parameter can have. Simulink ignores this property. ( <b>Maximum</b> )
DocUnits	RW	Measurement units in which this parameter's value is expressed. ( <b>Units</b> )
RTWInfo	RW	Information used by Real-Time Workshop for generating code for this parameter. The value of this property is an object of Simulink.ParamRTWInfo class.

# Simulink.ParamRTWInfo

**Purpose** Specify information needed to generate code for a parameter

**Description** Simulink creates an instance of this class for each instance of a Simulink.Parameter object that it creates. Simulink uses the Simulink.ParamRTWInfo object to store information needed to generate code for the parameter specified by the Simulink.Parameter object.

You can set the properties of an instance of this class via the RTWInfo property or the property dialog box of the Simulink.Parameter object that uses it. For example, the following MATLAB expression sets the StorageClass property of a Simulink.ParamRTWInfo object used by a parameter object name gain.

```
gain.RTWInfo.StorageClass = 'ExportedGlobal';
```

**Property Dialog Box** Use the Simulink.Parameter property dialog box to set the StorageClass and Alias properties of objects of this class. You must use MATLAB commands to set the other properties.

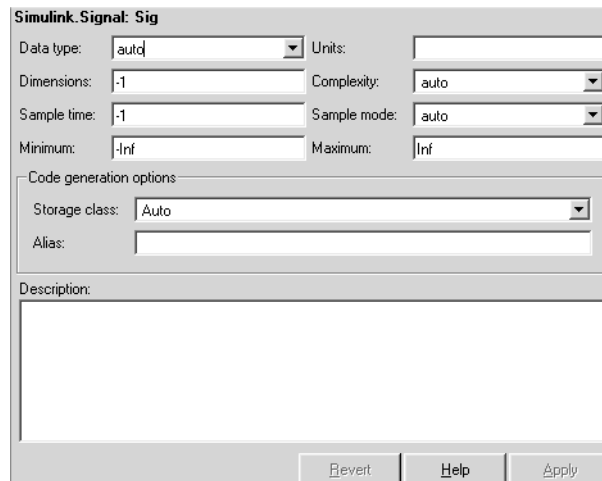
Properties	Name	Description
	Alias	Alternate name for this parameter.
	CustomAttributes	Custom storage class attributes of this parameter. See “Custom Storage Classes” in the Real-Time Workshop Embedded Coder documentation for more information.
	CustomStorageClass	Custom storage class of this parameter.
	StorageClass	Storage class of this parameter. See “Storage Classes of Tunable Parameters” in the Real-Time Workshop documentation for more information.

**Purpose** Specify the attributes of a signal

**Description** Objects of this class allow you to specify the attributes that a signal or discrete state should have, e.g., its data type, numeric type, dimensions, and so on. You do this by giving the signal or discrete state the same name as the base (MATLAB) workspace variable that references the `Simulink.Signal` object. Then, whenever Simulink updates the diagram containing the signal or state, Simulink checks the actual properties of the signal against the properties specified by the signal object. For most properties, if the actual and specified properties are not the same, Simulink displays an error message. This enables you to quickly determine whether the actual attributes of your model's signals are the attributes you intend them to have.

In the case of a signal's storage class property, Simulink does not display an error if the model specifies the signal's storage class as `auto` (i.e., determined by Simulink) and the associated `Simulink.Signal` object specifies a different value. In this case, the value specified by the object overrides the value specified by the model. This allows you to use `Simulink.Signal` objects to control the visibility of signals (i.e., whether they are test points) from the MATLAB workspace (see “Using Signal Objects to Designate Test Points”).

## Property Dialog Box



The dialog box is titled "Simulink.Signal: Sig". It contains several input fields and dropdown menus for configuring signal properties. The fields are arranged in a grid-like fashion. At the bottom, there are three buttons: "Revert", "Help", and "Apply".

Property	Value
Data type	auto
Units	
Dimensions	-1
Complexity	auto
Sample time	-1
Sample mode	auto
Minimum	-Inf
Maximum	Inf
Storage class	Auto
Alias	

Description:

Revert Help Apply

**Data type**

Data type of the signal. The default entry, `auto`, specifies that Simulink, should determine the data type. Use the adjacent pull-down list to specify built-in data types (e.g., `uint8`). To specify a custom data type, enter a MATLAB expression that specifies the type, e.g., a base workspace variable that references a `Simulink.NumericType` object or a Simulink data type function (e.g., `sint(16)`) that returns a MATLAB structure that defines the custom data type.

**Units**

Measurement units in which the value of this signal is expressed, e.g., inches. This field is intended for use in documenting this signal. Simulink ignores it.

**Dimensions**

Dimensions of this signal. Valid values are `-1` (the default) specifying any dimensions, `N` specifying a vector signal of size `N`, or `[M N]` specifying an `MxN` matrix signal.

**Complexity**

Numeric type of the signal. Valid values are `auto` (determined by Simulink), `real`, or `complex`.

**Sample time**

Rate at which the value of this signal should be computed. See “Specifying Sample Time” for information on how to specify the sample time.

**Sample mode**

Sample mode of this signal. Simulink ignores the setting of this field.

**Minimum**

Minimum value that the signal can have. This field is intended for use in documenting this signal.

**Maximum**

Maximum value that the signal can have. This field is intended for use in documenting this signal.

**Storage class**

Storage class of this signal. See “Storage Classes of Tunable Parameters” in the Real-Time Workshop documentation for more information.

Alias

Alternate name for this parameter. Simulink ignores this setting.

Description

Description of this parameter. This field is intended for use in documenting this signal.

Properties

Name	Access	Description
DataType	RW	String specifying the name of the data type of this signal. ( <b>Data type</b> )
Dimensions	RW	Scalar or vector specifying the dimensions of this parameter. ( <b>Dimensions</b> )
Complexity	RW	String specifying the numeric type of this signal. Valid values are 'auto', 'real', or 'complex'. ( <b>Complexity</b> )
Min	RW	Minimum value that this signal can have. ( <b>Dimensions</b> )
Max	RW	Maximum value that this signal can have. ( <b>Maximum</b> )
DocUnits	RW	Measurement units in which this signal's value is expressed. ( <b>Units</b> )
RTWInfo	RW	Information used by Real-Time Workshop for generating code for this parameter. The value of this property is an object of Simulink.ParamRTWInfo class.
SampleTime	RW	Rate at which this signal should be updated. ( <b>Sample time</b> )
Sampling Mode	RW	Sampling mode of this signal. ( <b>Sample mode</b> )

# Simulink.StructElement

**Purpose** Describe an element of a data structure

**Description** Objects of this class describe elements of structures described by objects of `Simulink.StructType` class.

**Property  
Dialog Box**

Simulink.StructElement: e1

Name:

a

Dimensions:

1

Data type:

double

Complexity:

real

Revert

Help

Apply

**Name**  
Name of the element.

**Dimensions**  
A vector specifying the dimensions of the element.

**Data type**  
Name of the data type of this element.

**Complexity**  
Numeric type (i.e., real or complex) of this element.

**Properties**

Name	Access	Description
Name	RW	String specifying the name of this element. ( <b>Name</b> )
Dimensions	RW	A vector specifying the dimensions of this element. ( <b>Dimensions</b> )

Name	Access	Description
DataType	RW	String that specifies the name of the data type of this element. ( <b>Data type</b> )
Complexity	RW	String that specifies the numeric type ('real' or 'complex') of this element. ( <b>Complexity</b> )

## See Also

Simulink.StructType

# Simulink.StructElement

---

<b>Purpose</b>	Simulink.StructType Describe a data structure used as the value of a signal or parameter
<b>Description</b>	An object of this class describes a signal or parameter whose values are data structures (i.e., aggregates of data of different types as opposed to arrays of values of the same type). This class is intended to support development and use of custom blocks (e.g., S-Function blocks) that accept or output data structures. The class allows users of such blocks to determine the structure of the signals connected to them.

You can use either the Model Explorer or the MATLAB command line to create an instance of this class. To define the elements of a structure, create an array of instances of `Simulink.StructElement` at the MATLAB command line and assign the array as the value of the structure's `Elements` property. For example, the following commands define a structure that contains a floating point and an integer element.

```
v = Simulink.StructElement;  
v.Name = 'v';  
v.DataType = 'single';  
n = Simulink.StructElement;  
n.Name = 'n';  
n.DataType = 'uint8';  
  
s = Simulink.StructType;  
s.Elements = [v n];
```

You can use a structure type object to specify the data type of Inport and Signal Specification blocks. To do this, enter the name of the variable that references the structure type object as the data type in the block's parameter dialog box.

The Simulink S-function API lets you create S-functions capable of generating and manipulating signal structures (see the `simstruct.h` header file for more information). You can connect signal structures created by S-function blocks to any standard Simulink block that accepts any data type. This includes virtual blocks and the Switch block configured to require the same data type on all its data inputs.



## Property Dialog Box

**Simulink.StructType: state**

Struct elements

Name	Dimension	Data/Bus Type	Complexity
velocity	1	single	real
roll	1	double	real
pitch	1	double	real
yaw	1	double	real

Header file:

Description:

### Struct elements

Table that displays the properties of the structure's elements. You cannot edit this table. To add or delete this structure's elements or change the properties of elements, you must use MATLAB commands, e.g.,

```
state.Elements(1).DataType = 'double';
```

### Header file

Name of a C header file that declares this structure. This field is intended for use by Real-Time Workshop. Simulink ignores it.

### Description

Description of this structure. This field is intended for you to use to document this structure. Simulink itself does not use this field.

# Simulink.StructElement

## 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. ( <b>Struct elements</b> )
Description	RW	String that describes this structure. This property is intended for user use. Simulink itself does not use it. ( <b>Description</b> )
HeaderFile	RW	String that specifies the name of a C header file that declares this structure. ( <b>Header file</b> )

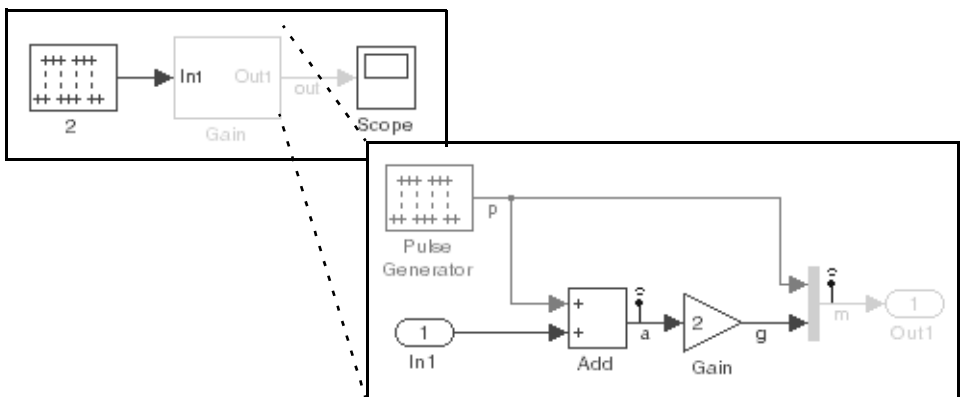
## See Also

Simulink.StructElement

**Purpose** Log signals in a subsystem

**Description** Simulink creates instances of this class to contain logs for signals belonging to a subsystem (see “Logging Signals”). 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 logouts, that contains a Simulink.SubsysDataLogs object, named Gain, that contains the log data for signals a and g in the subsystem named Gain.

```
>> logouts.Gain
```

```
ans =
```

```
Simulink.SubsysDataLogs (Gain):
```

Name	Elements	Simulink Class
a	1	Timeseries
m	2	TsArray

# Simulink.SubsysDataLogs

---

You can use either fully qualified log names or the `unpack` command to access the signal logs contained by a `SubsysDataLogs` object. For example, to access the amplitudes logged for signal `a` in the preceding example, you could enter the following at the MATLAB command line:

```
>> data = logsout.Gain.a.Data;
```

or

```
>> logsout.unpack('all');  
data = a.Data;
```

## See Also

`Simulink.ModelDataLogs`, `Simulink.Timeseries`, `Simulink.TsArray`, `unpack`

**Purpose** Provide information about the time data in a Simulink.Timeseries object

**Description** Simulink creates instances of these objects to describe the time data that it includes in Simulink.Timeseries objects.

Properties	Name	Access	Description
	Units	RW	The units, e.g., 'secs', in which the time series data are expressed in the associated Simulink.Timeseries object.
	Start	RW	If the associated signal is not in a conditionally executed subsystem, this field contains the simulation time of the first signal value recorded in the associated Simulink.Timeseries object. If the signal is in a conditionally executed subsystem, this field contains an array of times when the system became active.
	End	RW	If the associated signal is not in a conditionally executed subsystem, this field contains the simulation time of the last signal value recorded in the associated Simulink.Timeseries object. If the signal is in a conditionally executed subsystem, this field contains an array of times when the system became inactive.
	Increment	RW	The interval between simulation times at which signal data is logged in the associated Simulink.Timeseries object.
	Length	W	The number of signal samples recorded in the associated Simulink.Timeseries object, i.e., the length of the arrays referenced by the object's Time and Data properties.

**See Also** Simulink.Timeseries

# Simulink.Timeseries

**Purpose** Log signal data

**Description** Simulink creates instances of this class to store signal data that it logs while simulating a model (see “Logging Signals”).

Properties	Name	Access	Description
	Name	RW	Name of this signal log.
	BlockPath	RW	Path of the block that output the signal logged in this signal log.
	PortIndex	RW	Index of the output port that emitted the signal logged in this signal log.
	SignalName	RW	Name of the signal logged in this signal log.
	ParentName	RW	Name of the parent of the signal recorded in this log, if the signal is an element of a composite signal; otherwise, the same as SignalName.
	TimeInfo	RW	An object of Simulink.TimeInfo class that describes the time data in this log.
	Time	RW	An array containing the simulation times at which signal data was logged.
	Data	RW	An array containing the signal data.

**See Also** Simulink.ModelDataLogs, Simulink.TimeInfo, unpack

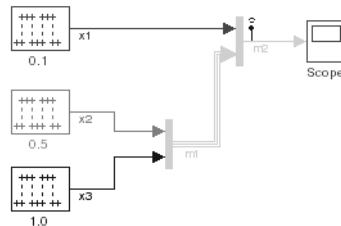
## Purpose

Log composite virtual signals

## Description

Simulink creates instances of this class to contain the data that it logs for a composite virtual signal, e.g., the output of a Mux or of a virtual Bus Creator block (see “Logging Signals”). Objects of the `Simulink.TsArray` class have a variable number of properties. The first property, called `Name`, specifies the log name of the composite signal. The remaining properties reference logs (i.e., `Simulink.Timeseries` objects) for the elements of the composite signal. The name of each property is the log name of the corresponding signal.

Consider, for example, the following model.



This model specifies that Simulink should log the values of the composite signal `m2` during simulation. After simulation of this model, the MATLAB workspace contains a `Simulink.ModelDataLogs` object, named `logsout`, that contains a `Simulink.TsArray` object, named `m2`, that contains the logs for the elements of `m2`, i.e., for the elementary signals `x1`, `x2`, and `x3`. Entering the fully qualified name of the `Simulink.TsArray` object, i.e., `logsout.m2`, at the MATLAB command line reveals the structure of the signal log for this model.

```
>> logsout.m2
Simulink.TsArray:
      Name          Simulink Class
      ----          -
      x1            Timeseries
      x2            Timeseries
      x3            Timeseries
```

You can use either fully qualified log names or the `unpack` command to access the signal logs contained by a `Simulink.TsArray` object. For example, to access

# Simulink.TsArray

---

the amplitudes logged for signal `x1` in the preceding example, you could enter the following at the MATLAB command line:

```
>> data = logouts.m2.x1.Data;
```

or

```
>> logouts.unpack('all');  
data = x1.Data;
```

## See Also

`Simulink.ModelDataLogs`, `Simulink.Timeseries`, `unpack`







# Model and Block Parameters

---

The following sections list parameters that you can set for Simulink models and blocks, using the `set_param` command.

Model Parameters (p. 9-2)

Parameters specific to models.

Common Block Parameters (p. 9-30)

Parameters that all blocks have.

Block-Specific Parameters (p. 9-37)

Parameters that a specific block has.

Mask Parameters (p. 9-126)

Parameters of a masked subsystem.

## Model Parameters

This table lists and describes parameters that describe a model. The parameters appear in the order they are defined in the model file, described in Chapter 10, “Model File Format.” The table also includes model callback parameters (see “Using Callback Routines”). The **Description** column indicates where you can set the value on the **Simulation Parameters** dialog box. Model parameters that are simulation parameters are described in more detail in “The Simulation Parameters Dialog Box”. Examples showing how to change parameters follow the table (see “Examples of Setting Model Parameters” on page 9–28).

Parameter values must be specified as quoted strings. The string contents depend on the parameter and can be numeric (scalar, vector, or matrix), a variable name, a filename, or a particular value. The **Values** column shows the type of value required, the possible values (separated with a vertical line), and the default value, enclosed in braces.

### Model Parameters

Parameter	Description	Values
AbsTol	Absolute error tolerance. Setting for the <b>Absolute tolerance</b> on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	string { 'auto' }
AccelMakeCommand	Program that builds the Simulink Accelerator target for this model.	string { 'make_rtw' }
AccelSystemTargetFile	TLC file used to build the Simulink Accelerator target for this model.	string { 'accel.tlc' }
AccelTemplateMakefile	Template for the makefile used to build the Simulink Accelerator target for this model.	string { 'accel_default_tmf' }
AlgebraicLoopMsg	Specifies diagnostic action to take when there is an algebraic loop. Set by the <b>Algebraic loop</b> option on the <b>Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   { 'warning' }   'error'
AnalyticLinearization	For internal use.	

**Model Parameters (Continued)**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
ArrayBoundsChecking	Setting for the <b>Array bounds exceeded</b> diagnostic on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'   'error'
ArtificialAlgebraic LoopMsg	Setting for the <b>Minimize algebraic loop</b> diagnostic on the <b>Solver Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
AssertionControl	Setting for the <b>Model Verification block enabling</b> control on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'UseLocalSettings' }   'EnableAll'   'DisableAll'
BlockDescription StringDataTip	Specifies whether to display the user description string for a block as a data tip. Set by the <b>User Description String</b> command on the model editor's <b>View-&gt;Block Data Tips Options</b> menu.	'on'   {'off'}
BlockDiagramType	Type of block diagram.	'model'   'library'
BlockNameDataTip	Specifies whether to display the user description string for a block as a data tip. Set by the <b>Block Name</b> command on the model editor's <b>View-&gt;Block Data Tips Options</b> menu.	'on'   {'off'}
BlockParametersData Tip	Specifies whether to display the user description string for a block as a data tip. Set by the <b>Parameter Names and Values</b> command on the model editor's <b>View-&gt;Block Data Tips Options</b> menu.	'on'   {'off'}
BlockPriority ViolationMsg	Setting for the <b>Block priority violation</b> diagnostic on the <b>Solver Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'

### Model Parameters (Continued)

Parameter	Description	Values
BlockReductionOpt	Enables block reduction optimization. Set by the <b>Block reduction optimization</b> option on the <b>Optimization</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'
Blocks	Names of the blocks that this model contains.	cell array {{{}}
BooleanDataType	Enable Boolean mode. Set by the <b>Implement logic signals as boolean data (vs. double)</b> option on the <b>Optimization</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'
Browser	Deprecated.	
BrowserHandle	Deprecated.	
BrowserLookUnderMasks	Show masked subsystems in the Model Browser. Set by the <b>Show Masked Subsystems</b> command on the model editor's <b>View-&gt;Model Browser Options</b> menu.	'on'   { 'off' }
BrowserShowLibraryLinks	Show library links in the Model Browser. Set by the <b>Show Library Links</b> command on the model editor's <b>View-&gt;Model Browser Options</b> menu.	'on'   { 'off' }
BufferReuse	Enable reuse of block I/O buffers. Set by the <b>Reuse block outputs</b> option on the <b>Optimization</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'
CheckExecutionContextRuntimeOutputMsg	Set by the <b>Check runtime output of execution context</b> option on the <b>Compatibility Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'
CheckExecutionOrderContextPreStartOutputMSg	Set by the <b>Check preactivation output of execution context</b> option on the <b>Compatibility Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'

**Model Parameters (Continued)**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
CheckForMatrixSingularity	Set by the <b>Attempted division by singular matrix</b> option on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'   'error'
CheckModelReferenceTargetMessage	Message behavior when the <b>Rebuild options</b> is set to never in the <b>Model Referencing Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   'warning'   {'error'}
CheckSSInitialOutputMsg	Enable checking for undefined initial subsystem output. Set by the <b>Check undefined subsystem initial output</b> option on the <b>Compatibility Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'
CloseFcn	Close callback.	command or variable
ConditionallyExecuteInputs	Enables conditional input branch execution optimization. Set by the <b>Conditional input branch execution</b> control on the <b>Optimization</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'
ConfigurationManager	Configuration manager for this model.	string { 'None' }
ConsistencyChecking	Consistency checking. Set by the <b>Solver data inconsistency</b> option on the <b>Solver Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'   'error'
CovCompData	<p>If CovHTMLOptions is set to on, and CovCumulativeReport is set to off, this parameter specifies cvdata objects containing additional model coverage data to include in the model coverage report.</p> <p>Specified by the <b>Additional data to include in report (cvdata objects)</b> field in the <b>Report</b> pane of the <b>Coverage Settings</b> dialog box.</p>	string

Model Parameters (Continued)

Parameter	Description	Values
CovCumulativeReport	<p>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.</p> <p>If set to on, display the coverage results for the last simulation in the report. If set to off, display the coverage results from successive simulations in the report. Set by the radial buttons <b>Cumulative runs</b> (on)/<b>Last runs</b> (off) in the <b>Report</b> pane of the <b>Coverage Settings</b> dialog box.</p>	'on'   {'off'}
CovCumulativeVarName	<p>If covSaveCumulativeToWorkSpace Var is set to on, model coverage saves the results of successive simulations in the workspace variable specified by this property. Entered in the field below the selected <b>Save cumulative results in workspace variable</b> check box on the <b>Results</b> pane of the <b>Coverage Settings</b> dialog box.</p>	string {'covCumulativeData'}



**Model Parameters (Continued)**

Parameter	Description	Values
CovHTMLOptions	If CovHTMLReporting is set to on, use this parameter to select from a set of display options for the resulting model coverage report. In the <b>Reporting</b> pane of the <b>Coverage Settings</b> dialog box, select <b>Settings</b> to receive a dialog box for selecting these options.	<p>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):</p> <ul style="list-style-type: none"> <li>• ' -aTS=1 ' <p>Include each test in the model summary</p> </li> <li>• ' -bRG=1 ' <p>Produce bar graphs in the model summary</p> </li> <li>• ' -bTC=0 ' <p>Use two color bar graphs (red, blue)</p> </li> <li>• ' -hTR=0 ' <p>Display hit/count ratio in the model summary</p> </li> <li>• ' -nFC=0 ' <p>Do not report fully covered model objects</p> </li> <li>• ' -scm=1 ' <p>Include cyclomatic complexity numbers in summary</p> </li> <li>• ' -bcm=1 ' <p>Include cyclomatic complexity numbers in block details</p> </li> </ul>
CovHTMLReporting	Set to on to tell Simulink to create an HTML report containing the coverage data in the MATLAB Help browser at the end of the simulation. Set by the <b>Generate HTML report</b> check box on the <b>Report</b> pane of the <b>Coverage Settings</b> dialog box.	{ 'on' }   'off'

Model Parameters (Continued)

Parameter	Description	Values
CovMetricSettings	Selects coverage metrics for coverage report. Coverage metrics are enabled by selecting the check boxes for individual coverages in the <b>Coverage Metrics</b> section of the <b>Coverage</b> pane of the <b>Coverage Settings</b> dialog box. Options 's' and 'w' are enabled by selecting the check boxes <b>Treat Simulink logic blocks as short-circuited</b> and <b>Warn when unsupported blocks exist in model</b> , respectively, in the <b>Options</b> pane of the <b>Coverage Settings</b> dialog box. Option 'e' is disabled by selecting the check box <b>Display coverage results using model coloring</b> in the <b>Results</b> pane of the <b>Coverage Settings</b> dialog box.	string { 'dw' } Each order-independent character in the string enables a coverage metric or option as follows: <ul style="list-style-type: none"><li>• 'd' Enable decision coverage</li><li>• 'c' Enable condition coverage</li><li>• 'm' Enable MCDC coverage</li><li>• 't' Enable lookup table coverage</li><li>• 'r' Enable signal range coverage</li><li>• 's' Treat Simulink logic blocks as short-circuited</li><li>• 'w' Warn when unsupported blocks exist in model</li><li>• 'e' Eliminate model coloring for coverage results</li></ul>
CovNameIncrementing	If CovSaveSingleToWorkspaceVar is set to on, setting this parameter to on tells Model Coverage to increment the workspace variable specified in CovSaveName to store the results succeeding simulations. Entered in the <b>Increment variable name with each simulation</b> check box below the selected <b>Save last run in workspace variable</b> check box on the <b>Results</b> pane of the <b>Coverage Settings</b> dialog box.	'on'   {'off'}

**Model Parameters (Continued)**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
CovPath	Model path of the subsystem for which Simulink gathers and reports coverage data. Set by browsing for the path in <b>Coverage Instrumentation Path</b> on the <b>Coverage</b> pane of the <b>Coverage Settings</b> dialog box.	string { '/' }
CovReportOnPause	Specifies that when you pause during simulation the model coverage report appears in updated form with coverage results up to the current pause or stop time. Set by selecting the <b>Update results on pause</b> check box on the <b>Results</b> pane of the <b>Coverage Settings</b> dialog box.	{ 'on' }   'off'
covSaveCumulativeToWorkspaceVar	If set to on, causes Model Coverage to accumulate and save the results of successive simulations in the workspace variable in CovCumulativeVarName. Set by selecting the <b>Save cumulative results in workspace variable</b> check box on the <b>Results</b> pane of the <b>Coverage Settings</b> dialog box.	{ 'on' }   'off'
CovSaveName	If CovSaveSingleToWorkspaceVar is set to on, Model Coverage saves the results of the last simulation run in the workspace variable specified by this property. Entered in the field below the selected <b>Save last run in workspace variable</b> check box on the <b>Results</b> pane of the <b>Coverage Settings</b> dialog box.	string { 'covdata' }
CovSaveSingleToWorkspaceVar	If enabled, tells Model Coverage to save the results of the last simulation run in the workspace variable specified by the CovSaveName property. Set by selecting the <b>Save last run in workspace variable</b> check box on the <b>Results</b> pane of the <b>Coverage Settings</b> dialog box.	{ 'on' }   'off'
Created	Date and time model was created.	string

### Model Parameters (Continued)

Parameter	Description	Values
Creator	Name of model creator.	string {''}
CurrentBlock	For internal use.	
CurrentOutputPort	For internal use.	
DataTypeOverride	Specifies data type used to override fixed-point datatypes. Set by the <b>Data type override</b> control on the <b>Fixed-Point Settings</b> dialog box.	{'UseLocalSettings'}   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'
Decimation	Decimation factor. Set by the <b>Decimation</b> field on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	string {'1'}
DefaultBlockFontSize	Default font size for blocks contained by this model.	{'10'}
DeleteChildFcn	Delete child callback.	string {''}
Description	Description of this model.	string
Dirty	If the paramter is on, the model has unsaved changes.	'on'   {'off'}
DiscreteInheritContinuousMsg	Specifies diagnostic action to take when a Unit Delay block inherits a continuous sample time. Set by the <b>Discrete used as continuous</b> control on the <b>Sample Time Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
DisplayBdSearchResults	For internal use.	
DisplayBlockIO	For internal use.	
DisplayCallGraphDominators	For internal use.	
DisplayCompileStatus	For internal use.	
DisplayCondInputTree	For internal use.	

**Model Parameters (Continued)**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
DisplayCondStIDTree	For internal use.	
DisplayError Directions	For internal use.	
DisplayInvisible Sources	For internal use.	
DisplaySortedLists	For internal use.	
DisplayVectorAnd FunctionCounts	For internal use.	
DisplayVect PropagationResults	For internal use.	
ExecutionContextIcon	Toggles display of execution context icons on this model's block diagram.	'on'   {'off'}
ExternalInput	Names of MATLAB workspace variables used to designate data and times to be loaded from the workspace. Set by the <b>Input</b> option on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	scalar or vector {'[t, u]'}
ExtMode...	Parameters whose names start with ExtMode apply to Simulink External Mode. See “External Mode” in the Real-Time Workshop documentation for more information.	
ExtrapolationOrder	Extrapolation order of the ode14x implicit fixed-step solver. Set by the <b>Extrapolation order</b> control on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	1   2   3   {4}
FileName	For internal use.	
FinalStateName	Names of final states to be saved to the workspace. Set by the <b>Final states</b> option on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	string {'xFinal'}

Model Parameters (Continued)

Parameter	Description	Values
FixedStep	Fixed step size. Set by the <b>Fixed step size (fundamental sample time)</b> field on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	string {'auto'}
FixPtInfo	For internal use.	
ForceArrayBounds Checking	For internal use.	
ForceConsistency Checking	For internal use.	
ForceModelCoverage	For internal use.	
ForwardingTable	Specifies the forwarding table for this library. See “Updating Links to Reflect Block Path Changes” in the Using Simulink documentation for more information.	{{'old_path_1', 'new_path_1'} ... {'old_path_n', 'new_path_n'}}
ForwardingTable String	For internal use.	
GridSpacing	Spacing of model editor grid in pixels.	double
Handle	Handle of this model’s block diagram.	double
HiliteAncestors	For internal use.	
IgnoreBidirectional Lines	For internal use.	
InheritedTsInSrcMsg	Message behavior when the sample time is inherited. Set by the <b>Source block specifies -1 sample time</b> control on the <b>Sample Time Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
InitFcn	Function that is called when this model is first compiled for simulation.	string {''}

**Model Parameters (Continued)**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
InitialState	Initial state name or values. Set by the <b>Initial state</b> field on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	variable or vector {'xInitial'}
InitialStep	Initial step size. Set by the <b>Initial step size</b> field on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	string {'auto'}
Int32ToFloatConvMsg	Message behavior when a 32-bit integer is converted to a single-precision float. Set by the <b>32-bit integer to single precision float conversion</b> control on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}
IntegerOverflowMsg	Message behavior when there is an integer overflow. Set by the <b>Data overflow</b> control on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
InvalidFcnCallConnMsg	Message behavior when there is an invalid function call connection. Set by the <b>Invalid function call connection</b> control on the <b>Connectivity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   'warning'   {'error'}
InvariantConstants	Invariant constant setting.	'on'   {'off'}
Jacobian	For internal use.	
LastModifiedBy	Name used for version control.	string
LastModifiedDate	Date used for version control.	string
LibraryLinkDisplay	Shows which blocks in the model are linked or have disabled or modified links. Set by the <b>Library Link Display</b> option under the <b>Format</b> menu.	{'none'}   'user'   'all'

### Model Parameters (Continued)

Parameter	Description	Values
LibraryType	For internal use.	{ 'none' }   'BlockLibrary'   'IOLibrary'
LimitDataPoints	Limit output. Set by the <b>Limit data points to last</b> check box on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'
LinearizationMsg	For internal use.	
Lines	For internal use.	
LoadExternalInput	Load input from workspace. Set by the <b>Input</b> check box on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	'on'   { 'off' }
LoadInitialState	Load initial state from workspace. Set by the <b>Initial state</b> check box on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	'on'   { 'off' }
Location	For internal use.	
Lock	For internal use.	
MaxDataPoints	Maximum number of output data points to save. Set by the <b>Limit data points to last</b> field on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	string { '1000' }
MaxNumMinSteps	Maximum number of times the solver uses the minimum step size.	string { '-1' }
MaxOrder	Maximum order for ode15s. Set by the <b>Maximum order</b> option on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	1   2   3   4   {5}
MaxStep	Maximum step size. Set by the <b>Max step size</b> field on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	string { 'auto' }



**Model Parameters (Continued)**

Parameter	Description	Values
MinMaxOverflowArchiveData	For internal use.	
MinMaxOverflowArchiveMode	Logging type for fixed-point logging. Set by the <b>Logging type</b> option in the <b>Fixed-Point Settings</b> dialog box.	{ 'Overwrite' }   'Merge'
MinMaxOverflowLogging	Setting for fixed-point logging. Set by the <b>Logging mode</b> option in the <b>Fixed-Point Settings</b> dialog box.	{ 'UseLocalSettings' }   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
MinStep	Minimum step size for the solver. Set by the <b>Min step size</b> field on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	string { 'auto' }
MinStepSizeMsg	Message shown when minimum step size is violated. Set by the <b>Min step size violation</b> option on the <b>Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'warning' }   'error'
ModelBrowserVisibility	Show the Model Browser. Set by the <b>Model Browser</b> command of the model's <b>View-&gt;Model Browser Options</b> menu.	'on'   { 'off' }
ModelDataFile	For internal use.	string { '' }
ModelDependencies	List of model dependencies. Set by the <b>Model dependencies</b> field on the <b>Model Referencing</b> pane of the <b>Configuration Parameters</b> dialog box.	string { '' }
ModelReferenceCSMismatchMessage	Message shown when there is a model configuration mismatch. Set by the <b>Model configuration mismatch</b> option on the <b>Model Referencing Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'   'error'

## Model Parameters (Continued)

Parameter	Description	Values
ModelReferenceData LoggingMessage	Message shown when there is unsupported data logging. Set by the <b>Unsupported data logging</b> option on the <b>Model Referencing Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
ModelReferenceIO MismatchMessage	Message shown when there is a port and parameter mismatch. Set by the <b>Port and parameter mismatch</b> option on the <b>Model Referencing Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{'none'}   'warning'   'error'
ModelReferenceIOMsg	Message shown when there is an invalid root Inport/Outport block connection. Set by the <b>Invalid root Inport/Outport block connection</b> option on the <b>Model Referencing Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{'none'}   'warning'   'error'
ModelReferenceNum InstancesAllowed	Total number of instances allowed per top model. Set by the <b>Total number of instances allowed per top model</b> option on the <b>Model Referencing</b> pane of the <b>Configuration Parameters</b> dialog box.	'Zero'   'Single'   {'Multi'}
ModelReferenceSim TargetVerbose	Print detailed information when generating simulation targets for models referenced by a top-level model.	'on'   {'off'}
ModelReferenceTarget Type	For internal use.	
ModelReferenceVersion MismatchMessage	Message shown when there is a model block version mismatch. Set by the <b>Model block version mismatch</b> option on the <b>Model Referencing Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{'none'}   'warning'   'error'

**Model Parameters (Continued)**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
ModelrefMinAlgLoopOccurrences	Toggles the minimization of algebraic loop occurrences. Set by the <b>Pass scalar root inputs by value</b> check box on the <b>Model Referencing</b> pane of the <b>Configuration Parameters</b> dialog box.	'on'   {'off'}
ModelrefPassRootInputsByReference	Toggles the passing of scalar root inputs by value. Set by the <b>Minimize algebraic loop occurrences</b> check box on the <b>Model Referencing</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'
ModelVersion	Version number of model.	string {'1.1'}
ModelVersionFormat	Format of model's version number.	string {'1.%.<AutoIncrement>0>'}
ModelWorkspace	References this model's model workspace object.	an instance of the Simulink.ModelWorkspace class
ModifiedBy	Last modifier of this model.	string
ModifiedByFormat	Format for the display of last modifier.	string {'%.<Auto>'}
ModifiedComment	Field for user comments.	string {' '}
ModifiedDate	Date of last model modification.	string
ModifiedDateFormat	Format of modified date.	string {'%.<Auto>'}
ModifiedHistory	Area for keeping notes about the history of the model. This is set by the <b>Model history</b> field on the <b>History</b> pane of the <b>Model Explorer</b> dialog box.	string {' '}
MultiTaskRateTransMsg	Specifies diagnostic action to take when an invalid rate transition takes place between two blocks operating in single-tasking mode. Set by the <b>Multitask rate transition</b> control on the <b>Sample Time Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'warning'   {'error'}

### Model Parameters (Continued)

Parameter	Description	Values
Name	Model name.	string
NumberNewton Iterations	Number of Newton's Method iterations performed by the ode14x implicit fixed-step solver. Set by the <b>Number Newton's iterations</b> control on the <b>Solver</b> pane of the pane of the <b>Configuration Parameters</b> dialog box.	integer {'1'}
ObjectParameters	Names/attributes of model parameters.	structure
Open	For internal use.	
OptimizeBlockIO Storage	Enables signal storage reuse optimization. Set by the <b>Signal storage reuse</b> control on the <b>Optimization</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'on' }   'off'
OutputOption	Time step output options for variable-step solvers. Set by the <b>Output options</b> option on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	'AdditionalOutputTimes'   {'RefineOutputTimes'}   'SpecifiedOutputTimes'
OutputSaveName	Workspace variable to store the model outputs. Set by the <b>Output</b> field on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	variable {'yout'}
OutputTimes	Output times set when <b>Output options</b> on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box is set to <b>Produce additional output</b> . Set by the <b>Output times</b> option on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	string {'[]'}
PaperOrientation	Printing paper orientation.	'portrait'   {'landscape'}   'rotated'
PaperPosition	Position of diagram on paper.	[left, bottom, width, height]

**Model Parameters (Continued)**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
PaperPositionMode	Paper position mode.	{ 'auto' }   'manual'
PaperSize	Size of PaperType in PaperUnits.	[width height] (read only)
PaperType	Printing paper type.	'usletter'   'uslegal'   'a0'   'a1'   'a2'   'a3'   'a4'   'a5'   'b0'   'b1'   'b2'   'b3'   'b4'   'b5'   'arch-A'   'arch-B'   'arch-C'   'arch-D'   'arch-E'   'A'   'B'   'C'   'D'   'E'   'tabloid'
PaperUnits	Printing paper size units.	'normalized'   {'inches'}   'centimeters'   'points'
ParameterArgument Names	List of parameters used as arguments when this model is called as a reference. Set in the <b>Model arguments (for referencing this model)</b> field in the <b>Model Workspace</b> pane of the <b>Model Explorer</b>	string { '' }
ParameterDowncastMsg	Specifies diagnostic action to take when a parameter downcast occurs during simulation. Set by the <b>Parameter downcast</b> control on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   'warning'   { 'error' }
ParameterOverflowMsg	Specifies diagnostic action to take when a parameter overflow occurs during simulation. Set by the <b>Parameter overflow</b> control on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   'warning'   { 'error' }
ParameterPooling	Toggles pooling of paramters in Real-Time Workshop generated code.	{ 'on' }   'off'

### Model Parameters (Continued)

Parameter	Description	Values
ParameterPrecisionLossMsg	Specifies diagnostic action to take when parameter precision loss occurs during simulation. Set by the <b>Parameter precision loss</b> control on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
ParamWorkspaceSource	For internal use.	
Parent	Name of the model or subsystem that owns this object. The value of this parameter for a model is an empty string.	string { '' }
PostLoadFcn	Function invoked just after this model is loaded.	string { '' }
PostSaveFcn	Function invoked just after this model is saved to disk.	string { '' }
PreLoadFcn	Preload callback.	command or variable { '' }
PreSaveFcn	Function invoked just before this model is saved to disk.	string { '' }
ProdHWDeviceType	Predefined hardware device to specify the C language constraints for your microprocessor. Set by the <b>Device type</b> option on the <b>Hardware Implementation</b> pane of the <b>Configuration Parameters</b> dialog box.	string {'32-bit Generic'}
ProdHWWordLengths	Number of bits used for char, short, int, and long, respectively (set by the hardware device type).	string {'8,16,32,32'}
Profile	Enables the simulation profiler for this model.	'on'   {'off'}

**Model Parameters (Continued)**

Parameter	Description	Values
RecordCoverage	<p>A value of on causes Simulink to gather and report model coverage data during simulation. The format of this report is controlled by the values of the following parameters:</p> <p>CovCompData CovCumulativeReport CovCumulativeVarName CovHTMLOptions CovHTMLReporting CovMetricSettings CovNameIncrementing CovPath CovReportOnPause covSaveCumulativeToWorkspaceVar CovSaveName CovSaveSingleToWorkspaceVar</p> <p>If the value is off, no model coverage data is collected or reported and the preceding coverage report parameters have no effect.</p>	'on'   {'off'}
Refine	Refine factor. Set by the <b>Refine factor</b> field on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	string {'1'}
RelTol	Relative error tolerance. Set by the <b>Relative tolerance</b> field on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	string {'1e-3'}
ReportName	Name of the associated file for the Report Generator	string { 'simulink-default.rpt' }
RequirementInfo	For internal use.	
RTW...	See the Real-Time Workshop documentation for more information on parameters whose names begin with RTW.	

Model Parameters (Continued)

Parameter	Description	Values
SampleTimeColors	Set by the <b>Sample Time Colors</b> option under the <b>Format</b> menu.	'on'   {'off'}
SaveDefaultBlockParams	For internal use.	
SaveFcn	Save callback.	command or variable
SaveFinalState	Save final states to workspace. Set by the <b>Final states</b> check box on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	'on'   {'off'}
SaveFormat	Format used to save data to the MATLAB workspace. Set by the <b>Format</b> option on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	{'Array'}   'Structure'   'StructureWithTime'
SaveOutput	Save simulation output to workspace. Set by the <b>Output</b> check box on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	{'on'}   'off'
SaveState	Save states to workspace. Set by the <b>States</b> check box on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	'on'   {'off'}
SaveTime	Save simulation time to workspace. Set by the <b>Time</b> check box on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	{'on'}   'off'
ScreenColor	Background color of the model window. Set by the <b>Screen color</b> option under the <b>Format</b> menu.	'black'   {'white'}   'red'   'green'   'blue'   'cyan'   'magenta'   'yellow'   'gray'   'lightBlue'   'orange'   'darkGreen'
ScrollbarOffset	For internal use.	



**Model Parameters (Continued)**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
SfunCompatibility CheckMsg	Specifies diagnostic action to take when S-function upgrades are needed. Set by the <b>S-function upgrades needed</b> option on the <b>Compatibility Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'   'error'
ShowGrid	Show the model editor grid.	'on'   { 'off' }
ShowLinearization Annotations	Toggles linearization icons in the model.	{ 'on' }   'off'
ShowLineDimensions	Show signal dimensions on this model's block diagram. Set by the <b>Show Signal Dimensions</b> command on the <b>Format-&gt;Port/Signal Displays</b> menu.	'on'   { 'off' }
ShowLineDimensions OnError	For internal use.	
ShowLineWidths	<b>Show Line Widths</b> menu option.	'on'   { 'off' }
ShowLoopsOnError	Highlight invalid loops graphically.	{ 'on' }   'off'
ShowModelReference BlockIO	Toggles display of version on block.	'on'   { 'off' }
Shown	For internal use.	
ShowPortDataTypes	Show data types of ports on this model's block diagram. Set by the <b>Port Datatypes</b> command on the <b>Format-&gt;Port/Signal Displays</b> menu.	'on'   { 'off' }
ShowPortDataTypesOn Error	For internal use.	
ShowStorageClass	Show storage classes of signals on this model's block diagram. Set by the <b>Storage Class</b> command on the <b>Format-&gt;Port/Signal Displays</b> menu.	'on'   { 'off' }

### Model Parameters (Continued)

Parameter	Description	Values
ShowTestPointIcons	Show test point icons on this model's block diagram. Set by the <b>Testpoint Icons</b> command on the <b>Format-&gt;Port/Signal Displays</b> menu.	'on'   {'off'}
ShowViewerIcons	Show viewer icons on this model's block diagram. Set by the <b>Viewer Icons</b> command on the <b>Format-&gt;Port/Signal Displays</b> menu.	'on'   {'off'}
SignalLabelMismatch Msg	Specifies diagnostic action to take when there is a signal label mismatch. Set by the <b>Signal label mismatch</b> option on the <b>Connectivity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'   'error'
SignalLoggingName	Signal logging name for saving data. Set by the <b>Signal logging name</b> field on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	string { 'logsOut' }
SignalResolution Control	Control which named states and signals get resolved to Simulink signal objects. Set by the <b>Signal resolution control</b> check box on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'UseLocalSettings'   'TryResolveAll'   { 'TryResolveAllWithWarning' }
SimParamPage	For internal use.	
SimulationCommand	Executes a simulation command.	'start'   'stop'   'pause'   'continue'   'step'   'update'   'SimParamDialog'   'connect'   'disconnect'   'WriteExtModeParamVect'   'AccelBuild'
SimulationMode	Indicates whether Simulink should run in normal, accelerated, or external mode.	{ 'normal' }   'accelerator'   'external'

**Model Parameters (Continued)**

Parameter	Description	Values
SimulationStatus	Indicates simulation status.	{ 'stopped' }   'updating'   'initializing'   'running'   'paused'   'terminating'   'external'
SimulationTime	Current time value for the simulation.	double {0}
SingleTaskRateTransMsg	Specifies diagnostic action to take when a rate transition takes place between two blocks operating in single-tasking mode. Set by the <b>Single task rate transition</b> control on the <b>Sample Time Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'   'error'
Solver	Solver used for the simulation. Set by the <b>Solver</b> check box on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	'VariableStepDiscrete'   { 'ode45' }   'ode23'   'ode113'   'ode15s'   'ode23s'   'ode23t'   'ode23tb'   'FixedStepDiscrete'   'ode5'   'ode4'   'ode3'   'ode2'   'ode1'   'ode14x'
SolverMode	Solver mode for this model. Set by the <b>Tasking mode for periodic sample times</b> option on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'Auto' }   'SingleTasking'   'MultiTasking'
SortedOrder	Show the sorted order of this model's blocks on the block diagram. Set by the <b>SortedOrder</b> command on the model editor's <b>Format-&gt;Block Displays</b> menu.	'on'   { 'off' }
StartFcn	Start simulation callback.	command or variable { '' }
StartTime	Simulation start time. Set by the <b>Start time</b> field on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	string { '0.0' }

### Model Parameters (Continued)

Parameter	Description	Values
StateSaveName	State output name to be saved to workspace. Set by the <b>States</b> field on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	variable {'xout'}
StatusBar	Show/hide the status bar on the model editor window. Set by the <b>Status Bar</b> command on the model editor's <b>View</b> menu.	{ 'on' }   'off'
StopFcn	Stop simulation callback.	command or variable { '' }
StopTime	Simulation stop time. Set by the <b>Stop time</b> field on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	string { '10.0' }
TargetFcnLib	For internal use.	
TimeSaveName	Simulation time name. Set by the <b>Time</b> field on the <b>Data Import/Export</b> pane of the <b>Configuration Parameters</b> dialog box.	variable {'tout'}
TLC...	Parameters whose names begin with TLC are used for code generation. See the Real-Time Workshop documentation for more information.	
Toolbar	Show/hide the toolbar on the model editor window. Set by the <b>Toolbar</b> command on the model editor's <b>View</b> menu.	{ 'on' }   'off'
TryForcingSFcnDF	This flag is used for backward compatibility with user S-functions that were written prior to R12.	'on'   { 'off' }
TunableVars	List of global (tunable) parameters. Set in the <b>Model Parameter Configuration</b> dialog box.	string { '' }
TunableVarsStorage Class	List of storage classes for their respective tunable parameters. Set in the <b>Model Parameter Configuration</b> dialog box.	string { '' }

**Model Parameters (Continued)**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
TunableVarsTypeQualifier	List of storage type qualifiers for their respective tunable parameters. Set in the <b>Model Parameter Configuration</b> dialog box.	string { '' }
UnconnectedInputMsg	Unconnected input ports diagnostic. Set by the <b>Unconnected block input ports</b> option on the <b>Connectivity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
UnconnectedLineMsg	Unconnected lines diagnostic. Set by the <b>Unconnected line</b> option on the <b>Connectivity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
UnconnectedOutputMsg	Unconnected block output ports diagnostic. Set by the <b>Unconnected block output ports</b> option on the <b>Connectivity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
UnderSpecifiedDataTypeMsg	Detect usage of heuristics to assign signal data types. Set by the <b>Underspecified data types</b> option on the <b>Data Integrity Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'   'error'
UnknownTsInhSupMsg	Detect blocks that have not set whether they allow the model containing them to inherit a sample time. Set by the <b>Unspecified inheritability of sample time</b> option on the <b>Solver Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	'none'   {'warning'}   'error'
UnnecessaryDataTypeConvMsg	Detect unnecessary data type conversion blocks. Set by the <b>Unnecessary type conversions</b> option on the <b>Conversion Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'

Model Parameters (Continued)

Parameter	Description	Values
UpdateHistory	Specifies when to prompt the user about updating the model history. This is set by the <b>Prompt to update model history</b> option on the <b>History</b> pane of the <b>Model Explorer</b> dialog box.	{ 'UpdateHistoryNever' }   'UpdateHistoryWhenSave'
UpdateModelReferenceTargets	Rebuilding options. Set on the <b>Model Referencing</b> pane of the <b>Configuration Parameters</b> dialog box.	'IfOutOfDate'   'Force'   'AssumeUpToDate'   { 'IfOutOfDateOrStructuralChange' }
UseAnalysisPorts	For internal use.	
VectorMatrixConversionMsg	Detect vector-to-matrix or matrix-to-vector conversions. Set by the <b>Vector/matrix block input conversion</b> option on the <b>Conversion Diagnostics</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'none' }   'warning'   'error'
Version	Simulink version used to modify the model (read only).	release version number
WideLines	For internal use.	
WideVectorLines	<b>Wide Vector Lines</b> menu option.	'on'   { 'off' }
ZeroCross	For internal use.	
ZeroCrossControl	Enable zero-crossing detection. Set by the <b>Zero crossing control</b> control on the <b>Solver</b> pane of the <b>Configuration Parameters</b> dialog box.	{ 'UseLocalSettings' }   'EnableAll'   'DisableAll'
ZoomFactor	Zoom factor of the model editor window expressed as a percentage of normal (100%). Set by the zoom commands on the model editor's <b>View</b> menu.	string { '100' }

Examples of Setting Model Parameters

These examples show how to set model parameters for the mymodel system.

This command sets the simulation start and stop times.

```
set_param('mymodel','StartTime','5','StopTime','100')
```

This command sets the solver to ode15s and changes the maximum order.

```
set_param('mymodel','Solver','ode15s','MaxOrder','3')
```

This command associates a SaveFcn callback.

```
set_param('mymodel','SaveFcn','my_save_cb')
```

# Common Block Parameters

This table lists the parameters common to all Simulink blocks, including block callback parameters (see “Using Callback Routines”). Examples of commands that change these parameters follow this table (see “Examples of Setting Block Parameters” on page 9-35).

Common Block Parameters

Parameter	Description	Values
AncestorBlock	Name of the library block that the block is linked to (for blocks with a disabled link).	string
AttributesFormat String	String format specified for block annotations in the <b>Block Properties</b> dialog box.	string
BackgroundColor	Block background color.	RGB value array string
BlockDescription	Block description shown in the <b>Block Properties</b> dialog box.	string
BlockType	Block type (read only).	string
ClipboardFcn	Function called when block is copied to the clipboard ( <b>Ctrl+C</b> )	string
CloseFcn	Function called when <code>close_system</code> is run on block.	string
CompiledPort ComplexSignals	Complexity of port signals after updating diagram.	
CompiledPortData Types	Data types of port signals after updating diagram.	
CompilePort Dimensions	Dimensions of port signals after updating diagram.	
CompiledPortFrame Data	Frame mode of port signals after updating diagram.	



**Common Block Parameters (Continued)**

Parameter	Description	Values
CompiledPortWidths	Structure of port widths after updating diagram.	
CompiledSampleTime	Block sample time after updating diagram.	
CopyFcn	Function called when block is copied.	string
DataTypeOverride_Compiled	For internal use.	
DeleteFcn	Function called when block is deleted.	MATLAB expression
DestroyFcn	Function called when block is destroyed.	MATLAB expression
Description	Description of block. Set by the <b>Description</b> field in the <b>General</b> pane of the <b>Block Properties</b> dialog box.	text and tokens
Diagnostics		text and tokens
DialogParameters	Names/attributes of parameters in block's parameter dialog box.	structure
DropShadow	Display drop shadow.	{'off'}   'on'
ExtModeUpload Option		{'none'}   'log'   'monitor'
ExtModeLogging Supported		{'off'}   'on'
ExtModeLogging Trig		{'off'}   'on'
FontAngle	Font angle.	'normal'   'italic'   'oblique'   {'auto'}
FontName	Font.	string

Common Block Parameters (Continued)

Parameter	Description	Values
FontSize	Font size. A value of -1 specifies that this block inherits the font size specified by the DefaultBlockFontSize model parameter.	real { '-1' }
FontWeight	Font weight.	'light'   'normal'   'demi'   'bold'   { 'auto' }
ForegroundColor	Foreground color of block's icon.	string { 'black' }
Handle	Block handle.	real
HiliteAncestors		{ 'none' }   'error'   'redWhite'   'blueWhite'   'greenWhite'   'orangeWhite'   'blackWhite'   'find'   'debug'   'unique'   'different'   'lineTrace'   'fade'   'default'   'user1'   'user2'   'user3'   'user4'   'user5'
InitFcn	Initialization function for a masked block.	MATLAB expression
InputPorts	Array of input port locations.	[ h1,v1; h2,v2; ... ]
InputSignalNames	Names of input signals.	matrix
IOSignalStrings		list
IOType		{ 'none' }   'viewer'   'sigen'
LineHandles	Handles of lines connected to block.	matrix
LinkStatus	Link status of block.	{ 'none' }   'resolved'   'unresolved'   'implicit'   'inactive'   'restore'   'propagate'
LoadFcn	Function called when block is loaded.	MATLAB expression

**Common Block Parameters (Continued)**

Parameter	Description	Values
MinMaxOverflow Logging_Compiled	For internal use.	
ModelCloseFcn	Function called when model is closed.	MATLAB expression
ModelParamTable Info	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's <b>Block Parameters</b> dialog box is opened.	MATLAB expression
Orientation	Where block faces.	{'right'}   'left'   'up'   'down'
OutputPorts	Array of output port locations.	[h1,v1; h2,v2; ...]
OutputSignalNames	Names of output signals.	matrix
Parent	Name of the system that owns the block.	string {'untitled'}
ParentCloseFcn	Function called when parent subsystem is closed.	MATLAB expression
PortConnectivity	Information about port types, position of ports, source blocks, source ports, destination blocks, and destination ports.	matrix
PortHandles	Handles of all ports.	matrix

### Common Block Parameters (Continued)

Parameter	Description	Values
Ports	Numbers of each kind of port.	
Position	Position of block in model window.	vector [left top right bottom] <i>not</i> enclosed in quotation marks
PostOutputsFcn	Function called after the block's output function is called.	MATLAB expression
PostSaveFcn	Function called after the block is saved.	MATLAB expression
PreOutputsFcn	Function called before block's preoutputs function is called.	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 <b>Priority</b> field on the <b>General</b> pane of the <b>Block Properties</b> dialog box.	string { ' ' }
ReferenceBlock	Name of the library block that this block is linked to.	string { ' ' }
RequirementsInfo	For internal use.	
RTWData	User specified data, used by Real-Time Workshop.	
SampleTime	Value of the sample time parameter.	
Selected	Status of whether or not block is selected.	{ 'on' }   'off'
ShowName	Display block name.	{ 'on' }   'off'
StartFcn	Function called at the start of a simulation.	MATLAB expression

Common Block Parameters (Continued)

Parameter	Description	Values
StatePerturbationJacobian	See Simulink Control documentation for details.	
StopFcn	Function called at the termination of a simulation.	MATLAB expression
Tag	Text that appears in the block label that Simulink generates. Set by the <b>Tag</b> field on the <b>General</b> pane of the <b>Block Properties</b> dialog box.	string {''}
Type	Simulink object type (read only).	'block'
UndoDeleteFcn	Function called when block deletion is undone.	MATLAB expression
UserData	User-specified data that can have any MATLAB data type.	{'[]'}
UserDataPersistent	Status of whether or not UserData will be saved in the model file.	'on'   {'off'}

Examples of Setting Block Parameters

These examples illustrate how to change common block parameters.

This command changes the orientation of the Gain block in the mymodel system so it faces the opposite direction (right to left).

```
set_param('mymodel/Gain','Orientation','left')
```

This command associates an OpenFcn callback with the Gain block in the mymodel system.

```
set_param('mymodel/Gain','OpenFcn','my_open_cb')
```

This command sets the Position parameter of the Gain block in the mymodel system. The block is 75 pixels wide by 25 pixels high. The position vector is *not* enclosed in quotation marks.

```
set_param('mymodel/Gain','Position',[50 250 125 275])
```

# Block-Specific Parameters

These tables list block-specific parameters for all Simulink blocks. The type of the block appears in parentheses after the block name. Some Simulink blocks are implemented as masked subsystems. The tables indicate masked blocks by adding the designation “masked” after the block type.

**Note** The type listed for nonmasked blocks is the value of the block’s BlockType parameter; the type listed for masked blocks is the value of the block’s MaskType parameter. For more information, see “Mask Parameters” on page 9-126.

The **Dialog Box Prompt** column indicates the text of the prompt for the parameter on the block’s dialog box. The **Values** column shows the type of value required (scalar, vector, variable), the possible values (separated with a vertical line), and the default value (enclosed in braces).

## Continuous Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Derivative (Derivative)		
LinearizePole	Linearization Time Constant $s/(Ns+1)$	string {'inf'}
Integrator (Integrator)		
ExternalReset	External reset	{'none'}   'rising'   'falling'   'either'   'level'
InitialConditionSource	Initial condition source	{'internal'}   'external'
InitialCondition	Initial condition	scalar or vector {'0'}
LimitOutput	Limit output	{'off'}   'on'
UpperSaturationLimit	Upper saturation limit	scalar or vector {'inf'}
LowerSaturationLimit	Lower saturation limit	scalar or vector {'-inf'}

**Continuous Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
ShowSaturationPort	Show saturation port	{ 'off' }   'on'
ShowStatePort	Show state port	{ 'off' }   'on'
AbsoluteTolerance	Absolute tolerance	string { 'auto' }
ZeroCross	Enable zero crossing detection	'off'   { 'on' }
State-Space (StateSpace)		
A	A	matrix { '1' }
B	B	matrix { '1' }
C	C	matrix { '1' }
D	D	matrix { '1' }
X0	Initial conditions	vector { '0' }
AbsoluteTolerance	Absolute tolerance	string { 'auto' }
Transfer Fcn (TransferFcn)		
Numerator	Numerator	vector or matrix { '[1]' }
Denominator	Denominator	vector { '[1 1]' }
AbsoluteTolerance	Absolute tolerance	string { 'auto' }
Transport Delay (TransportDelay)		
DelayTime	Time delay	scalar or vector { '1' }
InitialInput	Initial input	scalar or vector { '0' }
BufferSize	Initial buffer size	scalar { '1024' }
PadeOrder	Pade order (for linearization)	string { '0' }
TransDelayFeedthrough	Direct feedthrough of input during linearization	{ 'off' }   'on'
Variable Transport Delay (VariableTransportDelay)		



**Continuous Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
MaximumDelay	Maximum delay	scalar or vector { '10' }
InitialInput	Initial input	scalar or vector { '0' }
MaximumPoints	Buffer size	scalar { '1024' }
PadeOrder	Pade order (for linearization)	string { '0' }
TransDelayFeedthrough	Direct feedthrough of input during linearization	{ 'off' }   'on'
Zero-Pole (ZeroPole)		
Zeros	Zeros	vector { '[1]' }
Poles	Poles	vector { '[0 -1]' }
Gain	Gain	vector { '[1]' }
AbsoluteTolerance	Absolute tolerance	string { 'auto' }

**Discontinuities Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Backlash (Backlash)		
BacklashWidth	Deadband width	scalar or vector {1}
InitialOutput	Initial output	scalar or vector {0}
ZeroCross	Enable zero crossing detection	'off'   { 'on' }
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Coulomb & Viscous Friction (Coulombic and Viscous Friction) (masked subsystem)		
offset	Coulomb friction value (Offset)	string { '[1 3 2 0]' }

### Discontinuities Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
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	'off'   { 'on' }
LinearizeAsGain	Treat as gain when linearizing	'off'   { 'on' }
ZeroCross	Enable zero crossing detection	'off'   { 'on' }
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Dead Zone Dynamic (Dead Zone Dynamic) (masked subsystem)		
Hit Crossing (HitCross)		
HitCrossingOffset	Hit crossing offset	scalar or vector { '0' }
HitCrossingDirection	Hit crossing direction	'rising'   'falling'   { 'either' }
ShowOutputPort	Show output port	{ 'on' }   'off'
ZeroCross	Enable zero crossing detection	'off'   { 'on' }
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Quantizer (Quantizer)		
QuantizationInterval	Quantization interval	scalar or vector { '0.5' }
LinearizeAsGain	Treat as gain when linearizing	'off'   { 'on' }
SampleTime	Sample time (-1 for inherited)	string { '-1' }

**Discontinuities Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
Rate Limiter (RateLimiter)		
RisingSlewLimit	Rising slew rate	string {'1'}
FallingSlewLimit	Falling slew rate	string {'-1'}
SampleTimeMode	Sample time mode	'continuous'   {'inherited'}
InitialCondition	Initial condition	string {'0'}
LinearizeAsGain	Treat as gain when linearizing	'off'   {'on'}
Rate Limiter Dynamic (Rate Limiter Dynamic) (masked subsystem)		
Relay (Relay)		
OnSwitchValue	Switch on point	string {'eps'}
OffSwitchValue	Switch off point	string {'eps'}
OnOutputValue	Output when on	string {'1'}
OffOutputValue	Output when off	string {'0'}
OutputDataTypeScaling Mode	All ports same datatype	'Specify via dialog'   'Inherit via back propagation'   {'All ports same datatype'}
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
ConRadixGroup	Parameter scaling mode	{ 'Use specified scaling' }   'Best Precision: Vector-wise'
ZeroCross	Enable zero crossing detection	'off'   {'on'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}

**Discontinuities Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
Saturation (Saturate)		
UpperLimit	Upper limit	scalar or vector { '0.5' }
LowerLimit	Lower limit	scalar or vector { '-0.5' }
LinearizeAsGain	Treat as gain when linearizing	'off'   { 'on' }
ZeroCross	Enable zero crossing detection	'off'   { 'on' }
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Saturation Dynamic (Saturation Dynamic) (masked subsystem)		
Wrap To Zero (Wrap To Zero) (masked subsystem)		
Threshold	Threshold	string { '255' }

**Discrete Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Difference (Difference) (masked subsystem)		
ICPrevInput	Initial condition for previous input	string { '0.0' }
OutputDataTypeScaling Mode	Output data type and scaling	'Specify via dialog'   { 'Inherit via internal rule' }   'Inherit via back propagation'
OutDataType	Output data type: ex. sfix(16), uint(8), float('single')	string { 'sfix(16)' }
OutScaling	Output scaling: Slope or [Slope Bias] ex. 2^-9	string { '2^-10' }

**Discrete Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
LockScale	Lock output scaling so autoscaling tool can't change it	{ 'off' }   'on'
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   { 'Floor' }
DoSatur	Saturate to max or min when overflows occur	{ 'off' }   'on'
Discrete Derivative (Discrete Derivative) (masked subsystem)		
gainval	Gain value	string { '1.0' }
ICPrevScaledInput	Initial condition for previous weighted input $K*u/Ts$	string { '0.0' }
OutputDataTypeScaling Mode	Output data type and scaling	'Specify via dialog'   { 'Inherit via internal rule' }   'Inherit via back propagation'
OutDataType	Output data type: ex. sfix(16), uint(8), float('single')	string { 'sfix(16)' }
OutScaling	Output scaling: Slope or [Slope Bias] ex. $2^{-9}$	string { ' $2^{-10}$ ' }
LockScale	Lock output scaling so autoscaling tool can't change it	{ 'off' }   'on'
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   { 'Floor' }
DoSatur	Saturate to max or min when overflows occur	{ 'off' }   'on'
Discrete Filter (DiscreteFilter)		
Numerator	Numerator	vector { '[1]' }
Denominator	Denominator	vector { '[1 0.5]' }

**Discrete Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string {'1'}
StateIdentifier	State name	string {}
StateMustResolveToSignalObject	State name must resolve to Simulink signal object	{ 'off' }   'on'
RTWStateStorageClass	RTW storage class	{ 'Auto' }   'ExportedGlobal'   'ImportedExtern'   'ImportedExternPointer'
RTWStateStorageTypeQualifier	RTW storage type qualifier	string {}
Discrete State-Space (DiscreteStateSpace)		
A	A	string {'1'}
B	B	string {'1'}
C	C	string {'1'}
D	D	string {'1'}
X0	Initial conditions	string {'0'}
SampleTime	Sample time	string {'1'}
StateIdentifier	State name	string {}
StateMustResolveToSignalObject	State name must resolve to Simulink signal object	{ 'off' }   'on'
RTWStateStorageClass	RTW storage class	{ 'Auto' }   'ExportedGlobal'   'ImportedExtern'   'ImportedExternPointer'
RTWStateStorageTypeQualifier	RTW storage type qualifier	string {}
Discrete Transfer Fcn (DiscreteTransferFcn)		
Numerator	Numerator	vector {'[1]'} 

**Discrete Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Denominator	Denominator	vector {'[1 0.5]'}
SampleTime	Sample time (-1 for inherited)	string {'1'}
StateIdentifier	State name	string {}
StateMustResolveToSignalObject	State name must resolve to Simulink signal object	{'off'}   'on'
RTWStateStorageClass	RTW storage class	{'Auto'}   'ExportedGlobal'   'ImportedExtern'   'ImportedExternPointer'
RTWStateStorageTypeQualifier	RTW 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	{'off'}   'on'
RTWStateStorageClass	RTW storage class	{'Auto'}   'ExportedGlobal'   'ImportedExtern'   'ImportedExternPointer'
RTWStateStorageTypeQualifier	RTW storage type qualifier	string {}
Discrete-Time Integrator (DiscreteIntegrator)		

### Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
IntegratorMethod	Integrator method	{ 'Integration: Forward Euler' }   'Integration: Backward Euler'   'Integration: Trapezoidal'   'Accumulation: Forward Euler'   'Accumulation: Backward Euler'   'Accumulation: Trapezoidal'
gainval	Gain value	string { '1.0' }
ExternalReset	External reset	{ 'none' }   'rising'   'falling'   'either'   'level'
InitialConditionSource	Initial condition source	{ 'internal' }   'external'
InitialCondition	Initial condition	scalar or vector { '0' }
InitialConditionMode	Use initial condition as initial and reset value for	'State only (most efficient)'   { 'State and output' }
SampleTime	Sample time (-1 for inherited)	string { '1' }
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   { 'Inherit via internal rule' }   'Inherit via back propagation'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string { 'sfix(16)' }
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string { '2^0' }
LockScale	Lock output scaling against changes by the autoscaling tool	{ 'off' }   'on'
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   { 'Floor' }



**Discrete Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
SaturateOnIntegerOverflow	Saturate on integer overflow	{ 'off' }   'on'
LimitOutput	Limit output	{ 'off' }   'on'
UpperSaturationLimit	Upper saturation limit	scalar or vector {inf}
LowerSaturationLimit	Lower saturation limit	scalar or vector {-inf}
ShowSaturationPort	Show saturation port	{ 'off' }   'on'
ShowStatePort	Show state port	{ 'off' }   'on'
StateIdentifier	State name	string {}
StateMustResolveToSignalObject	State name must resolve to Simulink signal object	{ 'off' }   'on'
RTWStateStorageClass	RTW storage class	{ 'Auto' }   'ExportedGlobal'   'ImportedExtern'   'ImportedExternPointer'
RTWStateStorageTypeQualifier	RTW storage type qualifier	string {}
<b>First-Order Hold (First-Order Hold) (masked subsystem)</b>		
Ts	Sample time	string { '1' }
<b>Integer Delay (S-Function) (Integer Delay) (masked subsystem)</b>		
vinit	Initial condition	string { '0.0' }
samptime	Sample time	string { '-1' }
NumDelays	Number of delays	string { '4' }
<b>Memory (Memory)</b>		
X0	Initial condition	scalar or vector { '0' }
InheritSampleTime	Inherit sample time	{ 'off' }   'on'
LinearizeMemory	Direct feedthrough of input during linearization	{ 'off' }   'on'

### Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
StateIdentifier	State name	string {}
StateMustResolveToSignalObject	State name must resolve to Simulink signal object	{ 'off' }   'on'
RTWStateStorageClass	RTW storage class	{ 'Auto' }   'ExportedGlobal'   'ImportedExtern'   'ImportedExternPointer'
RTWStateStorageTypeQualifier	RTW storage type qualifier	string {}
Tapped Delay (S-Function) (Tapped Delay Line) (masked subsystem)		
vinit	Initial condition	string { '0.0' }
samptime	Sample time	string { '-1' }
NumDelays	Number of delays	string { '4' }
DelayOrder	Order output vector starting with	{ 'Oldest' }   'Newest'
includeCurrent	Include current input in output vector	{ 'off' }   'on'
Transfer Fcn (First Order Transfer Fcn) (masked subsystem)		
PoleZ	Pole (in Z plane)	string { '0.95' }
ICPrevOutput	Initial condition for previous output	string { '0.0' }
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   { 'Floor' }
DoSatur	Saturate to max or min when overflows occur	{ 'off' }   'on'
Transfer Fcn Lead or Lag (Lead or Lag Compensator) (masked subsystem)		
PoleZ	Pole of compensator (in Z plane)	string { '0.95' }

**Discrete Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
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' }
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
DoSatur	Saturate to max or min when overflows occur	{ 'off' }   'on'
Transfer Fcn Real Zero (Transfer Fcn Real Zero) (masked subsystem)		
ZeroZ	Zero (in Z plane)	string { '0.75' }
ICPrevInput	Initial condition for previous input	string { '0.0' }
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
DoSatur	Saturate to max or min when overflows occur	{ 'off' }   'on'
Unit Delay (UnitDelay)		
X0	Initial condition	scalar or vector { '0' }
SampleTime	Sample time (-1 for inherited)	string { '1' }
StateIdentifier	State name	string { }
StateMustResolveToSignalObject	State name must resolve to Simulink signal object	{ 'off' }   'on'
RTWStateStorageClass	RTW storage class	{ 'Auto' }   'ExportedGlobal'   'ImportedExtern'   'ImportedExternPointer'

### Discrete Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWStateStorageType Qualifier	RTW storage type qualifier	string {}
Weighted Moving Average (S-Function) (Weighted Moving Average) (masked subsystem)		
mgainval	Weights	string {'[0.1:0.1:1 0.9:-0.1:0.1]'} 
vinit	Initial condition	string {'0.0'} 
samptime	Sample time	string {'-1'} 
GainDataTypeScalingMode	Gain data type and scaling	'Specify via dialog'   {'Inherit via internal rule'}
GainDataType	Parameter data type: ex. sfix(16), uint(8), float('single')	string {'sfix( 16 )'} 
MatRadixGroup	Parameter scaling mode	'Use Specified Scaling'   'Best Precision: Element-wise'   'Best Precision: Row-wise'   'Best Precision: Column-wise'   {'Best Precision: Matrix-wise'}
GainScaling	Parameter scaling: Slope ex. 2 <sup>-9</sup>	string {'2 <sup>-10</sup> '}
OutputDataTypeScaling Mode	Output data type and scaling	'Specify via dialog'   {'Inherit via internal rule'}   'Inherit via back propagation'
OutDataType	Output data type: ex. sfix(16), uint(8), float('single')	string {'sfix(16)'} 
OutScaling	Output scaling: Slope or [Slope Bias] ex. 2 <sup>-9</sup>	string {'2 <sup>-10</sup> '}
LockScale	Lock output scaling so autoscaling tool can't change it	{ 'off' }   'on'
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   { 'Floor' }

**Discrete Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
DoSatur	Saturate to max or min when overflows occur	{ 'off' }   'on'
Zero-Order Hold (ZeroOrderHold)		
SampleTime	Sample time (-1 for inherited)	string { '1' }

**Logic and Bit Operations Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Bit Clear (Bit Clear) (masked subsystem)		
iBit	Index of bit (0 is least significant)	string { '0' }
Bit Set (Bit Set) (masked subsystem)		
iBit	Index of bit (0 is least significant)	string { '0' }
Bitwise Operator (S-Function) (Bitwise Operator) (masked subsystem)		
logicop	Operator	{ 'AND' }   'OR'   'NAND'   'NOR'   'XOR'   'NOT'
UseBitMask	Use bit mask ...	'off'   { 'on' }
NumInputPorts	Number of input ports	string { '1' }
BitMask	Bit mask	string { 'bin2dec('11011001')' }
BitMaskRealWorld	Treat mask as	'Real World Value'   { 'Stored Integer' }
Combinatorial Logic (CombinatorialLogic)		
TruthTable	Truth table	string { '[0 0;0 1;0 1;1 0;0 1;1 0;1 0;1 1]' }

**Logic and Bit Operations Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Compare To Constant (Compare To Constant) (masked subsystem)		
relop	Operator	'=='   '~='   '<'   {'<='}   '>='   '>'
const	Constant value	string { '3.0' }
LogicOutDataTypeMode	Output data type mode	{ 'uint8' }   'boolean'
Compare To Zero (Compare To Zero) (masked subsystem)		
relop	Operator	'=='   '~='   '<'   {'<='}   '>='   '>'
LogicOutDataTypeMode	Output data type mode	{ 'uint8' }   'boolean'
Detect Change (Detect Change) (masked subsystem)		
vinit	Initial condition	string { '0' }
Detect Decrease (Detect Decrease) (masked subsystem)		
vinit	Initial condition	string { '0.0' }
Detect Fall Negative (Detect Fall Negative) (masked subsystem)		
vinit	Initial condition	string { '0' }
Detect Fall Nonpositive (Detect Fall Nonpositive) (masked subsystem)		
vinit	Initial condition	string { '0' }
Detect Increase (Detect Increase) (masked subsystem)		
vinit	Initial condition	string { '0.0' }
Detect Rise Nonnegative (Detect Rise Nonnegative) (masked subsystem)		
vinit	Initial condition	string { '0' }
Detect Rise Positive (Detect Rise Positive) (masked subsystem)		
vinit	Initial condition	string { '0' }

**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	{ '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	{ 'Preserve fixed-point scaling' }   'Treat bit field as an integer'
Interval Test (Interval Test) (masked subsystem)		
IntervalClosedRight	Interval closed on right	'off'   { 'on' }
uplimit	Upper limit	string { '0.5' }
IntervalClosedLeft	Interval closed on left	'off'   { 'on' }
lowlimit	Lower limit	string { '-0.5' }
LogicOutDataTypeMode	Output data type mode	'uint8'   { 'boolean' }
Interval Test Dynamic (Interval Test Dynamic) (masked subsystem)		
IntervalClosedRight	Interval closed on right	'off'   { 'on' }
IntervalClosedLeft	Interval closed on left	'off'   { 'on' }
LogicOutDataTypeMode	Output data type mode	'uint8'   { 'boolean' }
Logical Operator (Logic)		
Operator	Operator	{ 'AND' }   'OR'   'NAND'   'NOR'   'XOR'   'NOT'
Inputs	Number of input ports	string { '2' }

### Logic and Bit Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
AllPortsSameDT	Require all inputs and output to have same data type	{ 'off' }   'on'
OutDataTypeMode	Output data type mode	{ 'Boolean' }   'Logical (see Advanced Sim. Parameters)'   'Specify via dialog'
LogicDataType	Output data type (e.g., uint(8), sint(32))	string { 'uint(8)' }
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Relational Operator (RelationalOperator)		
Operator	Relational Operator	'=='   '~='   '<'   {'<='}   '>='   '>'
InputSameDT	Require all inputs to have same data type	{ 'off' }   'on'
LogicOutDataTypeMode	Output data type mode	'uint8'   {'boolean'}
LogicDataType	Output data type (e.g., uint(8), sint(32))	string { 'uint(8)' }
ZeroCross	Enable zero crossing detection	'off'   {'on'}
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Shift Arithmetic (Shift Arithmetic) (masked subsystem)		
nBitShiftRight	Number of bits to shift right (use negative value to shift left)	string { '0' }
nBinPtShiftRight	Number of places by which binary point shifts right (use negative value to shift left)	string { '0' }



## Lookup Tables Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Cosine (Cosine) (masked subsystem)		
NumDataPoints	Number of data points for lookup table	string { '(2^5)+1' }
Direct Lookup Table (n-D) (S-Function) (LookupNDDirect) (masked subsystem)		
maskTabDims	Number of table dimensions	'1'   { '2' }   '3'   '4'   'More...'
explicitNumDims	Explicit number of table dimensions	string { '1' }
outDims	Inputs select this object from table	{ 'Element' }   'Column'   '2-D Matrix'
tabIsInput	Make table an input	{ 'off' }   'on'
mXTable	Table data	string { '[4 5 6;16 19 20;10 18 23]'
clipFlag	Action for out of range input	'None'   { 'Warning' }   'Error'
Interpolation (n-D) using PreLookup (LookupNDInterpIdx) (masked subsystem)		
numDimsPopupSelect	Number of table dimensions	'1'   { '2' }   '3'   '4'   'More...'
explicitNumDims	Explicit number of table dimensions	string { '2' }
table	Table data	string { 'sqrt([1:10]'.*[1:10])'
interpMethod	Interpolation method	'None - Flat'   { 'Linear' }
extrapMethod	Extrapolation method	'None - Flat'   { 'Linear' }

Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
rangeErrorMode	Action for out of range input	{ 'None' }   'Warning'   'Error'   'Error - No index checking in generated code'   'Warning - No index checking in generated code'   'None - No index checking in generated code'
NumSelectionDims	Number of sub-table selection dimensions	string { '0' }
Lookup Table (Lookup)		
InputValues	Vector of input values	vector { '[−5:5]' }
OutputValues	Vector of output values	vector { 'tanh([−5:5])' }
LookUpMeth	Look-up method	{ 'Interpolation-Extrapolation' }   'Interpolation-Use End Values'   'Use Input Nearest'   'Use Input Below'   'Use Input Above'
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   'Inherit via back propagation'   { 'Same as input' }
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string { 'sfix(16)' }
OutScaling	Output scaling value (Slope, e.g., 2 <sup>−9</sup> or [Slope Bias], e.g., [1.25 3])	string { '2^0' }
LockScale	Lock output scaling against changes by the autoscaling tool	{ 'off' }   'on'
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   { 'Floor' }

**Lookup Tables Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
SaturateOnIntegerOverflow	Saturate on integer overflow	{ 'off' }   'on'
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Lookup Table (2-D) (Lookup2D)		
RowIndex	Row index input values	string { '[1:3]' }
ColumnIndex	Column index input values	string { '[1:3]' }
OutputValues	Matrix of output values	string { '[4 5 6;16 19 20;10 18 23]' }
LookUpMeth	Lookup method	{ 'Interpolation-Extrapolation' }   'Interpolation-Use End Values'   'Use Input Nearest'   'Use Input Below'   'Use Input Above'
InputSameDT	Require all inputs to have same data type	'on'   { 'off' }
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   'Inherit via back propagation'   { 'Same as first input' }
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string { 'sfix(16)' }
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string { '2^0' }
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   { 'off' }
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   { 'Floor' }

### Lookup Tables Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Lookup Table (n-D) (LookupNDInterp) (masked subsystem)		
numDimsPopupSelect	Number of table dimensions	' 1 '   {' 2 ' }   ' 3 '   ' 4 '   'More...'
bp1	First input (row) breakpoint set	string {'[10,22,31]'}
bp2	Second (column) input breakpoint set	string {'[10,22,31]'}
bp3	Third input breakpoint set	string {'[1:3]'}
bp4	Fourth input breakpoint set	string {'[1:3]'}
bpcell	Fifth...Nth breakpoint sets (cell array)	string {'{ [1:3], [1:3] }'}
explicitNumDims	Explicit number of dimensions	string {'2'}
searchMode	Index search method	'Evenly Spaced Points'   'Linear Search'   {'Binary Search'}
cacheBpFlag	Begin index searches using previous index results	'on'   {'off'}
vectorInputFlag	Use one (vector) input port instead of N ports	'on'   {'off'}
tableData	Table data	string {'[4 5 6;16 19 20;10 18 23]'}
interpMethod	Interpolation method	'None - Flat'   {'Linear'}   'Cubic Spline'

**Lookup Tables Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
extrapMethod	Extrapolation method	'None - Clip'   {'Linear'}   'Cubic Spline'
rangeErrorMode	Action for out of range input	{ 'None' }   'Warning'   'Error'
Lookup Table Dynamic (Lookup Table Dynamic) (masked subsystem)		
LookUpMeth	Look-Up Method	'Interpolation-Extrapolation'   {'Interpolation-Use End Values'}   'Use Input Nearest'   'Use Input Below'   'Use Input Above'
OutputDataTypeScaling Mode	Output data type and scaling	{ 'Specify via dialog' }   'Inherit via back propagation'
OutDataType	Output data type: ex. sfix(16), uint(8), float('single')	string {'float('double')'}
OutScaling	Output scaling: Slope or [Slope Bias] ex. 2 <sup>-9</sup>	string {'2 <sup>-10</sup> '}
LockScale	Lock output scaling so autoscaling tool can't change it	'on'   {'off'}
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
DoSatur	Saturate to max or min when overflows occur	'on'   {'off'}
PreLookup Index Search (LookupIdxSearch) (masked subsystem)		
bpData	Breakpoint data	string {'[10:10:100]'}
searchMode	Index search method	'Evenly Spaced Points'   'Linear Search'   {'Binary Search'}
cacheBpFlag	Begin index search using previous index result	'on'   {'off'}
outputFlag	Output only the index	'on'   {'off'}

**Lookup Tables Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
IndexDataType	Index data type	{ 'uint32' }   'int32'
extrapMode	Process out of range input	'Clip to Range'   { 'Linear Extrapolation' }
rangeErrorMode	Action for out of range input	{ 'None' }   'Warning'   'Error'
Sine (Sine) (masked subsystem)		
NumDataPoints	Number of data points for lookup table	string { '(2^5)+1' }

**Math Operations Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Abs (Abs)		
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   { 'off' }
ZeroCross	Enable zero crossing detection	{ 'on' }   'off'
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Add (Sum)		
IconShape	Icon shape	{ 'rectangular' }   'round'
Inputs	List of signs	string { '++' }
InputSameDT	Require all inputs to have same data type	'on'   { 'off' }

**Math Operations Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   {'Inherit via internal rule'}   'Inherit via back propagation'   'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'} 
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2 <sup>-10</sup> '}
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   {'off'}
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Algebraic Constraint (Algebraic Constraint) (masked subsystem)		
z0	Initial guess	string {'0'}
Assignment (Assignment)		
InputType	Input type	{'Vector'}   'Matrix'
IndexMode	Index mode	'Zero-based'   {'One-based'}
IndexIsStartValue	Use index as starting value	'on'   {'off'}
ElementSrc	Source of element indices (E)	{'Internal'}   'External'

**Math Operations Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Elements	Elements (-1 for all elements)	string {'1'}
RowSrc	Source of row indices (R)	{ 'Internal' }   'External'
Rows	Rows (-1 for all rows)	string {'1'}
ColumnSrc	Source of column indices (C)	{ 'Internal' }   'External'
Columns	Columns (-1 for all columns)	string {'1'}
OutputInitialize	Output (Y)	'Initialize using input (U1)'   'Specify required dimensions'
OutputDimensions	Output dimensions	string {'[1 1]'}
DiagnosticForDimensions	Diagnostic if not all required dimensions are populated	{ 'Error' }   'Warning'   'None'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Bias (Bias)		
Bias	Bias	string {'0.0'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
Complex to Magnitude-Angle (ComplexToMagnitudeAngle)		
Output	Output	'Magnitude'   'Angle'   {'Magnitude and angle'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Complex to Real-Imag (ComplexToRealImag)		
Output	Output	'Real'   'Imag'   {'Real and imag'}



**Math Operations Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Divide (Product)		
Inputs	Number of inputs	string {'*/'}
Multiplication	Multiplication	{'Element-wise(*)'}   'Matrix(*)'
InputSameDT	Require all inputs to have same data type	'on'   {'off'}
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   {'Inherit via internal rule'}   'Inherit via back propagation'   'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'} string {'single'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2 <sup>-10</sup> '}
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   {'off'}
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnInteger Overflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Dot Product (Dot Product) (masked subsystem)		
InputSameDT	Require all inputs to have same data type	'on'   {'off'}

### Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
OutputDataTypeScaling Mode	Output data type mode	'Specify via dialog'   {'Inherit via internal rule'}   'Inherit via back propagation'   'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string {'2^-10'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   {'off'}
RndMeth	Round integer calculation toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
DoSatur	Saturate on integer overflow	'on'   {'off'}
Gain (Gain)		
Gain	Gain	string {'1'}
Multiplication	Multiplication	{ 'Element-wise(K.*u)' }   'Matrix(K*u)'   'Matrix(u*K)'   'Matrix(K*u) (u vector)'
ParameterDataTypeMode	Parameter data type mode	'Specify via dialog'   {'Inherit via internal rule'}   'Same as input'
ParameterDataType	Parameter data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
ParameterScalingMode	Parameter scaling mode	'Use specified scaling'   'Best Precision: Element-wise'   'Best Precision: Row-wise'   'Best Precision: Column-wise'   {'Best Precision: Matrix-wise'}

**Math Operations Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
ParameterScaling	Parameter scaling (Slope or [Slope Bias], e.g., $2^{-9}$ )	string {'2^0'}
OutDataTypeMode	Output data type mode	'Specify via dialog'   {'Inherit via internal rule'}   'Inherit via back propagation'   'Same as input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., $2^{-9}$ or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   {'off'}
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
<b>Magnitude-Angle to Complex (MagnitudeAngleToComplex)</b>		
Input	Input	'Magnitude'   'Angle'   {'Magnitude and angle'}
ConstantPart		string {'0'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
<b>Math Function (Math)</b>		

### Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Operator	Function	{ 'exp' }   'log'   '10^u'   'log10'   'magnitude^2'   'square'   'sqrt'   'pow'   'conj'   'reciprocal'   'hypot'   'rem'   'mod'   'transpose'   'hermitian'
OutputSignalType	Output signal type	{ 'auto' }   'real'   'complex'
SampleTime	Sample time (-1 for inherited)	string { '-1' }
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   'Inherit via internal rule'   'Inherit via back propagation'   { 'Same as first input' }
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string { 'sfix(16)' }
OutScaling	Output scaling value (Slope, e.g., 2^-9 or [Slope Bias], e.g., [1.25 3])	string { '2^0' }
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   { 'off' }
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   { 'Floor' }
SaturateOnIntegerOverflow	Saturate on integer overflow	{ 'on' }   'off'
Matrix Concatenation (Matrix Concatenation) (masked subsystem)		
numInputs	Number of inputs	string { '2' }
catMethod	Concatenation method	{ 'Horizontal' }   'Vertical'
MinMax (MinMax)		
Function	Function	{ 'min' }   'max'

**Math Operations Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Inputs	Number of input ports	string {'1'}
InputSameDT	Require all inputs to have same data type	'on'   {'off'}
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   {'Inherit via internal rule'}   'Inherit via back propagation'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   {'off'}
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
ZeroCross	Enable zero crossing detection	{'on'}   'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
MinMax Running Resettable (MinMax Running Resettable) (masked subsystem)		
Function	Function	{'min'}   'max'
vinit	Initial condition	string {'0.0'}
Polynomial (Polyval) (masked subsystem)		

### Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
coefs	Polynomial coefficients	string { '[ +2.081618890e-019, -1.441693666e-014, +4.719686976e-010, -8.536869453e-006, +1.621573104e-001, -8.087801117e+001 ]' }
Product (Product)		
Inputs	Number of inputs	string { '2' }
Multiplication	Multiplication	{ 'Element-wise(.*)' }   'Matrix(*)'
InputSameDT	Require all inputs to have same data type	'on'   { 'off' }
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   { 'Inherit via internal rule' }   'Inherit via back propagation'   'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string { 'sfix(16)' }
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string { '2^0' }
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   { 'off' }
RndMeth	Round integer calculations toward	{ 'Zero' }   'Nearest'   'Ceiling'   'Floor'
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   { 'off' }
SampleTime	Sample time (-1 for inherited)	string { '-1' }

**Math Operations Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
Product of Elements (Product)		
Inputs	Number of inputs	string {'*'}
Multiplication	Multiplication	{'Element-wise(.*)'}   'Matrix(*)'
InputSameDT	Require all inputs to have same data type	'on'   {'off'}
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   {'Inherit via internal rule'}   'Inherit via back propagation'   'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}   'single'   'double'
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2 <sup>-10</sup> '}
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   {'off'}
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Real-Imag toComplex (RealImagToComplex)		
Input	Input	'Real'   'Imag'   {'Real and imag'}
ConstantPart		string {'0'}

### Math Operations Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Reshape (Reshape) (masked subsystem)		
OutputDimensionality	Output dimensionality	{ '1-D array' }   'Column vector'   'Row vector'   'Customize'
OutputDimensions	Output dimensions	string {'[1,1]'}
Rounding Function (Rounding)		
Operator	Function	{ 'floor' }   'ceil'   'round'   'fix'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Sign (Signum)		
ZeroCross	Enable zero crossing detection	{ 'on' }   'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Sine Wave Function (Sin)		
SineType	Sine type	{ 'Time based' }   'Sample based'
TimeSource	Time (t)	'Use simulation time'   { 'Use external signal' }
Amplitude	Amplitude	string {'1'}
Bias	Bias	string {'0'}
Frequency	Frequency (rad/sec)	string {'1'}
Phase	Phase (rad)	string {'0'}
Samples	Samples per period	string {'10'}
Offset	Number of offset samples	string {'0'}
SampleTime	Sample time	string {'0'}



**Math Operations Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'
Slider Gain (Slider Gain) (masked subsystem)		
low	Low	string { '0' }
gain	Gain	string { '1' }
high	High	string { '2' }
Subtract (Sum)		
IconShape	Icon shape	{ 'rectangular' }   'round'
Inputs	List of signs	string { '+ - ' }
InputSameDT	Require all inputs to have same data type	'on'   { 'off' }
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   { 'Inherit via internal rule' }   'Inherit via back propagation'   'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string { 'sfix(16)' }
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string { '2 <sup>-10</sup> ' }
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   { 'off' }
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   { 'Floor' }
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   { 'off' }

**Math Operations Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Sum (Sum)		
IconShape	Icon shape	'rectangular'   {'round'}
Inputs	List of signs	string {' ++'}
InputSameDT	Require all inputs to have same data type	'on'   {'off'}
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   {'Inherit via internal rule'}   'Inherit via back propagation'   'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   {'off'}
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Sum of Elements (Sum)		
IconShape	Icon shape	{'rectangular'}   'round'
Inputs	List of signs	string {'+'}

**Math Operations Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
InputSameDT	Require all inputs to have same data type	'on'   {'off'}
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'Specify via dialog'   {'Inherit via internal rule'}   'Inherit via back propagation'   'Same as first input'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2 <sup>-10</sup> '}
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   {'off'}
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}

**Trigonometric Function (Trigonometry)**

Operator	Function	{ 'sin' }   'cos'   'tan'   'asin'   'acos'   'atan'   'atan2'   'sinh'   'cosh'   'tanh'   'asinh'   'acosh'   'atanh'
OutputSignalType	Output signal type	{ 'auto' }   'real'   'complex'
SampleTime	Sample time (-1 for inherited)	string {'-1'}

**Unary Minus (Unary Minus) (masked subsystem)**

**Math Operations Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
DoSatur	Saturate to max or min when overflows occur	'on'   {'off'}
Weighted Sample Time Math (Sample Time Math) (masked subsystem)		
TsampMathOp	Operation	{ '+'   '-'   '*'   '/'   'Ts Only'   '1/Ts Only' }
weightValue	Weight value	string { '1.0' }
TsampMathImp	Implement using	{ 'Online Calculations'   'Offline Scaling Adjustment' }
OutputDataTypeScaling Mode	Output data type and scaling	{ 'Inherit via internal rule'   'Inherit via back propagation' }
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
DoSatur	Saturate to max or min when overflows occur	'on'   {'off'}

**Model Verification Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Assertion (Assertion)		
Enabled	Enable assertion	{ 'on' }   'off'
AssertionFailFcn	Simulation callback when assertion fails	string { '' }
StopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Check Discrete Gradient (Checks_Gradient) (masked subsystem)		
gradient	Maximum gradient	string { '1' }

**Model Verification Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
enabled	Enable assertion	{ 'on' }   'off'
callback	Simulation callback when assertion fails (optional)	string { '' }
stopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'
export	Output assertion signal	'on'   { 'off' }
icon	Select icon type	{ 'graphic' }   'text'
Check Dynamic Gap (Checks_DGap) (masked subsystem)		
enabled	Enable assertion	{ 'on' }   'off'
callback	Simulation callback when assertion fails (optional)	string { '' }
stopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'
export	Output assertion signal	'on'   { 'off' }
icon	Select icon type	{ 'graphic' }   'text'
Check Dynamic Lower Bound (Checks_DMin) (masked subsystem)		
Enabled	Enable assertion	{ 'on' }   'off'
callback	Simulation callback when assertion fails (optional)	string { '' }
stopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'
export	Output assertion signal	'on'   { 'off' }
icon	Select icon type	{ 'graphic' }   'text'
Check Dynamic Range (Checks_DRange) (masked subsystem)		
enabled	Enable assertion	{ 'on' }   'off'
callback	Simulation callback when assertion fails (optional)	string { '' }

**Model Verification Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
stopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'
export	Output assertion signal	'on'   { 'off' }
icon	Select icon type	{ 'graphic' }   'text'
Check Dynamic Upper Bound (Checks_DMax) (masked subsystem)		
enabled	Enable assertion	{ 'on' }   'off'
callback	Simulation callback when assertion fails (optional)	string { '' }
stopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'
export	Output assertion signal	'on'   { 'off' }
icon	Select icon type	{ 'graphic' }   'text'
Check Input Resolution (Checks_Resolution) (masked subsystem)		
resolution	Resolution	string { '1' }
enabled	Enable assertion	{ 'on' }   'off'
callback	Simulation callback when assertion fails (optional)	string { '' }
stopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'
export	Output assertion signal	'on'   { 'off' }
Check Static Gap (Checks_SGap) (masked subsystem)		
max	Upper bound	string { '100' }
max_included	Inclusive upper bound	{ 'on' }   'off'
min	Lower bound	string { '0' }
min_included	Inclusive lower bound	{ 'on' }   'off'
enabled	Enable assertion	{ 'on' }   'off'

**Model Verification Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
callback	Simulation callback when assertion fails (optional)	string { '' }
stopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'
export	Output assertion signal	'on'   { 'off' }
icon	Select icon type	{ 'graphic' }   'text'
Check Static Lower Bound (Checks_SMin) (masked subsystem)		
min	Lower bound	string { '0' }
min_included	Inclusive boundary	{ 'on' }   'off'
enabled	Enable assertion	{ 'on' }   'off'
callback	Simulation callback when assertion fails (optional)	string { '' }
stopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'
export	Output assertion signal	'on'   { 'off' }
icon	Select icon type	{ 'graphic' }   'text'
Check Static Range (Checks_SRange) (masked subsystem)		
max	Upper bound	string { '100' }
max_included	Inclusive upper bound	{ 'on' }   'off'
min	Lower bound	string { '0' }
min_included	Inclusive lower bound	{ 'on' }   'off'
enabled	Enable assertion	{ 'on' }   'off'
callback	Simulation callback when assertion fails (optional)	string { '' }
stopWhenAssertionFail	Stop simulation when assertion fails	{ 'on' }   'off'

**Model Verification Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
export	Output assertion signal	'on'   {'off'}
icon	Select icon type	{'graphic'}   'text'
Check Static Upper Bound (Checks_SMax) (masked subsystem)		
max	Upper bound	string {'0'}
max_included	Inclusive boundary	{'on'}   'off'
enabled	Enable assertion	{'on'}   'off'
callback	Simulation callback when assertion fails (optional)	string {''}
stopWhenAssertionFail	Stop simulation when assertion fails	{'on'}   'off'
export	Output assertion signal	'on'   {'off'}
icon	Select icon type	{'graphic'}   'text'

**Model-Wide Utilities Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Block Support Table(Block Support Table) (masked subsystem)		
DocBlock (DocBlock) (masked subsystem)		
ECoderFlag	RTW Embedded Coder Flag	string {''}
DocumentType	Document Type	{'Text'}   'RTF'   'HTML'
Model Advisor (Model Advisor) (masked subsystem)		
Model Info (CMBlock) (masked subsystem)		
InitialSaveTempField	InitialSaveTempField	string {''}
InitialBlockCM	InitialBlockCM	string {'None'}



**Model-Wide Utilities Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
BlockCM	BlockCM	string { 'None' }
Frame	Frame	string { 'on' }
SaveTempField	SaveTempField	string { '' }
DisplayStringWithTags	DisplayStringWithTags	string { 'Model Info' }
MaskDisplayString	MaskDisplayString	string { 'Model Info' }
HorizontalTextAlignment	HorizontalTextAlignment	string { 'Center' }
LeftAlignmentValue	LeftAlignmentValue	string { '0.5' }
SourceBlockDiagram	SourceBlockDiagram	string { 'untitled' }
TagMaxNumber	TagMaxNumber	string { '20' }
CMTag1	CMTag1	string { '' }
CMTag2	CMTag2	string { '' }
CMTag3	CMTag3	string { '' }
CMTag4	CMTag4	string { '' }
CMTag5	CMTag5	string { '' }
CMTag6	CMTag6	string { '' }
CMTag7	CMTag7	string { '' }
CMTag8	CMTag8	string { '' }
CMTag9	CMTag9	string { '' }
CMTag10	CMTag10	string { '' }
CMTag11	CMTag11	string { '' }
CMTag12	CMTag12	string { '' }
CMTag13	CMTag13	string { '' }
CMTag14	CMTag14	string { '' }

**Model-Wide Utilities Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
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 (oflinearized model)	string { '0' }
Trigger-Based Linearization (Triggered Linearization) (masked subsystem)		
TriggerType	Trigger type	{ 'rising' }   'falling'   'either'   'function-call'
SampleTime	Sample time (oflinearized model)	string { '0' }

**Ports & Subsystems Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Configurable Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{ 'on' }   'off'
BlockChoice	Block choice	{ '' }
TemplateBlock	Template block	string { 'self' }
MemberBlocks	Member blocks	string { '' }
Permissions	Read/write permissions	{ 'ReadWrite' }   'ReadOnly'   'NoReadOrWrite'

**Ports & Subsystems Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
ErrorFcn	Name of error callback function	string { '' }
PermitHierarchicalResolution	Permit hierarchical resolution	{ 'All' }   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	'on'   { 'off' }
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   { 'off' }
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{ 'on' }   'off'
SystemSampleTime	Sample time (-1 for inherited)	string { '-1' }
RTWSystemCode	RTW system code	{ 'Auto' }   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{ 'Auto' }   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string { '' }
RTWFileNameOpts	RTW filename options	{ 'Auto' }   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string { '' }
DataTypeOverride		{ 'UseLocalSettings' }   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'
MinMaxOverflowLogging		{ 'UseLocalSettings' }   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
Atomic Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{ 'on' }   'off'

### Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
BlockChoice	Block choice	{ '' }
TemplateBlock	Template block	string { '' }
MemberBlocks	Member blocks	string { '' }
Permissions	Read/write permissions	{ 'ReadWrite' }   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string { '' }
PermitHierarchicalResolution	Permit hierarchical resolution	{ 'All' }   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{ 'on' }   'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   { 'off' }
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{ 'on' }   'off'
SystemSampleTime	Sample time (-1 for inherited)	string { '-1' }
RTWSystemCode	RTW system code	{ 'Auto' }   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{ 'Auto' }   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string { '' }
RTWFileNameOpts	RTW filename options	{ 'Auto' }   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string { '' }
DataTypeOverride		{ 'UseLocalSettings' }   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'

**Ports & Subsystems Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
MinMaxOverflowLogging		{ 'UseLocalSettings' }   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
Code Reuse Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{ 'on' }   'off'
BlockChoice	Block choice	{ '' }
TemplateBlock	Template block	string { '' }
MemberBlocks	Member blocks	string { '' }
Permissions	Read/write permissions	{ 'ReadWrite' }   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string { '' }
PermitHierarchicalResolution	Permit hierarchical resolution	{ 'All' }   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{ 'on' }   'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   { 'off' }
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{ 'on' }   'off'
SystemSampleTime	Sample time (-1 for inherited)	string { '-1' }
RTWSystemCode	RTW system code	'Auto'   'Inline'   'Function'   { 'Reusable function' }
RTWFcnNameOpts	RTW function name options	'Auto'   { 'Use subsystem name' }   'User specified'
RTWFcnName	RTW function name	string { '' }

### Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWFileNameOpts	RTW filename options	'Auto'   'Use subsystem name'   {'Use function name'}   'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'}   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'}   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
Enable (EnablePort)		
StatesWhenEnabling	States when enabling	{'held'}   'reset'
ShowOutputPort	Show output port	'on'   {'off'}
ZeroCross	Enable zero crossing detection	{'on'}   'off'
Enabled and Triggered Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'}   'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'}   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchicalResolution	Permit hierarchical resolution	{'All'}   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'}   'off'

**Ports & Subsystems Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   {'off'}
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{ 'on' }   'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{ 'Auto' }   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{ 'Auto' }   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{ 'Auto' }   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{ 'UseLocalSettings' }   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'
MinMaxOverflowLogging		{ 'UseLocalSettings' }   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
<b>EnabledSubsystem (SubSystem)</b>		
ShowPortLabels	Show port labels	{ 'on' }   'off'
BlockChoice	Block choice	{ '' }
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{ 'ReadWrite' }   'ReadOnly'   'NoReadOrWrite'

### Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
ErrorFcn	Name of error callback function	string {''}
PermitHierarchicalResolution	Permit hierarchical resolution	{'All'}   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'}   'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   {'off'}
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{'on'}   'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'}   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'}   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'}   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'}   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'}   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
For Iterator Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'}   'off'



**Ports & Subsystems Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
BlockChoice	Block choice	{ '' }
TemplateBlock	Template block	string { '' }
MemberBlocks	Member blocks	string { '' }
Permissions	Read/write permissions	{ 'ReadWrite' }   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string { '' }
PermitHierarchicalResolution	Permit hierarchical resolution	{ 'All' }   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{ 'on' }   'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   { 'off' }
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{ 'on' }   'off'
SystemSampleTime	Sample time (-1 for inherited)	string { '-1' }
RTWSystemCode	RTW system code	{ 'Auto' }   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{ 'Auto' }   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string { '' }
RTWFileNameOpts	RTW filename options	{ 'Auto' }   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string { '' }
DataTypeOverride		{ 'UseLocalSettings' }   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
MinMaxOverflowLogging		{ 'UseLocalSettings' }   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
Function-Call Generator (Function-Call Generator) (masked subsystem)		
sample_time	Sample time	string { '1' }
numberOfIterations	Number of iterations	string { '1' }
Function-Call Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{ 'on' }   'off'
BlockChoice	Block choice	{ '' }
TemplateBlock	Template block	string { '' }
MemberBlocks	Member blocks	string { '' }
Permissions	Read/write permissions	{ 'ReadWrite' }   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string { '' }
PermitHierarchicalResolution	Permit hierarchical resolution	{ 'All' }   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{ 'on' }   'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   { 'off' }
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{ 'on' }   'off'
SystemSampleTime	Sample time (-1 for inherited)	string { '-1' }
RTWSystemCode	RTW system code	{ 'Auto' }   'Inline'   'Function'   'Reusable function'

**Ports & Subsystems Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
RTWFcnNameOpts	RTW function name options	{ 'Auto' }   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string { '' }
RTWFileNameOpts	RTW filename options	{ 'Auto' }   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string { '' }
DataTypeOverride		{ 'UseLocalSettings' }   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'
MinMaxOverflowLogging		{ 'UseLocalSettings' }   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
If (If)		
NumInputs	Number of inputs	string { '1' }
IfExpression	If expression (e.g., u1 ~= 0)	string { 'u1 > 0' }
ElseIfExpressions	Elseif expressions (comma-separated list, e.g., u2 ~= 0, u3(2) < u2)	string { '' }
ShowElse	Show else condition	{ 'on' }   'off'
ZeroCross	Enable zero crossing detection	{ 'on' }   'off'
SampleTime	Sample time (-1 for inherited)	string { '-1' }
If Action Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{ 'on' }   'off'
BlockChoice	Block choice	{ '' }

Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'}   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchicalResolution	Permit hierarchical resolution	{'All'}   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'}   'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   {'off'}
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{'on'}   'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'}   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'}   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'}   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'}   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'

**Ports & Subsystems Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
MinMaxOverflowLogging		{'UseLocalSettings'}   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
In1 (Inport)		
Port	Port number	string {'1'}
IconDisplay	Icon display	'Signal name'   {'Port number'}   'Port number and signal name'
UseBusObject	Specify properties via bus object	'on'   {'off'}
BusObject	Bus object for validating input bus	string {'BusObject'}
BusOutputAsStruct	Output as nonvirtual bus	'on'   {'off'}
PortDimensions	Port dimensions (-1 for inherited)	string {'-1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
DataType	Data type	{'auto'}   'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'}   'real'   'complex'
SamplingMode	Sampling mode	{'auto'}   'Sample based'   'Frame based'
LatchInput	Latch (buffer) input	'on'   {'off'}

**Ports & Subsystems Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
Interpolate	Interpolate data	{ 'on' }   'off'
Model (ModelReference)		
ModelName	Model name (without the .mdl extension)	string { '<Enter Model Name>' }
ParameterArgumentNames	Model arguments	string { '' }
ParameterArgumentValues	Model argument values (for this instance)	string { '' }
AvailSigsInstanceProps		handle vector { '' }
AvailSigsDefaultProps		handle vector { '' }
DefaultDataLogging		'on'   { 'off' }
Out1 (Output)		
Port	Port number	string { '1' }
IconDisplay	Icon display	'Signal name'   { 'Port number' }   'Port number and signal name'
UseBusObject	Specify properties via bus object	'on'   { 'off' }
BusObject	Bus object for validating input bus	string { 'BusObject' }
BusOutputAsStruct	Output as nonvirtual bus in parent model	'on'   { 'off' }
PortDimensions	Port dimensions (-1 for inherited)	string { '-1' }
SampleTime	Sample time (-1 for inherited)	string { '-1' }
DataType	Data type	{ 'auto' }   'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Specify via dialog'

**Ports & Subsystems Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'} }
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'}   'real'   'complex'
SamplingMode	Sampling mode	{'auto'}   'Sample based'   'Frame based'
OutputWhenDisabled	Output when disabled	{'held'}   'reset'
InitialOutput	Initial output	string {'[]'}
Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'}   'off'
BlockChoice	Block choice	{''}
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'}   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchicalResolution	Permit hierarchical resolution	{'All'}   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	'on'   {'off'}
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   {'off'}
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{'on'}   'off'

**Ports & Subsystems Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'}   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'}   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'}   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'}   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'
MinMaxOverflowLogging		{'UseLocalSettings'}   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
Switch Case (SwitchCase)		
CaseConditions	Case conditions (e.g., {1,[2,3]})	string {'{1}'}
CaseShowDefault	Show default case	{'on'}   'off'
ZeroCross	Enable zero crossing detection	{'on'}   'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Switch Case Action Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{'on'}   'off'
BlockChoice	Block choice	{' '}



**Ports & Subsystems Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
TemplateBlock	Template block	string {''}
MemberBlocks	Member blocks	string {''}
Permissions	Read/write permissions	{'ReadWrite'}   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string {''}
PermitHierarchicalResolution	Permit hierarchical resolution	{'All'}   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{'on'}   'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   {'off'}
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{'on'}   'off'
SystemSampleTime	Sample time (-1 for inherited)	string {'-1'}
RTWSystemCode	RTW system code	{'Auto'}   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{'Auto'}   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string {''}
RTWFileNameOpts	RTW filename options	{'Auto'}   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string {''}
DataTypeOverride		{'UseLocalSettings'}   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'

### Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
MinMaxOverflowLogging		{ 'UseLocalSettings' }   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
Trigger (TriggerPort)		
TriggerType	Trigger type	{ 'rising' }   'falling'   'either'   'function-call'
StatesWhenEnabling	States when enabling	{ 'held' }   'reset'   'inherit'
ShowOutputPort	Show output port	'on'   { 'off' }
OutputDataType	Output data type	{ 'auto' }   'double'   'int8'
SampleTimeType	Sample time type	{ 'triggered' }   'periodic'
SampleTime	Sample time	string { '1' }
ZeroCross	Enable zero crossing detection	{ 'on' }   'off'
Triggered Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{ 'on' }   'off'
BlockChoice	Block choice	{ '' }
TemplateBlock	Template block	string { '' }
MemberBlocks	Member blocks	string { '' }
Permissions	Read/write permissions	{ 'ReadWrite' }   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string { '' }
PermitHierarchicalResolution	Permit hierarchical resolution	{ 'All' }   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{ 'on' }   'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   { 'off' }

**Ports & Subsystems Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{ 'on' }   'off'
SystemSampleTime	Sample time (-1 for inherited)	string { '-1' }
RTWSystemCode	RTW system code	{ 'Auto' }   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{ 'Auto' }   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string { '' }
RTWFileNameOpts	RTW filename options	{ 'Auto' }   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string { '' }
DataTypeOverride		{ 'UseLocalSettings' }   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'
MinMaxOverflowLogging		{ 'UseLocalSettings' }   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'
While Iterator Subsystem (SubSystem)		
ShowPortLabels	Show port labels	{ 'on' }   'off'
BlockChoice	Block choice	{ '' }
TemplateBlock	Template block	string { '' }
MemberBlocks	Member blocks	string { '' }
Permissions	Read/write permissions	{ 'ReadWrite' }   'ReadOnly'   'NoReadOrWrite'
ErrorFcn	Name of error callback function	string { '' }

### Ports & Subsystems Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
PermitHierarchicalResolution	Permit hierarchical resolution	{ 'All' }   'ParametersOnly'   'None'
TreatAsAtomicUnit	Treat as atomic unit	{ 'on' }   'off'
MinAlgLoopOccurrences	Minimize algebraic loop occurrences	'on'   { 'off' }
PropExecContextAcrossSSBoundary	Propagate execution context across subsystem boundary	{ 'on' }   'off'
SystemSampleTime	Sample time (-1 for inherited)	string { '-1' }
RTWSystemCode	RTW system code	{ 'Auto' }   'Inline'   'Function'   'Reusable function'
RTWFcnNameOpts	RTW function name options	{ 'Auto' }   'Use subsystem name'   'User specified'
RTWFcnName	RTW function name	string { '' }
RTWFileNameOpts	RTW filename options	{ 'Auto' }   'Use subsystem name'   'Use function name'   'User specified'
RTWFileName	RTW filename (no extension)	string { '' }
DataTypeOverride		{ 'UseLocalSettings' }   'ScaledDoubles'   'TrueDoubles'   'TrueSingles'   'ForceOff'
MinMaxOverflowLogging		{ 'UseLocalSettings' }   'MinMaxAndOverflow'   'OverflowOnly'   'ForceOff'

**Signal Attributes Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Data Type Conversion (DataTypeConversion)		
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Specify via dialog'   {'Inherit via back propagation'}
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
LockScale	Lock output scaling against changes by the autoscaling tool	'on'   {'off'}
ConvertRealWorld	Input and output to have equal	{'Real World Value (RWV)'}   'Stored Integer (SI)'
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Data Type Conversion Inherited (Conversion Inherited) (masked subsystem)		
ConvertRealWorld	Input and Output to have equal	{'Real World Value'}   'Stored Integer'
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
DoSatur	Saturate to max or min when overflows occur	'on'   {'off'}

### Signal Attributes Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Data Type Duplicate (Data Type Duplicate) (masked subsystem)		
NumInputPorts	Number of input ports	string {'2'}
Data Type Propagation (Data Type Propagation) (masked subsystem)		
PropDataTypeMode	1. Propagated data type	'Specify via dialog'   {'Inherit via propagation rule'}
PropDataType	1.1. Propagated datatype: ex. sfix(16), uint(8), float('single')	string {'sfix(16)'}
IfRefDouble	1.1. If any reference input is double, output is	{'double'}   'single'
IfRefSingle	1.2. If any reference input is single, output is	'double'   {'single'}
IsSigned	1.3. Is-Signed	'IsSigned1'   'IsSigned2'   {'IsSigned1 or IsSigned2'}   'TRUE'   'FALSE'
NumBitsBase	1.4.1. Number-of-Bits: Base	'NumBits1'   'NumBits2'   {'max([NumBits1 NumBits2])'}   'min([NumBits1 NumBits2])'   'NumBits1+NumBits2'
NumBitsMult	1.4.2. Number-of-Bits: Multiplicative adjustment	string {'1'}
NumBitsAdd	1.4.3. Number-of-Bits: Additive adjustment	string {'0'}
NumBitsAllowFinal	1.4.4. Number-of-Bits: Allowable final values	string {'1:128'}
PropScalingMode	2. Propagated scaling	'Specify via dialog'   {'Inherit via propagation rule'}   'Obtain via best precision'
PropScaling	2.1. Propagated scaling: Slope or [Slope Bias] ex. 2 <sup>-9</sup>	string {'2 <sup>-10</sup> '}

**Signal Attributes Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
ValuesUsedBestPrec	2.1. Values used to determine best precision scaling.	string { ' [5 -7] ' }
SlopeBase	2.1.1. Slope: Base	'Slope1'   'Slope2'   'max([Slope1 Slope2])'   {'min([Slope1 Slope2])'}   'Slope1*Slope2'   'Slope1/Slope2'   'PosRange1'   'PosRange2'   'max([PosRange1 PosRange2])'   'min([PosRange1 PosRange2])'   'PosRange1*PosRange2'   'PosRange1/PosRange2'
SlopeMult	2.1.2. Slope: Multiplicative adjustment	string { '1' }
SlopeAdd	2.1.3. Slope: Additive adjustment	string { '0' }
BiasBase	2.2.1. Bias: Base	{ 'Bias1' }   'Bias2'   'max([Bias1 Bias2])'   'min([Bias1 Bias2])'   'Bias1*Bias2'   'Bias1/Bias2'   'Bias1+Bias2'   'Bias1-Bias2'
BiasMult	2.2.2. Bias: Multiplicative adjustment	string { '1' }
BiasAdd	2.2.3. Bias: Additive adjustment	string { '0' }
Data Type Scaling Strip (Scaling Strip) (masked subsystem)		
IC (InitialCondition)		
Value	Initial value	string { '1' }
SampleTime	Sample time (-1 for inherited)	string { '-1' }
Probe (Probe)		
ProbeWidth	Probe width	{ 'on' }   'off'
ProbeSampleTime	Probe sample time	{ 'on' }   'off'

**Signal Attributes Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
ProbeComplexSignal	Detect complex signal	{ 'on' }   'off'
ProbeSignalDimensions	Probe signal dimensions	{ 'on' }   'off'
ProbeFramedSignal	Detect framed signal	{ 'on' }   'off'
ProbeWidthDataType	Data type for width	{ 'double' }   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Same as input'
ProbeSampleTimeDataType	Data type for sample time	{ 'double' }   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Same as input'
ProbeComplexityDataType	Data type for signal complexity	{ 'double' }   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Same as input'
ProbeDimensionsDataType	Data type for signal dimensions	{ 'double' }   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Same as input'
ProbeFrameDataType	Data type for signal frames	{ 'double' }   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Same as input'
Rate Transition (RateTransition)		
Integrity	Ensure data integrity during data transfer	{ 'on' }   'off'
Deterministic	Ensure deterministic data transfer (maximum delay)	{ 'on' }   'off'
X0	Initial conditions	string { '0' }
OutPortSampleTime	Output port sample time	string { '-1' }
Signal Conversion (SignalConversion)		



**Signal Attributes Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
ConversionOutput	Output	{'Contiguous copy'}   'Bus copy'   'Virtual bus'   'Nonvirtual bus'
OverrideOpt	Override optimizations and always copy signal	'on'   {'off'}
Signal Specification (SignalSpecification)		
Dimensions	Dimensions (-1 for inherited)	string {'-1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
DataType	Data type	{'auto'}   'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'}   'real'   'complex'
SamplingMode	Sampling mode	{'auto'}   'Sample based'   'Frame based'
Weighted Sample Time (Sample Time Math) (masked subsystem)		
TsampMathOp	Operation	'+'   '-'   '*'   '/'   {'Ts Only'}   '1/Ts Only'
weightValue	Weight value	string {'1.0'}
TsampMathImp	Implement using	{'Online Calculations'}   'Offline Scaling Adjustment'
OutputDataTypeScaling Mode	Output data type and scaling	{'Inherit via internal rule'}   'Inherit via back propagation'

**Signal Attributes Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
DoSatur	Saturate to max or min when overflows occur	'on'   {'off'}
Width (Width)		
OutputDataTypeScaling Mode	Output data type mode	{'Choose intrinsic data type'   'Inherit via back propagation'   'All ports same datatype'}
DataType	Output data type	{'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'}

**Signal Routing Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
BusCreator (BusCreator)		
Inputs	Number of inputs	string {'2'}
DisplayOption	Display option	'none'   'signals'   {'bar'}
UseBusObject	Specify properties via bus object	'on'   {'off'}
BusObject	Bus object for validating input bus	string {'BusObject'}
NonVirtualBus	Output as nonvirtual bus	'on'   {'off'}
Bus Selector (BusSelector)		
OutputSignals	Output signals	string {'signal1,signal2'}
MuxedOutput	Muxed output	'on'   {'off'}
InputSignals		matrix {'{'}

**Signal Routing Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Data Store Memory (DataStoreMemory)		
DataStoreName	Data store name	string {'A'}
InitialValue	Initial value	string {'0'}
StateMustResolveToSignalObject	Data store name must resolve to Simulink signal object	'on'   {'off'}
RTWStateStorageClass	RTW storage class	{'Auto'}   'ExportedGlobal'   'ImportedExtern'   'ImportedExternPointer'
RTWStateStorageTypeQualifier	RTW type qualifier	string {''}
VectorParams1D	Interpret vector parameters as 1-D	{'on'}   'off'
ShowAdditionalParam	Show additional parameters	'on'   {'off'}
DataType	Data type	{'auto'}   'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'}   'real'   'complex'
Data Store Read (DataStoreRead)		
DataStoreName	Data store name	string {'A'}
SampleTime	Sample time	string {'0'}
Data Store Write (DataStoreWrite)		

### Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
DataStoreName	Data store name	string {'A'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Demux (Demux)		
Outputs	Number of outputs	string {'2'}
DisplayOption	Display option	'none'   {'bar'}
BusSelectionMode	Bus selection mode	'on'   {'off'}
Environment Controller (Environment Controller) (masked subsystem)		
From (From)		
GotoTag	Goto tag	string {'A'}
IconDisplay	Icon display	'Signal name'   {'Tag'}   'Tag and signal name'
Goto (Goto)		
GotoTag	Tag	string {'A'}
IconDisplay	Icon display	'Signal name'   {'Tag'}   'Tag and signal name'
TagVisibility	Tag visibility	{'local'}   'scoped'   'global'
Goto Tag Visibility (GotoTagVisibility)		
GotoTag	Goto tag	string {'A'}
Index Vector (MultiPortSwitch)		
Inputs	Number of inputs	string {'1'}
zeroidx	Use zero-based indexing	{'on'}   'off'
InputSameDT	Require all data port inputs to have same data type	'on'   {'off'}

**Signal Routing Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
OutDataTypeMode	Output data type mode	{'Inherit via internal rule'}   'Inherit via back propagation'
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Manual Switch (Manual Switch) (masked subsystem)		
sw	Current setting	string {'1'}
action	Action	string {'0'}
Merge (Merge)		
Inputs	Number of inputs	string {'2'}
InitialOutput	Initial output	string {'[]'}
AllowUnequalInputPortWidths	Allow unequal port widths	'on'   {'off'}
InputPortOffsets	Input port offsets	string {'[]'}
Multiport Switch (MultiPortSwitch)		
Inputs	Number of inputs	string {'3'}
zeroidx	Use zero-based indexing	'on'   {'off'}
InputSameDT	Require all data port inputs to have same data type	'on'   {'off'}
OutDataTypeMode	Output data type mode	{'Inherit via internal rule'}   'Inherit via back propagation'
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}

### Signal Routing Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Mux (Mux)		
Inputs	Number of inputs	string {'2'}
DisplayOption	Display option	'none'   'signals'   {'bar'}
UseBusObject	Specify properties via bus object	'on'   {'off'}
BusObject	Bus object for validating input bus	string {'BusObject'}
NonVirtualBus	Output as nonvirtual bus	'on'   {'off'}
Selector (Selector)		
InputType	Input type	{'Vector'}   'Matrix'
IndexMode	Index mode	'Zero-based'   {'One-based'}
ElementSrc	Source of element indices (E)	{'Internal'}   'External'
Elements	Elements (-1 for all elements)	string {'[1 3]'}
RowSrc	Source of row indices (R)	{'Internal'}   'External'
Rows	Rows (-1 for all rows)	string {'1'}
ColumnSrc	Source of column indices (C)	{'Internal'}   'External'
Columns	Columns (-1 for all columns)	string {'1'}
InputPortWidth	Input port width	string {'3'}

**Signal Routing Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
IndexIsStartValue	Use index as starting value	'on'   {'off'}
OutputPortSize	Output port dimensions	string {'1'}
Switch (Switch)		
Criteria	Criteria for passing first input	{ 'u2 >= Threshold' }   'u2 > Threshold'   'u2 ~= 0'
Threshold	Threshold	string {'0'}
InputSameDT	Require all data port inputs to have same data type	'on'   {'off'}
OutDataTypeMode	Output data type mode	{ 'Inherit via internal rule' }   'Inherit via back propagation'
RndMeth	Round integer calculations toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
SaturateOnIntegerOverflow	Saturate on integer overflow	'on'   {'off'}
ZeroCross	Enable zero crossing detection	{ 'on' }   'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}

**Sinks Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Display (Display)		
Format	Format	{ 'short' }   'long'   'short_e'   'long_e'   'bank'   'hex (Stored Integer)'   'binary (Stored Integer)'   'decimal (Stored Integer)'   'octal (Stored Integer)'

### Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Decimation	Decimation	string {'1'}
Floating	Floating display	'on'   {'off'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
Floating Scope (Scope)		
Floating		{ 'on' }   'off'
Location		rectangle {'[376 294 700 533]'} 
Open		'on'   {'off'}
NumInputPorts		string {'1'}
TickLabels		'on'   'off'   {'OneTimeTick'}
ZoomMode		{ 'on' }   'xonly'   'yonly'
AxesTitles		list
Grid		'off'   {'on'}   'xonly'   'yonly'
TimeRange		string {'auto'}
YMin		string {'-5'}
YMax		string {'5'}
SaveToWorkspace		'on'   {'off'}
SaveName		string {'ScopeData'}
DataFormat		{ 'StructureWithTime' }   'Structure'   'Array'
LimitDataPoints		{ 'on' }   'off'
MaxDataPoints		string {'5000'}
Decimation		string {'1'}
SampleInput		'on'   {'off'}



**Sinks Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
SampleTime		string {'0'}
Out1 (Output)		
Port	Port number	string {'1'}
IconDisplay	Icon display	'Signal name'   {'Port number'}   'Port number and signal name'
UseBusObject	Specify properties via bus object	'on'   {'off'}
BusObject	Bus object for validating input bus	string {'BusObject'}
BusOutputAsStruct	Output as nonvirtual bus in parent model	'on'   {'off'}
PortDimensions	Port dimensions (-1 for inherited)	string {'-1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
DataType	Data type	{'auto'}   'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'}   'real'   'complex'
SamplingMode	Sampling mode	{'auto'}   'Sample based'   'Frame based'
OutputWhenDisabled	Output when disabled	{'held'}   'reset'
InitialOutput	Initial output	string {'[]'}

Sinks Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Scope (Scope)		
Floating		'on'   {'off'}
Location		rectangle {'[188 390 512 629]'}
Open		'on'   {'off'}
NumInputPorts		string {'1'}
TickLabels		'on'   'off'   {'OneTimeTick'}
ZoomMode		{'on'}   'xonly'   'yonly'
AxesTitles		list
Grid		'off'   {'on'}   'xonly'   'yonly'
TimeRange		string {'auto'}
YMin		string {'-5'}
YMax		string {'5'}
SaveToWorkspace		'on'   {'off'}
SaveName		string {'ScopeData1'}
DataFormat		{'StructureWithTime'}   'Structure'   'Array'
LimitDataPoints		{'on'}   'off'
MaxDataPoints		string {'5000'}
Decimation		string {'1'}
SampleInput		'on'   {'off'}
SampleTime		string {'0'}
Stop Simulation		
Terminator		
To File (ToFile)		

**Sinks Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
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'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
SaveFormat	Save format	'Structure With Time'   {'Structure'}   'Array'
FixptAsFi	Log fixed-point data as a fi object	'on'   {'off'}
XY Graph (XY scope) (masked subsystem)		
xmin	x-min	string {'-1'}
xmax	x-max	string {'1'}
ymin	y-min	string {'-1'}
ymax	y-max	string {'1'}
st	Sample time	string {'-1'}

Sources Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Band-Limited White Noise (Band-Limited White Noise) (masked subsystem)		
Cov	Noise power	string { '[0.1]' }
Ts	Sample time	string { '0.1' }
seed	Seed	string { '[23341]' }
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'
Chirp Signal (chirp) (masked subsystem)		
f1	Initial frequency (Hz)	string { '0.1' }
T	Target time (secs)	string { '100' }
f2	Frequency at target time (Hz)	string { '1' }
VectorParams1D	Interpret vectors parameters as 1-D	{ 'on' }   'off'
Clock (Clock)		
DisplayTime	Display time	'on'   { 'off' }
Decimation	Decimation	string { '10' }
Constant (Constant)		
Value	Constant value	string { '1' }
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'
OutDataTypeMode	Output data type mode	'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Specify via dialog'   { 'Inherit from 'Constant value'' }   'Inherit via back propagation'

**Sources Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'} 
ConRadixGroup	Output scaling mode	{'Use specified scaling'}   'Best Precision: Vector-wise'
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SampleTime	Sample time	string {'inf'}
Counter Free-Running (Counter Free-Running) (masked subsystem)		
NumBits	Number of Bits	string {'16'}
tsamp	Sample time	string {'-1'}
Counter Limited (Counter Limited) (masked subsystem)		
uplimit	Upper limit	string {'7'}
tsamp	Sample time	string {'-1'}
Digital Clock (DigitalClock)		
SampleTime	Sample time	string {'1'}
From File (FromFile)		
FileName	Filename	string {'untitled.mat'}
SampleTime	Sample time	string {'0'}
From Workspace (FromWorkspace)		
VariableName	Data	string {'simin'}
SampleTime	Sample time	string {'0'}
Interpolate	Interpolate data	{'on'}   'off'
OutputAfterFinalValue	Form output after final data value by	{'Extrapolation'}   'Setting to zero'   'Holding final value'   'Cyclic repetition'

### Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
Ground		
In1 (Inport)		
Port	Port number	string {'1'}
IconDisplay	Icon display	'Signal name'   {'Port number'}   'Port number and signal name'
UseBusObject	Specify properties via bus object	'on'   {'off'}
BusObject	Bus object for validating input bus	string {'BusObject'}
BusOutputAsStruct	Output as nonvirtual bus	'on'   {'off'}
PortDimensions	Port dimensions (-1 for inherited)	string {'-1'}
SampleTime	Sample time (-1 for inherited)	string {'-1'}
DataType	Data type	{'auto'}   'double'   'single'   'int8'   'uint8'   'int16'   'uint16'   'int32'   'uint32'   'boolean'   'Specify via dialog'
OutDataType	Output data type (e.g., sfix(16), uint(8), float('single'))	string {'sfix(16)'}
OutScaling	Output scaling value (Slope, e.g., 2 <sup>-9</sup> or [Slope Bias], e.g., [1.25 3])	string {'2^0'}
SignalType	Signal type	{'auto'}   'real'   'complex'
SamplingMode	Sampling mode	{'auto'}   'Sample based'   'Frame based'
LatchInput	Latch (buffer) input	'on'   {'off'}
Interpolate	Interpolate data	{'on'}   'off'

**Sources Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Pulse Generator (DiscretePulseGenerator)		
PulseType	Pulse type	{ 'Time based' }   'Sample based'
TimeSource	Time (t)	{ 'Use simulation time' }   'Use external signal'
Amplitude	Amplitude	string { '1' }
Period	Period	string { '2' }
PulseWidth	Pulse width	string { '50' }
PhaseDelay	Phase delay	string { '0' }
SampleTime	Sample time	string { '1' }
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'
Ramp (Ramp) (masked subsystem)		
slope	Slope	string { '1' }
start	Start time	string { '0' }
X0	Initial output	string { '0' }
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'
Random Number (RandomNumber)		
Mean	Mean	string { '0' }
Variance	Variance	string { '1' }
Seed	Initial seed	string { '0' }
SampleTime	Sample time	string { '0' }
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'
Repeating Sequence (Repeating table) (masked subsystem)		

### Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
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	{'Interpolation-Use End Values'}   'Use Input Nearest'   'Use Input Below'   'Use Input Above'
tsamp	Sample time	string {'0.01'}
OutputDataTypeScaling Mode	Output data type and scaling	{'Specify via dialog'}   'Inherit via back propagation'
OutDataType	Output data type: ex. sfix(16), uint(8), float('single')	string {'float('double')'}
OutScaling	Output scaling: Slope or [Slope Bias] ex. 2^-9	string {'2^-10'}
LockScale	Lock output scaling so autoscaling tool can't change it	'on'   {'off'}
Repeating Sequence Stair (Repeating Sequence Stair) (masked subsystem)		
OutValues	Vector of output values	string {'[3 1 4 2 1].''}
tsamp	Sample time	string {'-1'}
OutputDataTypeScaling Mode	Output data type and scaling	{'Specify via dialog'}   'Inherit via back propagation'
OutDataType	Output data type: ex. sfix(16), uint(8), float('single')	string {'float('double')'}
ConRadixGroup	Output scaling mode	'Use Specified Scaling'   {'Best Precision: Vector-wise'}



**Sources Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
OutScaling	Output scaling: Slope or [Slope Bias] ex. $2^{-9}$	string { ' $2^{-12}$ ' }
LockScale	Lock output scaling so autoscaling tool can't change it	'on'   {'off'}
Signal Builder (Sigbuilder block) (masked subsystem)		
Signal Generator (SignalGenerator)		
WaveForm	Wave form	{ 'sine' }   'square'   'sawtooth'   'random'
TimeSource	Time (t)	{ 'Use simulation time' }   'Use external signal'
Amplitude	Amplitude	string { '1' }
Frequency	Frequency	string { '1' }
Units	Units	'rad/sec'   {'Hertz'}
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'
Sine Wave (Sin)		
SineType	Sine type	{ 'Time based' }   'Sample based'
TimeSource	Time (t)	{ 'Use simulation time' }   'Use external signal'
Amplitude	Amplitude	string { '1' }
Bias	Bias	string { '0' }
Frequency	Frequency (rad/sec)	string { '1' }
Phase	Phase (rad)	string { '0' }
Samples	Samples per period	string { '10' }
Offset	Number of offset samples	string { '0' }
SampleTime	Sample time	string { '0' }

Sources Library Block Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'
Step (Step)		
Time	Step time	string { '1' }
Before	Initial value	string { '0' }
After	Final value	string { '1' }
SampleTime	Sample time	string { '0' }
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'
ZeroCross	Enable zero crossing detection	{ 'on' }   'off'
Uniform Random Number (UniformRandomNumber)		
Minimum	Minimum	string { '-1' }
Maximum	Maximum	string { '1' }
Seed	Initial seed	string { '0' }
SampleTime	Sample time	string { '0' }
VectorParams1D	Interpret vector parameters as 1-D	{ 'on' }   'off'

User-Defined Functions Library Block Parameters

Block (Type)/Parameter	Dialog Box Prompt	Values
Embedded MATLAB Fcn (Stateflow) (masked subsystem)		
Fcn (Fcn)		
Expr	Expression	string { 'sin(u(1)*exp(2.3*(-u(2))))' }

**User-Defined Functions Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
SampleTime	Sample time (-1 for inherited)	string {'-1'}
M-file S-Function (M-S-Function)		
MFile	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	{'auto'}   'real'   'complex'
Output1D	Collapse 2-D results to 1-D	{'on'}   'off'
SampleTime	Sample time (-1 for inherited)	string {'-1'}
S-Function (S-Function)		
FunctionName	S-function name	string {'system'}
Parameters	S-function parameters	string {''}
S-Function Builder (S-Function Builder) (masked subsystem)		
FunctionName	S-function name	string {'system'}
Parameters	S-function parameters	string {''}

**Additional Discrete Block Library Parameters**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
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]'}

### Additional Discrete Block Library Parameters (Continued)

Block (Type)/Parameter	Dialog Box Prompt	Values
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: ex. sfix(16), uint(8), float('single')	string { 'float('double')'
StateEqScaling	Scaling for State Equation AX+BU: ex. 2 <sup>-9</sup>	string { '2^0'
OutputEqScaling	Scaling for Output Equation CX+DU: ex. 2 <sup>-9</sup>	string { '2^0'
LockScale	Lock output scaling so autoscaling tool can't change it	'on'   {'off'}
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
DoSatur	Saturate to max or min when overflows occur	'on'   {'off'}
Transfer Fcn Direct Form II (Transfer Fcn Direct Form II) (masked subsystem)		
NumCoefVec	Numerator coefficients	string { '[0.2 0.3 0.2]'
DenCoefVec	Denominator coefficients excluding lead (which must be 1.0)	string { '[-0.9      0.6]'
vinit	Initial condition	string { '0.0'
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   {'Floor'}
DoSatur	Saturate to max or min when overflows occur	'on'   {'off'}

**Additional Discrete Block Library Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Transfer Fcn Direct Form II Time Varying (Transfer Fcn Direct Form II Time Varying) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
RndMeth	Round toward	'Zero'   'Nearest'   'Ceiling'   'Floor'
DoSatur	Saturate to max or min when overflows occur	'on'   {'off'}
Unit Delay Enabled (Unit Delay Enabled) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay Enabled External IC (Unit Delay Enabled External Initial Condition) (masked subsystem)		
tsamp	Sample time	string {'-1'}
Unit Delay Enabled Resettable (Unit Delay Enabled Resettable) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay Enabled Resettable External IC (Unit Delay Enabled Resettable External Initial Condition) (masked subsystem)		
tsamp	Sample time	string {'-1'}
Unit Delay External IC (Unit Delay External Initial Condition) (masked subsystem)		
tsamp	Sample time	string {'-1'}
Unit Delay Resettable (Unit Delay Resettable) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay Resettable External IC (Unit Delay Resettable External Initial Condition) (masked subsystem)		
tsamp	Sample time	string {'-1'}

**Additional Discrete Block Library Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
Unit Delay With Preview Enabled (Unit Delay With Preview Enabled) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay With Preview Enabled Resettable (Unit Delay With Preview Enabled Resettable) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay With Preview Enabled Resettable External RV (Unit Delay With Preview Enabled Resettable External RV) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay With Preview Resettable (Unit Delay With Preview Resettable) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}
Unit Delay With Preview Resettable External RV (Unit Delay With Preview Resettable External RV) (masked subsystem)		
vinit	Initial condition	string {'0.0'}
tsamp	Sample time	string {'-1'}

**Table 9-1: 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)		

**Table 9-1: Additional Math: Increment - Decrement Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
Decrement To Zero (Decrement To Zero) (masked subsystem)		
Increment Real World (Real World Value Increment) (masked subsystem)		
Increment Stored Integer (Stored Integer Value Increment) (masked subsystem)		

# Mask Parameters

This section lists parameters that describe masked blocks. This table lists masking parameters, which correspond to **Mask Editor** dialog box parameters (see “Setting Mask Parameters” on page 9-129).

## 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 <b>Dialog callback</b> field on the <b>Parameters</b> pane of the <b>Mask editor</b> dialog box.	pipe delimited string { ' ' }
MaskCallbacks	Cell array version of MaskCallbackString.	cell array { ' [ ] ' }
MaskDescription	Block description. Set by the <b>Mask description</b> field on the <b>Documentation</b> pane of the <b>Mask editor</b> dialog box.	string { ' ' }
MaskDisplay	Drawing commands for the block icon. Set by the <b>Drawing commands</b> field on the <b>Icon</b> pane of the <b>Mask editor</b> dialog box.	string { ' ' }
MaskEditorHandle	For internal use.	
MaskEnableString	Option that determines whether a parameter is greyed out in the dialog. Set by the <b>Enable parameter</b> check box on the <b>Parameters</b> pane of the <b>Mask editor</b> 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 <b>Mask help</b> field on the <b>Documentation</b> pane of the <b>Mask editor</b> dialog box.	string { ' ' }



**Mask Parameters (Continued)**

<b>Parameter</b>	<b>Description/Prompt</b>	<b>Values</b>
MaskIconFrame	Set the visibility of the icon frame (Visible is on, Invisible is off). Set by the <b>Frame</b> option on the <b>Icon</b> pane of the <b>Mask editor</b> dialog box.	{ 'on' }   'off'
MaskIconOpaque	Set the transparency of the icon (Opaque is on, Transparent is off). Set by the <b>Transparency</b> option on the <b>Icon</b> pane of the <b>Mask editor</b> dialog box.	{ 'on' }   'off'
MaskIconRotate	Set the rotation of the icon (Rotates is on, Fixed is off). Set by the <b>Rotation</b> option on the <b>Icon</b> pane of the <b>Mask editor</b> dialog box.	'on'   { 'off' }
MaskIconUnits	Set the units for the drawing commands. Set by the <b>Units</b> option on the <b>Icon</b> pane of the <b>Mask editor</b> dialog box.	'pixel'   { 'autoscale' }   'normalized'
MaskInitialization	Initialization commands. Set by the <b>Initialization commands</b> field on the <b>Initialization</b> pane of the <b>Mask editor</b> dialog box.	MATLAB command { '' }
MaskNames	Cell array of mask dialog parameter names. Set inside the <b>Variable</b> column in the <b>Parameters</b> pane of the <b>Mask editor</b> dialog box.	matrix { '[]' }
MaskPrompts	List of dialog parameter prompts (see below). Set inside the <b>Dialog parameters</b> area on the <b>Parameters</b> pane of the <b>Mask editor</b> dialog box.	cell array of strings { '[]' }
MaskPromptString	List of dialog parameter prompts (see below). Set inside the <b>Dialog parameters</b> area on the <b>Parameters</b> pane of the <b>Mask editor</b> dialog box.	string { '' }

### Mask Parameters (Continued)

Parameter	Description/Prompt	Values
MaskPropertyNameString	Pipe-delimited version of MaskNames.	string { '' }
MaskRunInitForIconRedraw	For internal use.	
MaskSelfModifiable	Indicates that the block can modify itself. Set by the <b>Allow library block to modify its contents</b> check box on the <b>Initialization</b> pane of the <b>Mask editor</b> dialog box.	'on'   {'off'}
MaskStyles	Determines whether the dialog parameter is a check box, edit field, or pop-up list. Set by the <b>Type</b> column in the <b>Parameters</b> pane of the <b>Mask editor</b> dialog box.	cell array { '[]' }
MaskStyleString	Comma-separated version of MaskStyles.	string { '' }
MaskTabNameString	For internal use.	
MaskTabNames	For internal use.	
MaskToolTipsDisplay	Determines which of the mask dialog parameters to display.	cell array of 'on' and 'off' { '' }
MaskToolTipString	Comma-delimited version of MaskToolTipsDisplay.	string { '' }
MaskTunableValues	Allows the changing of mask dialog values during simulation. Set by the <b>Tunable</b> column in the <b>Parameters</b> pane of the <b>Mask editor</b> dialog box.	cell array of strings { '[]' }
MaskTunableValueString	Delimited string version of MaskTunableValueString.	delimited string { '' }
MaskType	Mask type. Set by the <b>Mask type</b> field on the <b>Documentation</b> pane of the <b>Mask editor</b> dialog box.	string { 'Stateflow' }
MaskValues	Dialog parameter values.	cell array { '[]' }

**Mask Parameters (Continued)**

Parameter	Description/Prompt	Values
MaskValueString	Delimited string version of MaskValues.	delimited string { ' ' }
MaskVarAliases	For internal use.	
MaskVarAliasString	For internal use.	
MaskVariables	List of the dialog parameters' variables (see below). Set inside the <b>Dialog parameters</b> area on the <b>Parameters</b> pane of the <b>Mask editor</b> dialog box.	string { ' ' }
MaskVisibilities	Specifies visibility of parameters.	matrix { ' [ ] ' }
MaskVisibilityString	Cell array version of MaskVisibilities.	string { ' ' }
MaskWSVariables	List of the variables defined in the mask workspace (read only).	matrix { ' [ ] ' }

**Setting Mask Parameters**

When you use the **Mask Editor** to create a dialog box parameter for a masked block, you provide this information:

- The prompt, which you enter in the **Prompt** field
- The variable that holds the parameter value, which you enter in the **Variable** field
- The type of field created, which you specify by selecting a **Control type**
- Whether the value entered in the field is to be evaluated or stored as a literal, which you specify by selecting an **Assignment type**

**How Masked Parameters are Stored**

The mask parameters, listed in the preceding table, store the values specified for the dialog box parameters in these ways:

- The **Prompt** field values for all dialog box parameters are stored in the MaskPromptString parameter as a string, with individual values separated by a vertical bar (|), as shown in this example:

```
"Slope:|Intercept:"
```

- The **Variable** field values for all dialog box parameters are stored in the MaskVariables parameter as a string, with individual assignments separated by a semicolon. A sequence number indicates the prompt that is associated with a variable. A special character preceding the sequence number indicates the **Assignment** type: @ indicates **Evaluate**, & indicates **Literal**.

For example, "a=@1;b=&2;" indicates that the value entered in the first parameter field is assigned to variable a and is evaluated in MATLAB before assignment, and the value entered in the second field is assigned to variable b and is stored as a literal, which means that its value is the string entered in the dialog box.

- The **Control type** field values for all dialog box parameters are stored in the MaskStyleString parameter as a string, with individual values separated by a comma. The **Popup strings** values appear after the popup type, as shown in this example:

```
"edit,checkbox,popup(red|blue|green) "
```

- The parameter values are stored in the MaskValueString mask parameter as a string, with individual values separated by a vertical bar. The order of the values is the same as the order in which the parameters appear on the dialog box. For example, these statements define values for the parameter field prompts and the values for those parameters:

```
MaskPromptString    "Slope:|Intercept:"
MaskValueString     "2|5"
```

# Model File Format

---

This section describes the format of a Simulink model file.

## Model File Contents

A model file is a structured ASCII file that contains keywords and parameter-value pairs that describe the model. The file describes model components in hierarchical order.

The structure of the model file is as follows.

```
Model {
  <Model Parameter Name> <Model Parameter Value>
  ...
  BlockDefaults {
    <Block Parameter Name> <Block Parameter Value>
    ...
  }
  AnnotationDefaults {
    <Annotation Parameter Name> <Annotation Parameter Value>
    ...
  }
  System {
    <System Parameter Name> <System Parameter Value>
    ...
    Block {
      <Block Parameter Name> <Block Parameter Value>
      ...
    }
    Line {
      <Line Parameter Name> <Line Parameter Value>
      ...
      Branch {
        <Branch Parameter Name> <Branch Parameter Value>
        ...
      }
    }
    Annotation {
      <Annotation Parameter Name> <Annotation Parameter Value>
      ...
    }
  }
}
```

The model file consists of sections that describe different model components:

- The `Model` section defines model parameters.
- The `BlockDefaults` section contains default settings for blocks in the model.
- The `AnnotationDefaults` section contains default settings for annotations in the model.
- The `System` section contains parameters that describe each system (including the top-level system and each subsystem) in the model. Each `System` section contains block, line, and annotation descriptions.

See Chapter 9, “Model and Block Parameters” for descriptions of model and block parameters.

## Model Section

The `Model` section, located at the top of the model file, defines the values for model-level parameters. These parameters include the model name, the version of Simulink last used to modify the model, and simulation parameters.

## BlockDefaults Section

The `BlockDefaults` section appears after the simulation parameters and defines the default values for block parameters within this model. These values can be overridden by individual block parameters, defined in the `Block` sections.

## AnnotationDefaults Section

The `AnnotationDefaults` section appears after the `BlockDefaults` section. This section defines the default parameters for all annotations in the model. These parameter values cannot be modified using the `set_param` command.

## System Section

The top-level system and each subsystem in the model are described in a separate `System` section. Each `System` section defines system-level parameters and includes `Block`, `Line`, and `Annotation` sections for each block, line, and annotation in the system. Each `Line` that contains a branch point includes a `Branch` section that defines the branch line.





# Embedded MATLAB

## Function Block Reference

---

An Embedded MATLAB Function block lets you compose a MATLAB language function in a Simulink model that generates embeddable code. In an Embedded MATLAB Function block, you create functions with a rich subset of the MATLAB language. Later, when you simulate the model or generate code for a target environment, a function in an Embedded MATLAB Function block generates efficient C code. This code meets the strict memory and data type requirements of embedded target environments. In this way, Embedded MATLAB Function blocks bring the power of MATLAB for the embedded environment into Simulink.

The following sections describe the supported MATLAB features in Embedded MATLAB functions:

Supported Variable Types in  
Embedded MATLAB Functions  
(p. 11-2)

Data types supported by Embedded MATLAB Function  
blocks.

Operators in Embedded MATLAB  
Functions (p. 11-3)

Operators supported by Embedded MATLAB Function  
blocks.

Embedded MATLAB Run-Time  
Function Library (p. 11-7)

Lists of run-time library functions that you can call in an  
Embedded MATLAB function.

Unsupported MATLAB Features and  
Limitations (p. 11-23)

Features not supported by Embedded MATLAB Function  
blocks.

# Supported Variable Types in Embedded MATLAB Functions

Embedded MATLAB functions support a subset of MATLAB data types for input (argument) and output (return) variables, and variables local to the function. This section lists and describes the data types supported in Embedded MATLAB functions, along with any exceptions or deviations from MATLAB behavior.

Embedded MATLAB functions support scalar, vector, and matrix (two-dimensional) variables of the types represented by the following cast functions:

Type/Function	Description
char	Character array (string)
complex	Complex data. Cast function takes real and imaginary components. See “Using Complex Data in Embedded MATLAB Functions” in Simulink online documentation.
double	Double-precision floating point
int8, int16, int32	Signed integer
logical	Boolean true or false
single	Single-precision floating point
uint8, uint16, uint32	Unsigned integer

## Operators in Embedded MATLAB Functions

Embedded MATLAB functions support a large subset of MATLAB operators for data operands of type `double` and `single`. Allowed MATLAB operators in Embedded MATLAB functions are listed in the following topics:

- “Control Flow Statements in Embedded MATLAB Functions” on page 11-3 — Common programming statements supported in Embedded MATLAB functions
- “Arithmetic Operators in Embedded MATLAB Functions” on page 11-4 — Operators for arithmetic operations supported in Embedded MATLAB functions.
- “Relational Operators in Embedded MATLAB Functions” on page 11-5 — Operators for relational operations supported in Embedded MATLAB functions.
- “Logical Operators in Embedded MATLAB Functions” on page 11-5 — Lists Operators for logical operations supported in Embedded MATLAB functions.

Each listing includes a link to help for the equivalent MATLAB function along with a one line description and any limitations that apply.

### Control Flow Statements in Embedded MATLAB Functions

Embedded MATLAB functions support the following MATLAB program statements:

Statement	Description
<code>break</code>	break statement
<code>continue</code>	continue statement
<code>for</code>	for statement

Statement	Description
if	if statement  The conditions of an if statement cannot use & and   operators. In their place, use the && and    operators, respectively. To logically collapse vectors into scalars, use the function all.
return	return statement
switch	switch statement
while	while statement  The conditions of while statements cannot use & and   operators. In their place, use the && and    operators, respectively. To logically collapse vectors into scalars, use the function all.

## Arithmetic Operators in Embedded MATLAB Functions

Embedded MATLAB functions support the following MATLAB arithmetic operations for matrix operands (maximum two-dimensional) of type double or single. See “Arithmetic Operators + - \* / \ ^ ” in MATLAB online help for detailed descriptions of each operation and any requirements for the data types of the operands.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
.*	Elementwise multiplication
./	Elementwise division
.^	Elementwise raising to a scalar power

Operator	Description
[ ]	Concatenation of matrices
'	Complex conjugate transpose
.'	Transpose
(r, c)	Matrix indexing, where r and c are vectors of row and column indices, respectively

### Relational Operators in Embedded MATLAB Functions

Embedded MATLAB functions support the following elementwise relational operators between matrix operands (maximum two-dimensional) of type double or single of the same size. See “Relational Operators < > <= >= == ~=” in MATLAB online help for detailed descriptions of each operation and any requirements for the data types of the operands.

Operation	Description
<	Less than
<=	Less than or equal to
>=	Greater than or equal to
>	Greater than
==	Equal
~=	Not equal

### Logical Operators in Embedded MATLAB Functions

Embedded MATLAB functions support the following elementwise logical operators between matrix operands (maximum two-dimensional) of type double or single of the same size. See “Logical Operators, Element-wise & | ~” and “Logical Operators, Short-circuit && ||” in MATLAB online help for

detailed descriptions of each operation and any requirements for the data types of the operands.

Operation	Description
&	Logical AND  This & operator is limited to use outside if and while statement conditions. In its place, use the && operator. To logically collapse vectors into scalars, use the Embedded MATLAB function all.
	Logical OR  This   operator is limited to use outside if and while statements. In its place, use the    operator. To logically collapse vectors into scalars use, the Embedded MATLAB function all.
-	Element complement
xor	Logical XOR
&&	Logical AND (short-circuiting)
	Logical OR (short-circuiting)

## Embedded MATLAB Run-Time Function Library

This section lists the callable functions of the Embedded MATLAB run-time function library. Functions in the Embedded MATLAB run-time library have the same name, same arguments (with limitations), and the same functionality (with limitations) as their MATLAB counterparts, but are written in C to produce efficient embeddable generated code. If you restrict your function calls in Embedded MATLAB functions to these functions, you can use the code generated for an Embedded MATLAB function to build a stand-alone executable target that you can move to different platforms for execution.

The following topics list and describe the functions supported by the Embedded MATLAB run-time library:

- “Embedded MATLAB Run-Time Function Library — Alphabetical List” on page 11-7
- “Embedded MATLAB Run-Time Library — Categorical List” on page 11-13

### Embedded MATLAB Run-Time Function Library — Alphabetical List

This topic lists functions in the Embedded MATLAB run-time library in alphabetical order. Each entry includes a link to the help for the equivalent MATLAB function along with a one-line description and any limitations that apply to it. See also “Embedded MATLAB Run-Time Library — Categorical List” on page 11-13.

Function	Description
abs	Absolute value and complex magnitude of an array
acos	Inverse cosine
acosh	Inverse hyperbolic cosine
all	Test if all elements are nonzero
and	Logical AND (&)
any	Test for any nonzero elements

Function	Description
asin	Inverse sine
asinh	Inverse hyperbolic sine
atan	Inverse tangent
atan2	Four quadrant inverse tangent
atanh	Inverse hyperbolic tangent
ceil	Round toward plus infinity
char	Create character array (string)
chol	Cholesky factorization
class	Query class of object
complex	Construct complex data from real and imaginary components; see “Using Complex Data in Embedded MATLAB Functions” in Simulink online documentation
conj	Return the conjugate of a complex number
conv	Convolution and polynomial multiplication (Signal Processing Toolbox license required)
cos	Cosine
cosh	Hyperbolic cosine
ctranspose	Complex conjugate transpose (')
cumprod	Cumulative product of array elements
cumsum	Cumulative sum of array elements
diag	Return a matrix formed around the specified diagonal vector and the specified diagonal (0, 1, 2,...) it occupies
double	Convert to double-precision floating point
eps	Return floating-point relative accuracy



Function	Description
eq	Equal (==)
exp	Exponential
eye	Identity matrix
false	Return an array of 0's for the specified dimensions
fft	Discrete Fourier transform (Signal Processing Toolbox license required)
fix	Round toward zero
filter	Filter a data sequence using a digital filter that works for both real and complex inputs (Signal Processing Toolbox license required)
floor	Round toward minus infinity
ge	Greater than or equal to (>=)
gt	Greater than (>)
ifft	Inverse discrete Fourier transform (Signal Processing Toolbox license required)
imag	Return the imaginary part of a complex number
Inf or inf	Return IEEE arithmetic representation for positive infinity (simulation only)
interp1	One-dimensional interpolation (table lookup)
inv	Inverse of a square matrix
int8, int16, int32	Convert to signed integer data type
ischar	True for character array (string)
isempty	Test if an array is empty

Function	Description
isfinite	Detect finite elements of an array
isinf	Detect infinite elements of an array (simulation only)
isnan	Detect NaN elements of an array (simulation only)
isnumeric	True for numeric arrays
isreal	Return false (0) for a complex number
isscalar	True if array is a scalar
isvector	True if array is a vector
ldivide	Left array divide (\)
le	Less than or equal to (<=)
length	Return the length of a matrix
log	Natural logarithm
logical	Data type representing true or false value
lt	Less than (<)
lu	Matrix factorization
max	Maximum elements of a matrix
min	Minimum elements of a matrix
minus	Minus (-)
mldivide	Left matrix divide (\)
mean	Average or mean value
mod	Modulus (signed remainder after division)
mpower	Matrix power (^)
mrdivide	Right matrix divide (/)
mtimes	Matrix multiply (*)

<b>Function</b>	<b>Description</b>
NaN or nan	Return not a number (simulation only)
nargin	Return the number of input arguments a user has supplied
nargout	Return the number of output return values a user has requested
ndims	Number of dimensions
ne	Not equal (~=)
not	Logical NOT (~)
ones	Create a matrix of all ones
or	Logical OR ( )
pi	Return the ratio of the circumference to the diameter for a circle
pinv	Pseudoinverse of a matrix
plus	Plus (+)
power	Array power (.^)
prod	Product of array element
rdivide	Right array divide (/)
real	Return the real part of a complex number
realmax	Return the largest positive floating-point number
realmin	Return the smallest positive floating-point number
rem	Remainder after division
repmat	Replicate and tile an array
reshape	Reshape one array into the dimensions of another
round	Round toward nearest integer

Function	Description
sign	Signum function
sin	Sine
single	Convert to single precision
sinh	Hyperbolic sine
size	Return the size of a matrix
sort	Sort elements in ascending or descending order
sosfilt	Second order (biquadratic) IIR filtering (Signal Processing Toolbox license required)
sqrt	Square root
strcmp	Return a logical result for the comparison of two strings; limited to strings known at compile time
sum	Sum of matrix elements
svd	Singular value decomposition
tan	Tangent
tanh	Hyperbolic tangent
times	Array multiply (.*)
tril	Extract lower triangular part
triu	Extract upper triangular part
transpose	Matrix transpose (')
true	Return an array of logical (Boolean) 1's for the specified dimensions
uint8, uint16, uint32	Convert unsigned integer data type
uminus	Unary minus (-)

Function	Description
uplus	Unary plus (+)
xcorr	Cross-correlation function estimates (Signal Processing Toolbox license required)
zeros	Create a matrix of all zeros

### Embedded MATLAB Run-Time Library – Categorical List

The following topics list functions in the Embedded MATLAB run-time library by different function types. Each entry includes a function name link to online help for the equivalent MATLAB function along with a one-line description and any limitations that apply to it.

- “Arithmetic Operator Functions” on page 11-14
- “Casting Functions” on page 11-14
- “Complex Number Functions” on page 11-15
- “Exponential Functions” on page 11-16
- “Input and Output Functions” on page 11-16
- “Logical Operator Functions” on page 11-16
- “Matrix/Array Functions” on page 11-17
- “Relational Operator Functions” on page 11-18
- “Rounding and Remainder Functions” on page 11-19
- “Signal Processing Functions” on page 11-19
- “Special Values” on page 11-20
- “String Functions” on page 11-21
- “Trigonometric Functions” on page 11-21

See also “Embedded MATLAB Run-Time Function Library — Alphabetical List” on page 11-7.

Arithmetic Operator Functions

See “Arithmetic Operators + - \* / \ ^ ’” in MATLAB online help for detailed descriptions on any of the following operator equivalent functions.

Function	Description
ctranspose	Complex conjugate transpose ( ' )
ldivide	Arraywise left division
minus	Minus ( - )
mldivide	Left matrix divide ( \ )
mpower	Equivalent of elementwise power operator ( . ^ )
mrdivide	Matrix right division
mtimes	Matrix multiply ( * )
plus	Plus ( + )
power	Arraywise power
rdivide	Arraywise right division
times	Arraywise multiplication
transpose	Matrix transpose ( ' )
uminus	Unary minus ( - )
uplus	Unary plus ( + )

Casting Functions

Embedded MATLAB functions support the following functions for converting one type of data to another:

Data Type	Description
char	Create character array (string)
class	Query class of object argument

<b>Data Type</b>	<b>Description</b>
double	Convert to double-precision floating point
int8, int16, int32	Convert to signed integer data type
logical	Convert to Boolean true or false data type
single	Convert to single-precision floating point
uint8, uint16, uint32	Convert to unsigned integer data type

### Complex Number Functions

Embedded MATLAB functions support the following functions for complex numbers:

<b>Function</b>	<b>Description</b>
complex	Construct complex data from real and imaginary components; see “Declaring Complex Variables Implicitly in Embedded MATLAB Functions” in Simulink online documentation
conj	Return the conjugate of a complex number
imag	Return the imaginary part of a complex number
isnumeric	True for numeric arrays
isreal	Return false (0) for a complex number
isscalar	True if array is a scalar
real	Return the real part of a complex number

**Exponential Functions**

Embedded MATLAB functions support the following elementary exponential functions:

Function	Description
exp	Exponential
log	Natural logarithm
sqrt	Square root

**Input and Output Functions**

Embedded MATLAB functions support the following functions for accessing argument and return values:

Function	Description
nargin	Return the number of input arguments a user has supplied
nargout	Return the number of output return values a user has requested

**Logical Operator Functions**

Embedded MATLAB functions support the following functions for performing logical operations:

Function	Description
and	Logical AND (&)
not	Logical NOT (~)
or	Logical OR ( )



## Matrix/Array Functions

Embedded MATLAB functions support the following functions:

Function	Description
abs	Return absolute value and complex magnitude of an array
all	Test if all elements are nonzero
any	Test for any nonzero elements
cumprod	Cumulative product of array elements
cumsum	Cumulative sum of array elements
diag	Return a matrix formed around the specified diagonal vector and the specified diagonal (0, 1, 2,...) it occupies
eye	Identity matrix
false	Return an array of 0's for the specified dimensions
interp1	One-dimensional interpolation (table lookup)
inv	Inverse of a square matrix
isempty	Test if an array is empty
isfinite	Detect finite elements of an array
isinf	Detect infinite elements of an array (simulation only)
isnan	Detect NaN elements of an array (simulation only)
length	Return the length of a matrix
lu	Matrix factorization
max	Maximum elements of a matrix
min	Minimum elements of a matrix
ndims	Number of dimensions
ones	Create a matrix of all ones

Function	Description
pinv	Pseudoinverse of a matrix
prod	Product of array element
repmat	Replicate and tile an array
reshape	Reshape one array into the dimensions of another
sign	Signum function
size	Return the size of a matrix
sort	Sort elements in ascending or descending order.
sum	Sum of matrix elements
tril	Extract lower triangular part
triu	Extract upper triangular part
true	Return an array of logical (Boolean) 1's for the specified dimensions
zeros	Create a matrix of all zeros

**Relational Operator Functions**

Embedded MATLAB functions support the following functions for performing relational operations:

Function	Description
eq	Equal (==)
ge	Greater than or equal to (>=)
gt	Greater than (>)
le	Less than or equal to (<=)
lt	Less than (<)
ne	Not equal (~=)

## Rounding and Remainder Functions

Embedded MATLAB functions support the following rounding and remainder functions:

Function	Description
<code>ceil</code>	Round toward plus infinity
<code>fix</code>	Round toward zero
<code>floor</code>	Round toward minus infinity
<code>mean</code>	Average or mean value
<code>mod</code>	Modulus (signed remainder after division)
<code>rem</code>	Remainder after division
<code>round</code>	Round toward nearest integer

## Signal Processing Functions

Embedded MATLAB supports the following signal processing functions:

Function	Description
<code>chol</code>	Cholesky factorization
<code>conv</code>	Convolution and polynomial multiplication (license required — see note below)
<code>fft</code>	Discrete Fourier transform (license required — see note below)
<code>filter</code>	Filter a data sequence using a digital filter that works for both real and complex inputs (license required — see note below)
<code>ifft</code>	Inverse discrete Fourier transform (license required — see note below)

Function	Description
sosfilt	Second order (biquadratic) IIR filtering (license required — see note below)
svd	Singular value decomposition
xcorr	Cross-correlation function estimates (license required — see note below)

**Note** Some signal processing functions for the Embedded MATLAB Run-Time Function Library require a Signal Processing Toolbox license. These functions also have file dependencies on static functions supplied for some of the Signal Processing blocks. For code deployment to another environment, be sure to consult the generated make file for the RTW target you build to identify and locate these file dependencies.

Special Values

Embedded MATLAB functions support the following special data values:

Symbol	Description
eps	Return floating-point relative accuracy
Inf or inf	Return IEEE arithmetic representation for positive infinity (simulation only)
NaN or nan	Return not a number (simulation only)
pi	Return the ratio of the circumference to the diameter for a circle
realmax	Return the largest positive floating-point number
realmin	Return the smallest positive floating-point number

## String Functions

Embedded MATLAB functions support the following functions for handling strings:

Function	Description
char	Create character array (string)
ischar	True for character array (string)
strcmp	Return a logical result for the comparison of two strings; limited to strings known at compile time

## Trigonometric Functions

Embedded MATLAB functions support the following elementary trigonometric functions:

Function	Description
acos	Inverse cosine
acosh	Inverse hyperbolic cosine
asin	Inverse sine
asinh	Inverse hyperbolic sine
atan	Inverse tangent
atan2	Four quadrant inverse tangent
atanh	Inverse hyperbolic tangent
cos	Cosine
cosh	Hyperbolic cosine
sin	Sine
sinh	Hyperbolic sine

Function	Description
tan	Tangent
tanh	Hyperbolic tangent

# Unsupported MATLAB Features and Limitations

Embedded MATLAB Function blocks provide a subset of the full list of MATLAB features. Unsupported features and features with limitations are described in the following topics:

- “List of Unsupported Features” on page 11-23
- “Limitations on Indexing Operations” on page 11-24
- “Limitations with Complex Numbers” on page 11-25

## List of Unsupported Features

The following is a list of MATLAB features that are not supported by Embedded MATLAB functions.

Feature Not Supported	Remarks
Cell arrays	Supported data types are listed in “Supported Variable Types in Embedded MATLAB Functions” on page 11-2.
Command-line duality	
Dynamic variables	You cannot use variables of dynamic size, or variables of different sizes.
Function handles	
Global	
Integer math	
Java	
M-files	User M-files in MATLAB path are not supported for code generation, but they can be called during simulation.
Matrix deletion	

Feature Not Supported	Remarks
N-dimensional matrices	Supported sizes are scalar and two-dimensional matrices. Vectors are two-dimensional matrices with a row or column dimension of 1.
Nested functions	
Objects	
Sparse matrices	
Structures	
Try/catch	

## Limitations on Indexing Operations

Matrix indexing operations for a matrix **M** are supported with limitations for the following types of expressions:

- **M(i:j)** where **i** and **j** change in a loop

Embedded MATLAB never dynamically allocates memory for the size of the expressions that change as the program executes. The workaround is to use for loops as shown in the following example:

```
for i=1:10
    for j = i:10
        M(i:j) = 2 * M(i:j);
    end
end
```

- **M(i:i+k)** where **i** is unknown but **k** is known

In this case, since **i** and therefore **i+k** are not known, memory cannot be allocated for the numerical result. However, memory can be allocated for the following workaround:

```
M(i + (0:k))
```

In this case, an unknown scalar value **i** is added to each element of the known index vector **0...k**. This means that memory for **k+1** elements of **M** is allocated.



- Initialization of the following style:

```
for i = 1:10
    M(i) = 5;
end
```

In this case, the size of M changes as the loop is executed.

## Limitations with Complex Numbers

Complex numbers and operations are supported in Embedded MATLAB Function blocks with the following exceptions:

- The first use of a variable that is later assigned a complex result must also be complex. For example,

```
X = 3;
.
.
.
X = 4 + 5i;
```

fails because X is not defined as a complex variable by its first assignment. However,

```
X = 3 + 0i;
.
.
.
X = 4 + 5i;
```

succeeds because X is defined as a complex variable in its first assignment.

- Even if the imaginary part is zero, if the result might be complex, Embedded MATLAB will treat it as complex. For example, although

```
X = ifft(fft(Y));
```

yields a real answer, Embedded MATLAB assumes that the function `ifft` might return a complex result. The workaround is to use the `real` function:

```
X = real(ifft(fft(Y)));
```

- Inheritance of variable type from a complex-valued Simulink signal does not make the variable a complex number. Input (argument) variables are

assumed to be noncomplex unless they are specified complex in the Model Explorer.

## A

- Abs block 1-3
- absolute tolerance
  - simset parameter 4-12
  - specifying for a block state 1-271
- absolute value
  - generating 1-3
- Accumulator Resettable block 1-4
- Accumulator Resettable Limited block 1-4
- Action Port block 1-5
- Action subsystems
  - creating 1-5
  - with If block 1-246
  - with SwitchCase block 1-479
- add\_block command 3-7
- add\_line command 3-8
- add\_param command 3-10
- Additional Discrete block library
  - block parameters 9-121
- Additional Math: Increment - Decrement block
  - library
    - block parameters 9-124
- addterms command 3-11
- Algebraic Constraint block 1-9
- algebraic equations
  - modeling 1-9
- algebraic loops
  - integrator block reset or IC port 1-176
- analysis functions
  - perturbing model 1-256
- animate 7-7
- AnnotationDefaults section of mdl file 10-3
- annotations
  - annotation block
    - See* Model Info block
- ashow debug command 7-8
- Assert block 1-11

- Assignment block 1-14
- Atomic Subsystem block 1-465
- atrace debug command 7-9
- automatic scaling 7-22
  - and Look-Up Table (2D) block 1-295
  - autoscale safety margin 7-25
  - fixptbestprec 7-5
- autoscaling
  - fixptbestprec 7-5
- autoscaling Scope axes 1-417

## B

- Backlash block 1-21
- Backward Euler method 1-174
- Backward Rectangular method 1-174
- Band-Limited White Noise block 1-25
- bdclose command 3-12
- bdroot command 3-13
- bits
  - clear 1-33
  - mask 1-33
  - set 1-33
- block dialog boxes
  - closing 3-14
  - opening 3-33
- block names 5-3
  - newline character in 3-4
  - slash character in 3-5
- block parameters
  - Additional Discrete library 9-121
  - Additional Math: Increment - Decrement
    - library 9-124
  - changing during simulation 3-37
  - common 9-30
  - Continuous library 9-37

- Discontinuities library 9-39
- Discrete library 9-42
- Logic and Bit Operations library 9-51
- Lookup Tables library 9-55
- Math library 9-60
- Model Verification block library 9-74
- Model-Wide Utilities library 9-78
- Ports & Subsystems library 9-80
- Signal Attributes library 9-99
- Signal Routing library 9-104
- Sinks library 9-109
- Sources library 9-114
- User-defined functions library 9-120
- BlockDefaults section of mdl file 10-3
- blocks
  - Accumulator Resettable 1-4
  - Accumulator Resettable Limited 1-4
  - adding to model 3-7
  - Compare To Zero 1-81
  - Counter Limited 1-100
  - current 3-25
  - Data Type Propagation 1-122
  - Decrement Stored Integer 1-138
  - Decrement Time To Zero 1-139
  - Decrement To Zero 1-140
  - deleting
    - delete\_block command 3-16
  - Detect Decrease 1-150
  - Detect Fall Negative 1-151
  - Detect Fall Nonpositive 1-152
  - Detect Increase 1-153
  - Detect Rise Nonnegative 1-154
  - Detect Rise Positive 1-155
  - Filter Direct Form II 1-500
  - Filter Direct Form II Time Varying 1-502
  - Filter First Order 1-504
  - Filter Lead or Lag 1-506
  - Filter Real Zero 1-508
  - handle of current 3-26
  - Increment Stored Integer 1-253
  - Index Vector 1-254
  - Interval Test Dynamic 1-280
  - Product of Elements Inverted 1-364
  - Repeating Sequence Stair 1-401
  - Sample Time Divide 1-562
  - Sample Time Multiply 1-563
  - Sample Time Probe 1-564
  - Sample Time Subtract 1-564
    - specifying path 3-4, 5-3
  - Sum of Elements Negated 1-475
  - Unit Delay Enabled External IC 1-529
  - Unit Delay Enabled Resettable 1-531
  - Unit Delay Enabled Resettable External IC 1-533
  - Unit Delay External IC 1-535
  - Unit Delay Resettable 1-537
  - Unit Delay Resettable External IC 1-539
  - Unit Delay With Preview Enabled 1-541
  - Unit Delay With Preview Enabled Resettable 1-543
  - Unit Delay With Preview Enabled Resettable External RV 1-546
  - Unit Delay With Preview Resettable 1-549
  - Unit Delay With Preview Resettable External RV 1-551
  - See also* block parameters 9-37
- code function 2-5
- Boolean expressions
  - modeling 1-75
- break debug command 7-12
- bshow debug command 7-14
- Bus Creator block 1-36
- Bus Selector block 1-40

**C**

- capping unconnected blocks
  - using the Terminator block 1-486
- character encoding, model 3-42
- Check Discrete Gradient block 1-42
- Check Dynamic Gap block 1-45
- Check Dynamic Lower Bound block 1-48
- Check Dynamic Range block 1-51
- Check Dynamic Upper Bound block 1-54
- Check Input Resolution block 1-57
- Check Static Gap block 1-60
- Check Static Lower Bound block 1-63
- Check Static Range block 1-66
- Check Static Upper Bound block 1-69
- Chirp Signal block 1-72
- clear debug command 7-15
- clearing bits 1-33
- Clock block 1-74
- close\_system command 3-14
- clutch demo 1-242
- code generation
  - scaling 1-117
- Combinatorial Logic block 1-75
- combining input lines into vector line 1-344
- Compare To Zero block 1-81
- Complex to Magnitude-Angle block 1-83
- Complex to Real-Imag block 1-85
- concatenating matrices 1-321
- Configurable Subsystem block 1-87
- Constant block 1-91
- constant value
  - generating 1-91
- continue debug command 7-16
- Continuous block library
  - block parameters 9-37
- control flow diagrams
  - Action subsystem 1-5

- do-while

- While Iterator block 1-565

- for

- For Iterator block 1-215

- if-else

- If block 1-246

- switch

- Switch Case block 1-479

- while

- While Iterator block 1-565

- control flow statements in eML blocks 11-3

- Coulomb and Viscous Friction block 1-96

- Coulomb friction 1-96

- Counter Limited block 1-100

- Create Subsystem menu item 1-465

- current block

- getting pathname 3-25

- handle 3-26

- current system

- getting pathname 3-27

**D**

- Data Store Memory block 1-102

- Data Store Read block 1-106

- Data Store Write block 1-109

- Data Type Conversion block 1-112

- Data Type Propagation block 1-122

- data types

- propagation 1-122

- Dead Zone block 1-133

- deadband 1-21

- debug commands

- ashow 7-8

- atrace 7-9

- break 7-12

- bshow 7-14

- clear 7-15
- continue 7-16
- disp 7-17
- emode 7-22
- etrace 7-23
- help 7-24
- ishow 7-39
- minor 7-25
- nanbreak 7-26
- next 7-27
- probe 7-28
- quit 7-29
- run 7-31
- states 7-34
- status 7-35
- step 7-36
- stop 7-38
- systems 7-40
- tbreak 7-41
- trace 7-42
- undisp 7-43
- untrace 7-44
- xbreak 7-47
- zcbreak 7-48
- zclist 7-49
- decimation factor 4-13
- decision tables
  - modeling 1-75
- Decrement Stored Integer block 1-138
- Decrement Time To Zero block 1-139
- Decrement To Zero block 1-140
- delaying input by variable amount 1-554
- delete\_block command 3-16
- delete\_line command 3-17
- delete\_param command 3-18
- demos
  - clutch 1-242
  - fohdemo 1-210
  - hardstop 1-242
  - lorenzs 1-573
- Demux block 1-141
- Derivative block 1-147
  - accuracy of 1-147
- derivatives
  - calculating 1-147
  - limiting 1-379
- Detect Decrease block 1-150
- Detect Fall Negative block 1-151
- Detect Fall Nonpositive block 1-152
- Detect Increase block 1-153
- Detect Rise Nonnegative block 1-154
- Detect Rise Positive block 1-155
- differential/algebraic systems
  - modeling 1-9
- Digital Clock block 1-158
- Discontinuities block library
  - block parameters 9-39
- Discrete block library
  - block parameters 9-42
- Discrete Filter block 1-168
- Discrete State-Space block 1-170
- discrete state-space model 2-5
- Discrete Transfer Fcn block 1-183
- Discrete Zero-Pole block 1-185
- Discrete-Time Integrator block 1-173
- discrete-time systems
  - linearization 2-4
- disp command 5-6
- disp debug command 7-17
- Display block 1-187
  - as floating display 1-188
- displaying
  - signals graphically 1-414
- dlinmod function 2-3

DocBlock block 1-191  
Dot Product block 1-192  
dpoly command 5-7

## E

eigenvalues of linearized matrix 2-5  
Embedded MATLAB Function  
    signal processing functions 11-19  
eML blocks  
    arithmetic operators 11-4  
    control flow statements 11-3  
    description 11-1  
    eML run-time library 11-7  
    logical operators 11-5  
    MATLAB features NOT supported 11-23  
    operators 11-3  
    relational operators 11-5  
    variable types 11-2  
eML run-time library 11-7  
    alphabetical list of functions 11-7  
    casting functions 11-14  
    categorized list of functions 11-13  
    complex number functions 11-15  
    exponential functions 11-16  
    input and output functions 11-16  
    logical operator functions 11-16  
    matrix/array functions 11-17  
    relational operator functions 11-18  
    rounding and remainder functions 11-19  
    special value functions 11-20  
    string functions 11-21  
    trigonometric functions 11-21  
emode debug command 7-22  
Enable block 1-199  
Enabled and Triggered Subsystem block 1-201  
Enabled Subsystem block 1-202

enabled subsystems  
    Enable block 1-199  
etrace debug command 7-23  
expressions  
    applying to block inputs 1-207  
        MATLAB Fcn block 1-318  
external inputs  
    flag 4-5  
    from workspace 1-256  
    ut 4-8

## F

Fcn block 1-207  
    compared to Math Function block 1-313  
    compared to Rounding Function block 1-408  
    compared to Trigonometric Function block 1-520  
files  
    reading from 1-222  
    writing to  
        To File block 1-490  
Filter Direct Form II block 1-500  
Filter Direct Form II Time Varying block 1-502  
Filter First Order block 1-504  
Filter Lead or Lag block 1-506  
Filter Real Zero block 1-508  
find\_system command 3-20  
finding objects 3-20  
Finite Impulse Response filter 1-168  
finite-state machines  
    implementing 1-75  
First-Order Hold block 1-210  
    compared to Zero-Order Hold block 1-210  
fixdt function 7-3  
fixed step size 4-13  
Fixed-Point Interface Tool 7-22

- fixpt\_interp1 function 7-9
- fixpt\_look1\_func\_approx function 7-10
- fixpt\_look1\_func\_plot function 7-18
- fixpt\_set\_all function 7-20
- fixptbestexp function 7-4
- fixptbestprec function 7-5
  - autoscaling 7-5
- flip-flops
  - implementing 1-75
- float function 7-21
- floating scope
  - definition 1-422
- Floating Scope block 1-414
- fohdemo demo 1-210
- for control flow diagram
  - creating 1-215
- For Iterator block 1-215
- For Iterator Subsystem block 1-219
- For subsystems
  - creating 1-215
- Forward Euler method 1-173
- Forward Rectangular method 1-173
- fprintf command 5-9
- From block 1-220
- From File block 1-222
- From Workspace block 1-225
- Function-Call Generator block 1-229
- Function-Call Subsystem block 1-231
- functions
  - fixdt 7-3
  - fixpt\_interp1 7-9
  - fixpt\_look1\_func\_approx 7-10
  - fixpt\_look1\_func\_plot 7-18
  - fixpt\_set\_all 7-20
  - fixptbestexp 7-4
  - fixptbestprec 7-5
  - float 7-21

- fxptdlg 7-22
- num2fixpt 7-28
- sfix 7-29
- sfrac 7-30
- sint 7-31
- ufix 7-32
- ufrac 7-33
- uint 7-34
- fxptdlg function 7-22

## G

- gain
  - varying during simulation 1-457
- gcb command 3-25
- gcbh command 3-26
- gcs command 3-27
- get\_param command 3-28
- global Goto tag visibility 1-237
- Goto block 1-237
- Goto Tag Visibility block 1-240
- graphics
  - displaying on mask icon 5-12
- Greek letters
  - displaying on mask icons 5-6
  - using the text function 5-14
- Ground block 1-241
- GUI
  - See also* Fixed-Point Interface Tool

## H

- handle of current block 3-26
- hardstop demo 1-242
- help debug command 7-24
- Hide Name menu item
  - suppressing display of port label 1-348



Hit Crossing block 1-242

hybrid systems

linearization 2-4

## I

IC block 1-244

If Action Subsystem block 1-251

If block 1-246

if-else control flow diagram

creating 1-246

image

displaying on mask icon 5-10

drawing on mask icon using patch 5-11

image command 5-10

Increment Stored Integer block 1-253

Index Vector block 1-254

inf values

in mask plotting commands 5-12

Infinite Impulse Response filter 1-168

inherited

data types

by backpropagation 1-122

scaling

by backpropagation 1-122

initial conditions

setting 1-244

initial states 4-13

initial step size 4-13

Inport block 1-255

Inport blocks

in subsystem 1-465

linmod function 2-4

input ports

unconnected 1-241

inputs

applying expressions to 1-207

applying MATLAB function to

Fcn block 1-207

MATLAB Fcn block 1-318

combining into vector line 1-344

delaying by variable amount 1-554

external 4-8

from outside system 1-255

from previous time step 1-323

from workspace 1-256

generating step between two levels 1-462

interpolated mapping 1-300

logical operations on 1-282

multiplying block inputs during simulation  
1-457

outputting minimum or maximum 1-332

passing through stair-step function 1-373

piecewise linear mapping of two 1-294

plotting 1-573

reading from file 1-222

width of 1-570

integration

block input 1-264

discrete-time 1-173

Integrator block 1-264

interpolated mapping 1-300

Interval Test Dynamic block 1-280

ishow debug command 7-39

## J

Jacobians 2-4

## L

left-hand approximation 1-173

limiting

signals 1-411

limiting derivative of signal 1-379

limiting integral 1-266

linear models

    extracting

        linmod function 2-4

linearization

    discrete-time systems 2-4

    linmod function 2-4

linearized matrix

    eigenvalues 2-5

lines

    adding 3-8

    deleting 3-17

linmod function 2-3

    Transport Delay block 1-510

linmod2 function 2-3

local Goto tag visibility 1-237

Logic and Bit Operations block library

    block parameters 9-51

logic circuits

    modeling 1-75

Logical Operator block 1-282

Look-Up Table (2-D) block 1-294

Look-Up Table (n-D) block 1-300

Lookup Tables block library

    block parameters 9-55

lorenzs demo 1-573

## M

MACs

    propagating data type information for 1-126

Magnitude-Angle to Complex block 1-310

Manual Switch block 1-312

mask icon

    displaying graphics on 5-12

    displaying image on 5-10

    displaying port label on 5-13

    displaying symbols and Greek letters on 5-14

    displaying text on 5-6

    displaying text using fprintf 5-9

    displaying text using text 5-14

    displaying transfer function on 5-7

    using the patch function 5-11

mask icons

    displaying symbols and Greek letters on 5-6

    question marks in 5-12

mask parameters

    undefined 5-8

masked blocks

    parameters 9-126

masked subsystems

    question marks in icon 5-12

masking bits 1-33

Math block library

    block parameters 9-60

Math Function block 1-313

mathematical functions

    performing

        Math Function block 1-313

        Rounding Function block 1-408

        Trigonometric Function block 1-520

mathematical symbols

    displaying on mask icons 5-6

    displaying on mask icons using text 5-14

MATLAB

    features NOT supported in eML blocks 11-23

MATLAB character encoding, changing 3-42

MATLAB Fcn block 1-318

MATLAB functions

    applying to block input

        Fcn block 1-207

        MATLAB Fcn block 1-318

matrices

- concatenation 1-321
  - writing to 1-492
- Matrix Concatenation block 1-321
- maximum number of output rows 4-14
- maximum order of ode15s solver 4-13
- maximum step size
  - simset command 4-14
- mdl file 10-2
- Memory block 1-323
- memory region
  - shared
    - Data Store Memory block 1-102
    - Data Store Read block 1-106
    - Data Store Write block 1-109
- Merge block 1-326
- MinMax block 1-332
- minor debug command 7-25
- model command 4-4
- model files 10-2
- Model Info block 1-338
- model parameters
  - table 9-2
- Model Verification block library
  - block parameters 9-74
- models
  - closing 3-12
  - creating
    - new\_system command 3-32
  - getting name 3-13
  - parameters 9-2
  - replacing blocks 3-34
  - simulating 4-7
- Model-Wide Utilities block library
  - block parameters 9-78
- multiplying block inputs
  - during simulation 1-457
- multirate systems

- linearization 2-4
- Mux block 1-344

## N

- Nan values
  - in mask plotting commands 5-12
- nanbreak debug command 7-26
- new\_system command 3-32
- newline in block name 3-4, 5-3
- next debug command 7-27
- nonlinear systems
  - spectral analysis of 1-72
- normally distributed random numbers 1-377
- num2fixpt function 7-28

## O

- objects
  - finding 3-20
  - specifying path 3-4
- obsolete blocks, replacing 3-48
- ode113 solver
  - Memory block 1-323
- ode15s solver
  - maximum order property 4-13
  - Memory block 1-323
- open\_system command 3-33
- opening
  - block dialog boxes 3-33
  - Simulink Library Browser 3-41
  - system windows 3-33
- operating point 2-3
- operators
  - arithmetic, in eML blocks 11-4
  - logical, in eML blocks 11-5
  - relational, in eML blocks 11-5

- options structure
  - getting values 4-16
  - setting values 4-12
- Output blocks 1-347
  - in subsystem 1-465
  - linmod function 2-4
- output
  - maximum rows 4-14
  - outside system 1-347
  - refine factor 4-14
  - selected elements of input vector 1-428
  - selected information about the signal on input 1-365
  - specifying points 4-14
  - switching between two inputs 1-312
  - values
    - displaying 1-187
  - variables 4-14
  - writing to file
    - To File block 1-490
  - writing to workspace
    - To Workspace block 1-492
  - zero within range 1-133
- output ports
  - capping unconnected 1-486
- P**
- parameters
  - adding 3-10
  - block
    - list 9-30
  - deleting 3-18
  - getting values of 3-28
  - masked blocks 9-126
  - model 9-2
    - setting values of
      - set\_param command 3-37
- patch command 5-11
- path
  - specifying 3-4, 5-3
- phase-shifted wave 1-441
- piecewise linear mapping
  - two inputs 1-294
- piecewise linear signal
  - generating
    - Signal Builder block 1-441
- plot command 5-12
- Plot systems Interface 7-26
- plotting input signals
  - Scope block 1-414
  - XY Graph block 1-573
- plotting simulation data 4-10
- port label
  - displaying on mask icon 5-13
- port labels
  - suppressing display 1-348
- port\_label command 5-13
- Ports & Subsystems block library
  - block parameters 9-80
- precision
  - best 7-4
  - maximum 7-5
- probe debug command 7-28
- Product of Elements Inverted block 1-364
- programmable logic arrays
  - modeling 1-75
- propagation of data types 1-122
- properties of Scope block 1-420
- Pulse Generator block 1-368

**Q**

Quantizer block 1-373  
 question marks in mask icon 5-12  
 quit debug command 7-29

**R**

random noise  
     generating 1-377  
 Random Number block 1-377  
     and Band-Limited White Noise block 1-25  
     compared to Band-Limited White Noise block 1-377  
 random numbers  
     generating normally distributed 1-25  
     normally distributed 1-377  
     uniformly distributed 1-523  
 Rate Limiter block 1-379  
 Rate Transition block 1-383  
 reading data  
     from data store 1-106  
     from file 1-222  
     from workspace 1-225  
 Real-Imag to Complex block 1-387  
 refine factor  
     simset command 4-14  
 region of zero output 1-133  
 regular expressions 3-22  
 relative tolerance 4-14  
 Repeating Sequence block 1-397  
 Repeating Sequence Stair block 1-401  
 repeating signals 1-397  
 replace obsolete blocks 3-48  
 replace\_block command 3-34  
 replacing blocks in model 3-34  
 Reshape block 1-405  
 right-hand approximation 1-174

Rounding Function block 1-408  
 run debug command 7-31

**S**

Sample Time Divide block 1-562  
 Sample Time Multiply block 1-563  
 Sample Time Probe block 1-564  
 Sample Time Subtract block 1-564  
 sample-and-hold  
     applying to block input 1-323  
 sampling interval  
     generating simulation time 1-158  
 Saturation block 1-411  
 save\_system command 3-36  
 sawtooth wave  
     generating 1-444  
 Scope axes  
     autoscaling 1-417  
 Scope block 1-414  
     properties 1-420  
     saving axes settings 1-420  
 scoped Goto tag visibility 1-237  
 Selector block 1-428  
 separating vector signal 1-141  
 sequence of signals 1-368  
 sequential circuits  
     implementing 1-77  
 set\_param command 3-37  
 setting bits 1-33  
 setting parameter values 3-37  
 sfix function 7-29  
 sfrac function 7-30  
 S-Function block 1-434  
 S-Function Builder block 1-436  
 shared data store  
     Data Store Memory block 1-102

- Data Store Read block 1-106
- Data Store Write block 1-109
- Sign block 1-439
- Signal Attributes block library
  - block parameters 9-99
- Signal Generator block 1-444
- Signal Inspection block 1-365
- signal processing functions
  - for Embedded MATLAB Function 11-19
- Signal Routing block library
  - block parameters 9-104
- Signal Specification block 1-447
- signals
  - displaying graphically 1-414
  - displaying vector 1-415
  - displaying X-Y plot of 1-573
  - generating pulses 1-368
  - limiting 1-411
  - limiting derivative of 1-379
  - passed from Goto block 1-220
  - passing to From block 1-237
  - plotting
    - Scope block 1-414
    - XY Graph block 1-573
  - repeating 1-397
- sim command 4-7
- simget command 4-16
- simplot command
  - plotting simulation data 4-10
- simset command 4-12
- simulating models 4-7
- simulation
  - parameters
    - specifying using simset command 4-12
  - stopping
    - Stop Simulation block 1-464
- simulation time
  - generating at sampling interval 1-158
  - outputting 1-74
- simulink command 3-41
- Simulink Library Browser
  - opening 3-41
- Simulink.NumericType 8-16
- Simulink.StructType 8-30
- sine wave
  - generating
    - Signal Generator block 1-444
    - Sine Wave block 1-451
  - generating with increasing frequency
    - Chirp Signal block 1-72
- Sine Wave block 1-451
- Sinks block library
  - block parameters 9-109
- sint function 7-31
- slash in block name 3-5
- Slider Gain block 1-457
- slupdate command 3-48
- solvers
  - properties
    - specifying 4-12
    - specifying using simset command 4-14
- Sources block library
  - block parameters 9-114
- spectral analysis of nonlinear systems 1-72
- square wave
  - generating 1-444
- ss2tf function 2-6
- ss2zp function 2-6
- stair-step function
  - passing signal through 1-373
- state derivatives
  - setting to zero 2-7
- state space in discrete system 1-170
- states

- initial 4-13
- outputting 4-14
- resetting 1-266
- saving at end of simulation 4-13
- specifying absolute tolerance for 1-271
- states debug command 7-34
- State-Space block 1-459
- status debug command 7-35
- Step block 1-462
- step debug command 7-36
- stop debug command 7-38
- Stop Simulation block 1-464
- stopping simulation 1-464
- Subsystem block 1-465
- subsystems
  - and Inport blocks 1-255
  - enabled 1-199
  - specifying path 3-4
- Sum block 1-471
- Sum of Elements Negated block 1-475
- Switch Case Action Subsystem block 1-483
- switch control flow diagram
  - creating 1-479
- switching output between inputs
  - Manual Switch block 1-312
- switching output between two inputs 1-312
- System section of mdl file 10-3
- system windows
  - closing 3-14
- systems
  - current 3-27
  - saving 3-36
  - specifying path 3-4
- systems debug command 7-40

## T

- tbreak debug command 7-41
- Terminator block 1-486
- terminators
  - adding 3-11
- TeX formatting commands
  - using in mask icon text 5-14
  - using with disp 5-6
- text command 5-14
- tf2ss utility
  - converting Transfer Fcn to state-space form 1-496
- time delay
  - simulating 1-510
- Time-Based Linearization block 1-487
- To File block 1-490
- To Workspace block 1-492
- trace debug command 7-42
- tracing facilities 4-15
- Transfer Fcn block 1-496
- transfer function form
  - converting to 2-6
- transfer functions
  - discrete 1-183
  - displaying on mask icon 5-7
  - linear 1-496
  - poles and zeros 1-577
  - discrete 1-185
- Transport Delay block 1-510
- Trapezoidal method 1-174
- Trigger block 1-513
- Trigger-Based Linearization block 1-517
- Triggered Subsystem block 1-519
- triggered subsystems
  - Trigger block 1-513
- Trigonometric Function block 1-520
- trim function 2-7

truth tables  
    implementing 1-75

## U

ufix function 7-32  
ufrac function 7-33  
uint function 7-34  
unconnected input ports 1-241  
unconnected output ports  
    using the Terminator block 1-486  
undisp debug command 7-43  
Uniform Random Number block 1-523  
    compared to Band-Limited White Noise block 1-523  
uniformly distributed random numbers 1-523  
Unit Delay block  
    compared to Transport Delay block 1-510  
Unit Delay Enabled External IC block 1-529  
Unit Delay Enabled Resettable block 1-531  
Unit Delay Enabled Resettable External IC block 1-533  
Unit Delay External IC block 1-535  
Unit Delay Resettable block 1-537  
Unit Delay Resettable External IC block 1-539  
Unit Delay With Preview Enabled block 1-541  
Unit Delay With Preview Enabled Resettable block 1-543  
Unit Delay With Preview Enabled Resettable External RV block 1-546  
Unit Delay With Preview Resettable block 1-549  
Unit Delay With Preview Resettable External RV block 1-551  
untrace debug command 7-44  
Update Diagram menu item  
    changing block parameters during simulation 3-37

User-defined functions block library  
    block parameters 9-120

## V

variable time delay 1-554  
Variable Transport Delay block 1-554  
variable types of eML blocks 11-2  
vdp model  
    Scope block 1-416  
vector signals  
    displaying 1-415  
    generating from inputs 1-344  
    separating 1-141  
viscous friction 1-96  
visibility of Goto tag 1-240

## W

while control flow diagram  
    creating 1-565  
While Iterator block 1-565  
While Iterator Subsystem block 1-569  
While subsystems  
    creating 1-565  
white noise  
    generating 1-25  
Width block 1-570  
workspace  
    destination 4-13  
    reading data from 1-225  
    source 4-15  
    writing output to 1-492  
writing data to data store 1-109  
writing output to file 1-490  
writing output to workspace 1-492



**X**

xbreak debug command 7-47

XY Graph block 1-573

**Z**

zcbreak debug command 7-48

zclist debug command 7-49

zero crossings

    detecting

        Hit Crossing block 1-242

        simset command 4-15

zero output in region

    generating 1-133

Zero-Order Hold block

    compared to First-Order Hold block 1-210

Zero-Pole block 1-577

zero-pole form

    converting to 2-6

zooming in on displayed data 1-418