

# SIMULINK<sup>®</sup>

Model-Based and System-Based Design

Modeling

Simulation

Implementation

Simulink Reference

*Version 5*



## How to Contact The MathWorks:



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



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



508-647-7000      Phone



508-647-7001      Fax



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

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

### *Simulink<sup>®</sup> Reference*

© COPYRIGHT 2002 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 or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as "commercial" computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227.7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government's use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government's minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

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)

## Block Libraries

**1**

<b>Continuous</b> .....	<b>1-2</b>
<b>Discontinuities</b> .....	<b>1-3</b>
<b>Discrete</b> .....	<b>1-4</b>
<b>Look-Up Tables</b> .....	<b>1-5</b>
<b>Math Operations</b> .....	<b>1-6</b>
<b>Model Verification</b> .....	<b>1-8</b>
<b>Model-Wide Utilities</b> .....	<b>1-10</b>
<b>Ports &amp; Subsystems</b> .....	<b>1-11</b>
<b>Signal Attributes</b> .....	<b>1-13</b>
<b>Signal Routing</b> .....	<b>1-14</b>
<b>Sinks</b> .....	<b>1-15</b>
<b>Sources</b> .....	<b>1-16</b>
<b>User-Defined Functions</b> .....	<b>1-18</b>
<b>Blocksets and Toolboxes</b> .....	<b>1-19</b>
<b>Demos Library</b> .....	<b>1-20</b>

Abs .....	2-3
Action Port .....	2-5
Algebraic Constraint .....	2-8
Assertion .....	2-10
Assignment .....	2-12
Backlash .....	2-17
Band-Limited White Noise .....	2-21
Bitwise Logical Operator .....	2-23
Bus Creator .....	2-27
Bus Selector .....	2-31
Check Discrete Gradient .....	2-33
Check Dynamic Gap .....	2-36
Check Dynamic Lower Bound .....	2-39
Check Dynamic Range .....	2-42
Check Dynamic Upper Bound .....	2-44
Check Input Resolution .....	2-46
Check Static Gap .....	2-48
Check Static Lower Bound .....	2-51
Check Static Range .....	2-54
Check Static Upper Bound .....	2-57
Chirp Signal .....	2-60
Clock .....	2-62
Combinatorial Logic .....	2-64
Complex to Magnitude-Angle .....	2-68
Complex to Real-Imag .....	2-69
Configurable Subsystem .....	2-70
Constant .....	2-74
Coulomb and Viscous Friction .....	2-77
Data Store Memory .....	2-79
Data Store Read .....	2-82
Data Store Write .....	2-84
Data Type Conversion .....	2-86
Dead Zone .....	2-88
Demux .....	2-90
Derivative .....	2-96
Digital Clock .....	2-98
Direct Look-Up Table (n-D) .....	2-99
Discrete Filter .....	2-105

Discrete State-Space .....	2-107
Discrete-Time Integrator .....	2-109
Discrete Transfer Fcn .....	2-116
Discrete Zero-Pole .....	2-118
Display .....	2-120
DocBlock .....	2-123
Dot Product .....	2-124
Enable .....	2-126
Enabled and Triggered Subsystem .....	2-128
Enabled Subsystem .....	2-129
Fcn .....	2-130
First-Order Hold .....	2-133
For Iterator .....	2-135
For Iterator Subsystem .....	2-140
From .....	2-141
From File .....	2-143
From Workspace .....	2-146
Function-Call Generator .....	2-150
Function-Call Subsystem .....	2-152
Gain, Matrix Gain .....	2-153
Goto .....	2-159
Goto Tag Visibility .....	2-162
Ground .....	2-163
Hit Crossing .....	2-164
IC .....	2-166
If .....	2-168
If Action Subsystem .....	2-173
Inport .....	2-174
Integrator .....	2-179
Interpolation (n-D) Using PreLook-Up .....	2-189
Logical Operator .....	2-192
Look-Up Table .....	2-196
Look-Up Table (2-D) .....	2-202
Look-Up Table (n-D) .....	2-207
Magnitude-Angle to Complex .....	2-213
Manual Switch .....	2-215
Math Function .....	2-216
MATLAB Fcn .....	2-218
Matrix Concatenation .....	2-220
Memory .....	2-222

Merge .....	2-224
MinMax .....	2-228
Model Info .....	2-230
Multi-Port Switch .....	2-233
Mux .....	2-236
Outport .....	2-238
Polynomial .....	2-242
Prelook-Up Index Search .....	2-244
Product .....	2-247
Probe .....	2-252
Pulse Generator .....	2-254
Quantizer .....	2-258
Ramp .....	2-260
Random Number .....	2-262
Rate Limiter .....	2-264
Rate Transition .....	2-266
Real-Imag to Complex .....	2-269
Relational Operator .....	2-271
Relay .....	2-275
Repeating Sequence .....	2-279
Reshape .....	2-281
Rounding Function .....	2-284
Saturation .....	2-286
Scope, Floating Scope .....	2-288
Selector .....	2-302
S-Function .....	2-306
S-Function Builder .....	2-308
Sign .....	2-309
Signal Builder .....	2-310
Signal Generator .....	2-311
Signal Specification .....	2-314
Sine Wave .....	2-317
Slider Gain .....	2-322
State-Space .....	2-324
Step .....	2-327
Stop Simulation .....	2-329
Subsystem, Atomic Subsystem .....	2-330
Sum .....	2-334
Switch .....	2-338
Switch Case .....	2-341

Switch Case Action Subsystem .....	2-345
Terminator .....	2-346
Time-Based Linearization .....	2-347
To File .....	2-349
To Workspace .....	2-351
Transfer Fcn .....	2-355
Transport Delay .....	2-358
Trigger .....	2-361
Trigger-Based Linearization .....	2-363
Triggered Subsystem .....	2-365
Trigonometric Function .....	2-366
Uniform Random Number .....	2-368
Unit Delay .....	2-370
Variable Transport Delay .....	2-372
While Iterator .....	2-375
While Iterator Subsystem .....	2-381
Width .....	2-382
XY Graph .....	2-383
Zero-Order Hold .....	2-385
Zero-Pole .....	2-387

## Linearization and Trimming Commands

### 3

linmod, dlinmod, linmod2 .....	3-2
trim .....	3-5

## Model Construction Commands

### 4

add_block .....	4-6
add_line .....	4-7
add_param .....	4-9
bdclose .....	4-10
bdroot .....	4-11
close_system .....	4-12
compare_model .....	4-14

delete_block .....	4-15
delete_line .....	4-16
delete_param .....	4-17
find_system .....	4-18
gcb .....	4-23
gcbh .....	4-24
gcs .....	4-25
get_param .....	4-26
new_system .....	4-28
open_system .....	4-29
replace_block .....	4-30
save_system .....	4-32
set_param .....	4-33
simulink .....	4-35
sldiscmdl .....	4-36
slmdliscui .....	4-40

## Simulation Commands

### 5

model .....	5-5
sim .....	5-7
simplot .....	5-10
simset .....	5-12
simget .....	5-16

## Mask Icon Drawing Commands

### 6

disp .....	6-5
dpoly .....	6-6
fprintf .....	6-8
image .....	6-9
patch .....	6-10
plot .....	6-11
port_label .....	6-12
text .....	6-13



ashow	7-1
atrace	7-2
bafter	7-3
break	7-4
bshow	7-5
clear	7-6
continue	7-7
disp	7-8
help	7-9
emode	7-10
ishow	7-11
minor	7-12
nanbreak	7-13
next	7-14
probe	7-15
quit	7-16
run	7-17
slist	7-18
states	7-19
status	7-20
step	7-21
stop	7-22
systems	7-23
tbreak	7-24
trace	7-25
undisp	7-26
untrace	7-27
xbreak	7-28
zcbreak	7-29
zclist	7-30

## Model and Block Parameters

---

### 8

<b>Model Parameters</b> .....	<b>8-2</b>
<b>Common Block Parameters</b> .....	<b>8-7</b>
<b>Block-Specific Parameters</b> .....	<b>8-10</b>
<b>Mask Parameters</b> .....	<b>8-26</b>

## Model File Format

---

### 9

<b>Model File Contents</b> .....	<b>9-2</b>
Model Section .....	<b>9-3</b>
BlockDefaults Section .....	<b>9-3</b>
AnnotationDefaults Section .....	<b>9-3</b>
System Section .....	<b>9-3</b>

# Block Libraries

---

The following sections describe the usage and contents of the Simulink block libraries. You can use either the Simulink Library Browser on Windows or the MATLAB command `simulink` on UNIX to display and browse the libraries.

## Continuous

The *Continuous* library contains blocks that model linear functions.

<b>Block Name</b>	<b>Purpose</b>
Derivative	Output the time derivative of the input.
Integrator	Integrate a signal.
State-Space	Implement a linear state-space system.
Transfer Fcn	Implement a linear transfer function.
Transport Delay	Delay the input by a given amount of time.
Variable Transport Delay	Delay the input by a variable amount of time.
Zero-Pole	Implement a transfer function specified in terms of poles and zeros.

## Discontinuities

The *Discontinuities* library contains blocks whose outputs are discontinuous functions of their inputs.

<b>Block Name</b>	<b>Purpose</b>
Backlash	Model the behavior of a system with play.
Coulomb and Viscous Friction	Model discontinuity at zero, with linear gain elsewhere.
Dead Zone	Provide a region of zero output.
Hit Crossing	Detect crossing point.
Quantizer	Discretize input at a specified interval.
Rate Limiter	Limit the rate of change of a signal.
Relay	Switch output between two constants.
Saturation	Limit the range of a signal.

## Discrete

The *Discrete* library contains blocks that represent discrete-time functions.

<b>Block Name</b>	<b>Purpose</b>
Discrete Filter	Implement IIR and FIR filters.
Discrete State-Space	Implement a discrete state-space system.
Discrete Transfer Fcn	Implement a discrete transfer function.
Discrete Zero-Pole	Implement a discrete transfer function specified in terms of poles and zeros.
Discrete-Time Integrator	Perform discrete-time integration of a signal.
First-Order Hold	Implement a first-order sample-and-hold.
Memory	Output the block input from the previous time step.
Unit Delay	Delay a signal one sample period.
Zero-Order Hold	Implement zero-order hold of one sample period.

## Look-Up Tables

The *Look-Up Tables* library contains blocks that use lookup tables to determine outputs from inputs.

<b>Block Name</b>	<b>Purpose</b>
Direct Look-Up Table (n-D)	Index into an N-dimensional table to retrieve a scalar, vector or 2-D matrix.
Interpolation (n-D) Using PreLook-Up	Perform high-performance constant or linear interpolation.
Look-Up Table	Perform piecewise linear mapping of the input.
Look-Up Table (2-D)	Perform piecewise linear mapping of two inputs.
Look-Up Table (n-D)	Perform piecewise linear or spline mapping of two or more inputs.
Prelook-Up Index Search	Perform index search and interval fraction calculation for input on a breakpoint set.

## Math Operations

The *Math Operations* library contains blocks that model general mathematical functions.

Block Name	Purpose
Abs	Output the absolute value of the input.
Algebraic Constraint	Constrain the input signal to zero.
Assignment	Assign values to specified elements of a signal
Bitwise Logical Operator	Logically mask, invert, or shift the bits of an unsigned integer signal.
Combinatorial Logic	Implement a truth table.
Complex to Magnitude-Angle	Output the phase and magnitude of a complex input signal.
Complex to Real-Imag	Output the real and imaginary parts of a complex input signal.
Dot Product	Generate the dot product.
Gain, Matrix Gain	Multiply block input by a specified value.
Logical Operator	Perform the specified logical operation on the input.
Magnitude-Angle to Complex	Output a complex signal from magnitude and phase inputs.
Math Function	Perform a mathematical function.
Matrix Concatenation	Concatenate inputs horizontally or vertically
MinMax	Output the minimum or maximum input value.



<b>Block Name</b>	<b>Purpose</b>
Polynomial	Perform evaluation of polynomial coefficients on input values.
Product	Generate the product or quotient of block inputs.
Real-Imag to Complex	Output a complex signal from real and imaginary inputs.
Relational Operator	Perform the specified relational operation on the input.
Reshape	Change the dimensionality of a signal.
Rounding Function	Perform a rounding function.
Sign	Indicate the sign of the input.
Slider Gain	Vary a scalar gain using a slider.
Sum	Generate the sum of inputs.
Trigonometric Function	Perform a trigonometric function.

## Model Verification

**Acknowledgment.** The Model Verification blocks were developed in conjunction with the Control System Design team of the Advanced Chassis SystemDevelopment group of DaimlerChrysler AG, Stuttgart, Germany.

The Model Verification library contains blocks that enable you to create self-validating models.

Block Name	Purpose
Assertion	Assert that the input signal is nonzero.
Check Discrete Gradient	Check that the absolute value of the difference between successive samples of a discrete signal is less than an upper bound.
Check Dynamic Gap	Check that a gap of varying width occurs in the range of a signal's amplitudes.
Check Dynamic Lower Bound	Check that a signal is always greater than a value that can vary at each time step.
Check Dynamic Range	Check that a signal always lies in a varying range of amplitudes.
Check Dynamic Upper Bound	Check that a signal is always less than a value that can vary at each time step.
Check Input Resolution	Check that a signal has a specified resolution.
Check Static Gap	Check that a fixed-width gap occurs in the range of a signal's amplitudes
Check Static Lower Bound	Check that a signal is greater than (or optionally equal to) a lower bound that does not vary with time.
Check Static Range	Check that the input signal falls in a fixed range of amplitudes.

---

<b>Block Name</b>	<b>Purpose</b>
Check Static Upper Bound	Check that a signal is less than (or optionally equal to) an upper bound that does not vary with time.

## Model-Wide Utilities

The *Model-Wide Utilities* library contains various utility blocks.

<b>Block Name</b>	<b>Purpose</b>
DocBlock	Create text that documents the model and save the text with the model.
Model Info	Display revision control information in a model.
Time-Based Linearization	Generate linear models in the base workspace at specific times.
Trigger-Based Linearization	Generate linear models in the base workspace when triggered.

## Ports & Subsystems

The Ports & Subsystems library contains blocks for creating various types of subsystems.

Block Name	Purpose
Action Port	Repository for conditionally executed logic for If and Switch-Case blocks. Note: this block resides inside the If Action Subsystem and Switch-Case Action Subsystem blocks in the Subsystems library.
Configurable Subsystem	Represent any block selected from a specified library.
Enable	Add an enabling port to a subsystem. Note that this block resides inside the Enabled Subsystem and the Enabled and Triggered Subsystem in the Subsystems library.
Enabled and Triggered Subsystem	Skeleton enabled and triggered subsystem.
Enabled Subsystem	Skeleton enabled subsystem.
For Iterator	Implements C-like for statement logic.
For Iterator Subsystem	Implements a C-like for loop.
Function-Call Subsystem	Skeleton function call subsystem.
If	Implements C-like if - else statement logic.
Inport	Create an input port for a subsystem or an external input. Note that this block resides inside the Subsystem block and inside other subsystem blocks in the Subsystems library.

<b>Block Name</b>	<b>Purpose</b>
Output	Create an output port for a subsystem or an external output. Note that this block resides inside the Subsystem block and inside other subsystem blocks in the Subsystems library.
Subsystem, Atomic Subsystem	Represent a system within another system.
Switch Case	Implements C-like switch statement logic.
Switch Case Action Subsystem	Represent a subsystem whose execution is triggered by a Switch Case block.
Trigger	Add a trigger port to a subsystem. Note this block resides inside the Triggered Subsystem and the Enabled and Triggered Subsystem in the Subsystems library.
Triggered Subsystem	Skeleton triggered subsystem.
While Iterator	Implement a C-like <i>while</i> or <i>do-while</i> control flow statement as a While subsystem.
While Iterator Subsystem	Represent a subsystem that executes repeatedly while a condition is satisfied during a simulation time step.

## Signal Attributes

The *Signal Attributes* library contains blocks that modify or output attributes of signals.

<b>Block Name</b>	<b>Purpose</b>
Data Type Conversion	Convert a signal to another data type.
IC	Set the initial value of a signal.
Rate Transition	Specify the data transfer mechanism between the data rates of a multirate system.
Probe	Output a signal's attributes, including width, sample time, and/or signal type.
Signal Specification	Specify attributes of a signal.
Width	Output the width of the input vector.

## Signal Routing

The *Signal Routing* library contains blocks that route signals from one point in a block diagram to another.

<b>Block Name</b>	<b>Purpose</b>
Bus Creator	Create a signal bus.
Bus Selector	Output signals selected from an input bus.
Data Store Memory	Define a shared data store.
Data Store Read	Read data from a shared data store.
Data Store Write	Write data to a shared data store.
Demux	Separate a vector signal into output signals.
From	Accept input from a Goto block.
Goto	Pass block input to From blocks.
Goto Tag Visibility	Define the scope of a Goto block tag.
Manual Switch	Switch between two inputs.
Merge	Combine several input lines into a scalar line.
Multi-Port Switch	Choose between block inputs.
Mux	Combine several input lines into a vector line.
Selector	Select or reorder the elements of the input vector.
Switch	Switch between two inputs.



## Sinks

The *Sinks* library contains blocks that display or write block output.

<b>Block Name</b>	<b>Purpose</b>
Display	Show the value of the input.
Output	Create an output port for a subsystem or an external output.
Scope, Floating Scope	Display signals generated during a simulation.
Stop Simulation	Stop the simulation when the input is nonzero.
Terminator	Terminate an unconnected output port.
To File	Write data to a file.
To Workspace	Write data to a variable in the workspace.
XY Graph	Display an X-Y plot of signals using a MATLAB figure window.

## Sources

The *Sources* library contains blocks that generate signals.

<b>Block Name</b>	<b>Purpose</b>
Band-Limited White Noise	Introduce white noise into a continuous system.
Chirp Signal	Generate a sine wave with increasing frequency.
Clock	Display and provide the simulation time.
Constant	Generate a constant value.
Digital Clock	Generate simulation time at the specified sampling interval.
From File	Read data from a file.
From Workspace	Read data from a variable defined in the workspace.
Ground	Ground an unconnected input port.
Inport	Create an input port for a subsystem or an external input.
Pulse Generator	Generate pulses at regular intervals.
Ramp	Generate a constantly increasing or decreasing signal.
Random Number	Generate normally distributed random numbers.
Repeating Sequence	Generate a repeatable arbitrary signal.
Signal Builder	Generate an arbitrary piecewise linear signal.
Signal Generator	Generate various waveforms.

---

<b>Block Name</b>	<b>Purpose</b>
Sine Wave	Generate a sine wave.
Step	Generate a step function.
Uniform Random Number	Generate uniformly distributed random numbers.

## User-Defined Functions

The *User-Defined Functions* library contains blocks that allow you to define the function that relates inputs to outputs.

<b>Block Name</b>	<b>Purpose</b>
Fcn	Apply a specified expression to the input.
MATLAB Fcn	Apply a MATLAB function or expression to the input.
S-Function	Access an S-function.
S-Function Builder	Builds a C MEX S-function from specifications and code that you supply.

## **Blocksets and Toolboxes**

The *Blocksets and Toolboxes* library contains the Extras block library of specialized blocks.

## **Demos Library**

The *Demos* library contains useful MATLAB and Simulink demos.

# 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 Blockset 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 and Fixed-Point block libraries, execute the following command at the MATLAB command line:

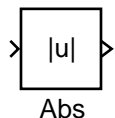
```
showblockdatatypetable
```



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

**Library** Simulink Math Operations and Fixed-Point Blockset Math

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

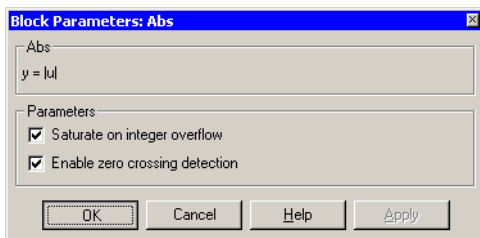


For signed data types, the absolute value of the most negative value is problematic since it is not representable by the data type. In this case, the behavior of the block is controlled by the **Saturate on integer overflow** check box. If checked, the absolute value of the data type saturates to the 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 the **Saturate on integer overflow** check box is selected, 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** An Abs block accepts a real- or complex-valued input of any data type except int64 and uint64 and outputs a real value of the same data type as the input.

**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.

### **Characteristics**

Dimensionalized	Yes
Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Zero Crossing	No, unless <b>Enable zero crossing detection</b> is selected

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

**Library** Ports & Subsystems

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

A rectangular icon with a black border containing the word "Action" in a sans-serif font.

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.  
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{ }`).
- 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

# Action Port

---

initial values when it is reenabled. See the “Parameters and Dialog Box” section following for more details.

---

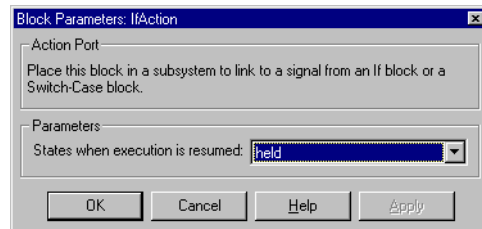
**Note** All blocks in an Action subsystem driven by an If or Switch Case block must run at the same rate as the driving block.

---

## Data Type Support

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

## Parameters and Dialog Box

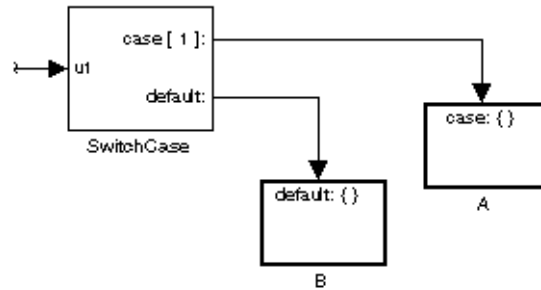


### States when execution is resumed

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

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

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



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

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

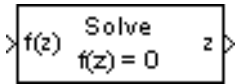
**Characteristics**    Sample Time                      Inherited from driving If or Switch Case block.

# Algebraic Constraint

**Purpose** Constrain the input signal to zero

**Library** Math Operations

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

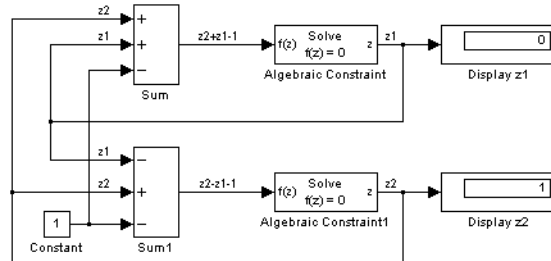


By default, the **Initial guess** parameter is zero. You can improve the efficiency of the algebraic loop solver by providing an **Initial guess** for the algebraic state  $z$  that is close to the solution value.

For example, the following model solves these equations.

$$\begin{aligned} z_2 + z_1 &= 1 \\ z_2 - z_1 &= 1 \end{aligned}$$

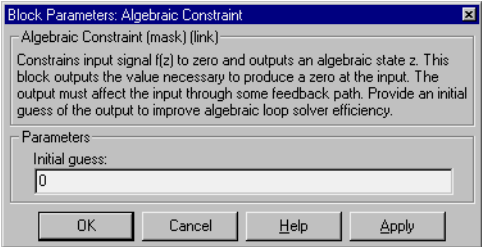
The solution is  $z_2 = 1$ ,  $z_1 = 0$ , as the Display blocks show.



## Data Type Support

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

## Parameters and Dialog Box



### Initial guess

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

## Characteristics

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

# Assertion

**Purpose** Check whether a signal is nonzero

**Library** Model Verification

## Description



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

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

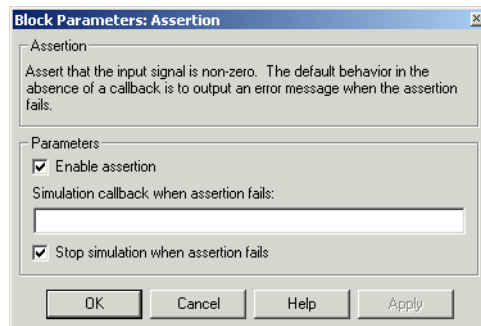
You can also use the **Advanced Pane** of the **Simulation Parameters** dialog box to enable or disable all Assertion blocks in a model.

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, including fixed-point data types, except int64 and uint64.

## 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 **Advanced Pane** of the **Simulation 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

## Characteristics

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

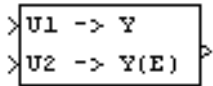
# Assignment

---

**Purpose** Assign values to specified elements of a signal

**Library** Math Operations

## Description



The Assignment block assigns values to specified elements of the signal connected to its U1 port. 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. You specify the values to be assigned to the signal at U1 by connecting a values signal to the Assignment block's U2 port. The block replaces the specified elements of U1 with elements from U2, leaving unassigned elements unchanged, and outputs the result.

You can use the block to assign values to scalar, 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. The index source can specify any of the following values as indices:

- -1 (internal source only)  
Replaces every element of U1 with the corresponding element of U2.
- Index of a single element specified as a positive integer  
Assigns a value to the specified element of U1, leaving the values of all the other elements unchanged.
- A set of indices specified as a vector  
Replaces the specified set of elements of U1 with elements of U2.

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 U1 specified by the first index, the second element of U2 to the U1 element specified by the second index, and so on.

## 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 U1 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 U1.
- Single row or index value  
Specifies a single row or column of U1.
- Vector of row or column indices  
Specifies a set of rows or columns of U1.

The block assigns values from U2 to the specified elements of U1 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 U1. It assigns the second element of the first row of U2 to the second specified element of the first specified row of U1, 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 single row or column 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 U1. For example, suppose your row and column index sources specify a single row index and two column indices. Then U2 must be a 1-by-2 matrix.

# Assignment

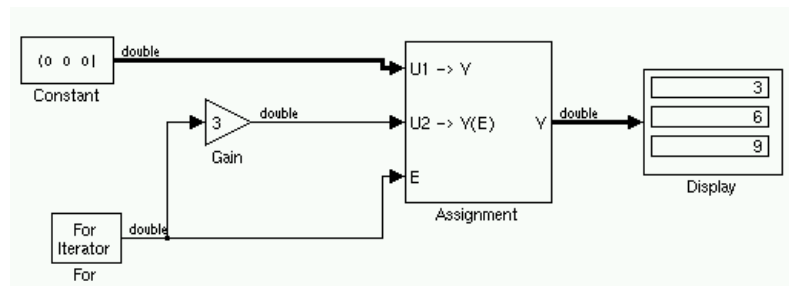
---

**Note** An Assignment block whose **Input type** is Matrix accepts only matrix signals at its U1 and U2 ports. Simulink displays an error dialog box if you update or simulate a model that connects a vector signal to either the U1 or U2 port of an Assignment block whose **Input type** is Matrix.

---

## Iterated Assignment

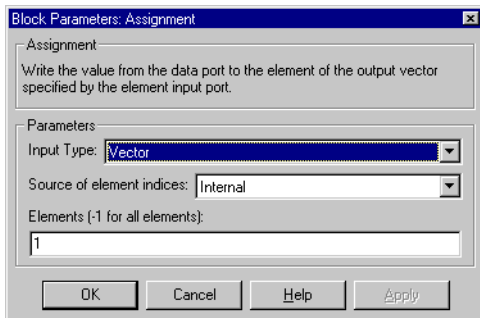
You can use the Assignment block to assign values computed in an iterator (For or While) loop to a vector or matrix signal. To do this, use an iterator block to generate the indices required by the Assignment block (or two iterator blocks if you need to compute row and column indices separately). For example, the following model uses a For block to create a vector signal each of whose elements equals  $3*i$  where  $i$  is the index of the element.



## Data Type Support

The Assignment block accepts signals of any data type, including fixed-point data types.

## 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.

### 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 **U1** to be replaced by elements in **U2**. The value of this parameter can be **-1**, a positive integer specifying a single index, or a vector of positive 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]**).

# Assignment

---

## 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]).

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

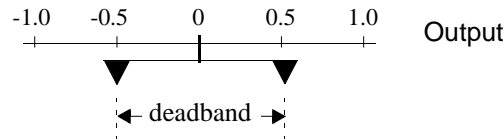
**Purpose** Model the behavior of a system with play

**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:

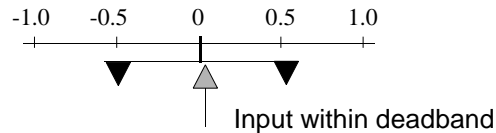
- Disengaged – In this mode, the input does not drive the output and the output remains constant.
- Engaged in a positive direction – In this mode, the input is increasing (has a positive slope) and the output is equal to the input *minus* half the deadband width.
- Engaged in a negative direction –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

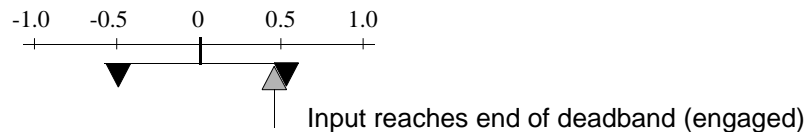
For example, the Backlash block can be used to model the meshing of two gears. The input and output are both shafts with a gear on one end, and the output shaft is driven by the input shaft. Extra space between the gear teeth introduces *play*. The width of this spacing is the **Deadband width** parameter.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

# Backlash

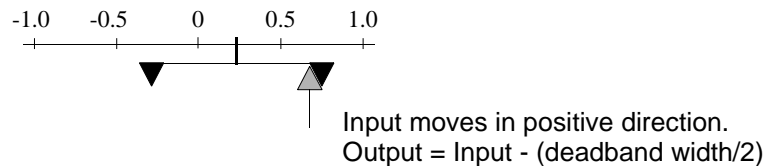
The following figures illustrate the block's operation when the initial input is within the deadband. The first figure shows the relationship between the input and the output while the system is in disengaged mode (and the default parameter values are not changed).



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.



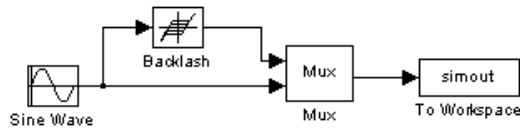
If the input reverses its direction, it disengages from the output. The output remains constant until the input either reaches the opposite end of the deadband or reverses its direction again and engages at the same end of the deadband. Now, as before, movement in the input causes equal movement in the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

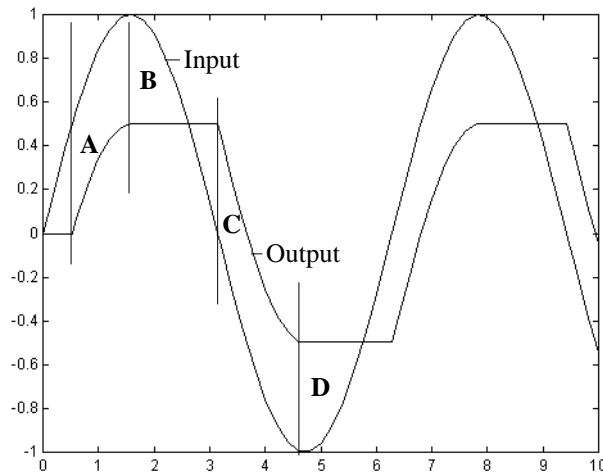
- 5 if the input,  $u$ , is between 4 and 6
- $u + 1$  if  $u < 4$
- $u - 1$  if  $u > 6$



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.



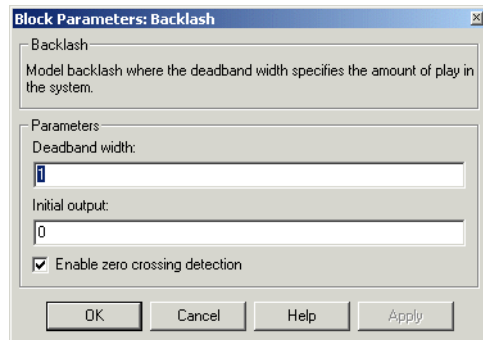
- A** Input engages in positive direction. Change
- B** Input disengages. Change in input does not affect output.
- C** Input engages in negative direction. Change in input causes equal change in output.
- D** Input disengages. Change in input does not affect output.

## Data Type Support

A Backlash block accepts and outputs real values of type double.

# Backlash

## Parameters and Dialog Box



### Deadband width

The width of the deadband. The default is 1.

### Initial output

The initial output value. The default is 0.

### 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.

## Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes
Dimensionalized	Yes
Zero Crossing	Yes, if <b>Enable zero crossing detection</b> is selected.

**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

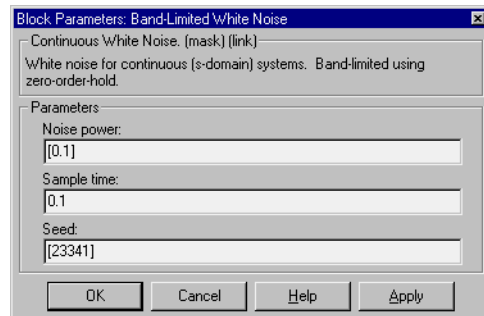
---

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

A 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.

## Characteristics

Sample Time	Discrete
Scalar Expansion	Of <b>Noise power</b> and <b>Seed</b> parameters and output
Dimensionalized	Yes
Zero Crossing	No

**Purpose** Logically mask, invert, or shift the bits of an unsigned integer signal

**Library** Math Operations

## Description



The Bitwise Logical Operator performs any of a set of logical masking (AND, OR, XOR), inversion (NOT), and shifting (SHIFT\_LEFT, SHIFT\_RIGHT) operations on the bits of an unsigned integer signal. The block's parameter dialog lets you choose the operation to perform. You can use the Bitwise Logical Operator block to perform bitwise operations on arrays of unsigned integer signals.

## Masking Operations

The Bitwise Logical Operator's masking operations (AND, OR, XOR) logically combine each bit of the input signal with the corresponding bit of a constant operand called the mask. You specify the mask's value and the logical operation via the block's parameter dialog. The mask and the logical operation determine the value of each bit of the output signal as follows.

Operation	Mask Bit	Input Bit	Output Bit
AND	1	1	1
	1	0	0
	0	1	0
	0	0	0
OR	1	1	1
	0	1	1
	1	0	1
	0	0	0
XOR	1	1	0
	1	0	1
	0	1	1
	0	0	0

# Bitwise Logical Operator

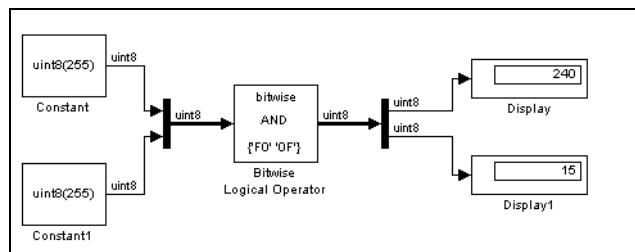
A Bitwise Operator block accepts arrays for both signals and masks. In general, the mask must have the same dimensionality as the input signal, i.e., a 5-by-4 input signal requires a 5-by-4 mask. The block applies each element of the mask to the corresponding input element. The following exceptions exist to the general rule that the input and the mask must have the same dimensionality:

- If the input is scalar and the mask is an array, the block outputs an array consisting of the result of applying each mask element to the input.
- If the input is an array and the mask is a scalar, the block outputs an array consisting of the result of applying the mask to each element of the input.
- If the input is a 1-D array (i.e., a vector), the mask can be a row or a column vector.

When selecting a masking operation, use the **Second operand** field of the block's parameter dialog to specify the mask or masks. You can enter any MATLAB expression that evaluates to a scalar, matrix, or cell array. Use strings in your mask expression to specify hexadecimal values (e.g., 'FFFF').

If necessary, the block truncates the high-order bits of the mask value to fit the word size of the input signal's data type. For example, suppose you specify the mask value as 'FF00' and the input signal is of type `uint8`. The block truncates the specified value to '00'.

You can use matrices to specify hexadecimal masks, but beware of the pitfalls of such an approach. For example, the MATLAB expression `['00' 'FF']` represents a single string 'FF00' rather than two strings. Similarly, the expression `['FFFF'; '0000']` represents two strings but the expression `['FFFF'; '00']` is invalid and hence causes MATLAB to signal an error. You can avoid these pitfalls by always using cell arrays to specify hexadecimal values, or to mix decimal and hexadecimal values, for masks. For example, the following model



uses a cell array ({'F0' '0F'}) to specify hexadecimal values for the masks for a two-element input vector.

## Inversion Operation

The Bitwise Logical Operator's NOT operation inverts the bits of the input signal. In particular, it performs a one's complement operation on the input signal to produce an output signal each of whose bits is 1 if the corresponding input bit is 0 and vice versa.

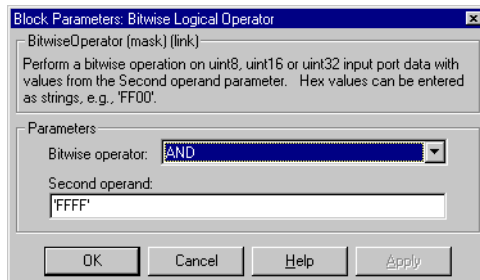
## Shift Operations

The Bitwise Logical Operator's shift operations, SHIFT\_LEFT and SHIFT\_RIGHT, shift the bits of the input signal left or right to produce the output signal. You specify the amount of the shift in the **Second operand** field of the block's parameter dialog. If you specify a shift amount that is greater than the word size of the input signal, the block uses the input word size as the shift amount, resulting in a zero output signal. The dimensionality rules that apply to masks and inputs also apply to shift factors and inputs.

## Data Type Support

The Bitwise Logical Operator accepts real-valued inputs of any of the unsigned integer data types: uint8, uint16, uint32. All the elements of a vector input must be of the same data type. The output signal is of the same data type as the input.

## Parameters and Dialog Box



## Bitwise operator

Specifies the bitwise operator applied to the input signal.

# Bitwise Logical Operator

---

## Second operand

Specifies the mask operand for masking operations and the shift amount for shift operations. You can enter any MATLAB expression that evaluates to a scalar, matrix, or cell array. If the block input is an array, the block applies each parameter value to the corresponding element of the input. If the input is a scalar, the block outputs an array, each of whose elements is the result of applying the corresponding parameter value to the input. (If the **Bitwise operator** is NOT, this parameter does not appear.)

<b>Characteristics</b>	Sample Time	Inherited from driving block
	Scalar Expansion	Of inputs and <b>Second operand</b> parameter
	Dimensionalized	Yes
	States	None
	Zero Crossing	No
	Direct Feedthrough	Yes



**Purpose** Create a signal bus

**Library** Signal Routing

## Description



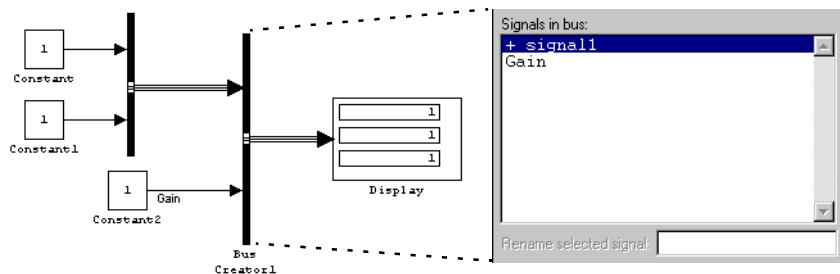
The Bus Creator block combines a set of signals into a bus, i.e., a group of signals represented by a single line in a block diagram. The Bus Creator block, when used in conjunction with the Bus Selector block, allows you to reduce the number of lines required to route signals from one part of a diagram 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 2-28) 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 Simulink's **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.

---

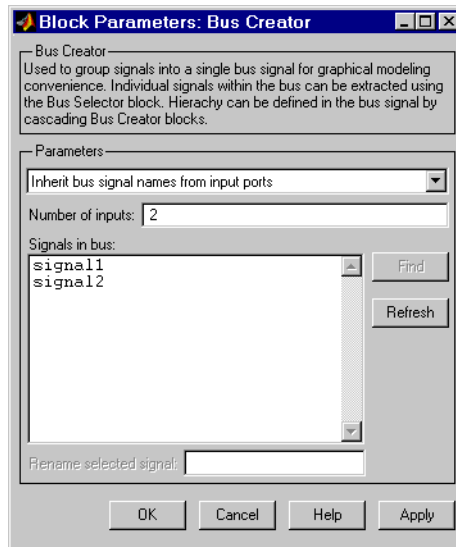
**Note** Simulink hides the name of a Bus Creator block when you copy it from the Simulink library to a model.

---

## Data Type Support

A Bus Creator block accepts and outputs real or complex values of any data type, including fixed-point data types, except int64 and uint64.

## 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 the Require input signal names to match signals below option is selected. Edit this field to change the name of the currently selected signal.

**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.



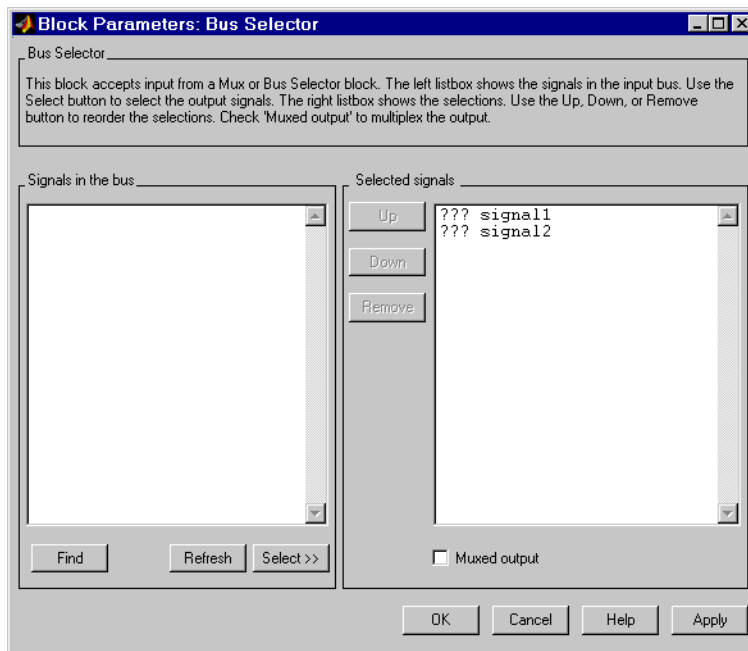
---

**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, including fixed-point data types, except int64 and uint64.

## Parameters and Dialog Box



# Bus Selector

---

## 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.

## 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.

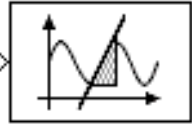
If an output signal listed in the **Selected signals** list box is not an input to the Bus Selector block, the signal name is preceded by three question marks (???).

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.

**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 Simulink's Diagnostic Viewer.

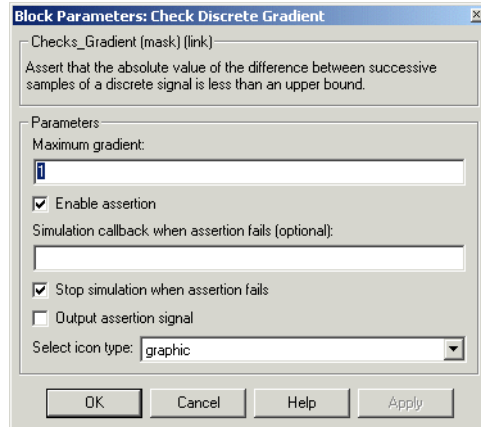
You can also use the **Advanced Pane** of the **Simulation Parameters** dialog box to enable or disable all model verification blocks, including Check Discrete Gradient blocks, in a model.

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 input signals of any dimensions and any built-in data type except int64 and uint64.

# Check Discrete Gradient

## 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 **Advanced Pane** of the **Simulation Parameters** dialog box allows you to enable or disable all Check Discrete Gradient blocks in a model regardless of the setting of this option.

### Simulation callback when assertion fails

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

### Stop simulation when assertion fails

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

### Output Assertion Signal

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



Simulink's **Simulation 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

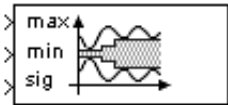
# Check Dynamic Gap

---

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

**Library** Model Verification

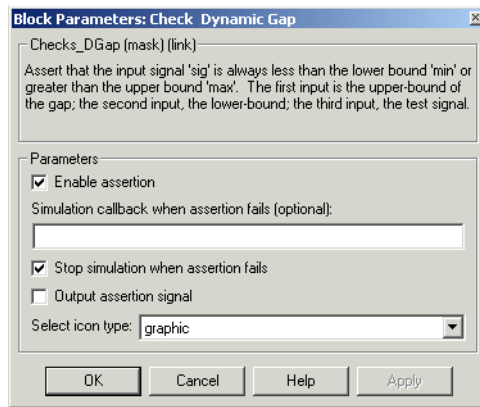
**Description** The Check Dynamic Gap block checks that a gap of possibly varying width occurs in the range of a signal's amplitudes. The test signal is the signal connected to the input labeled *sig*. The inputs labeled *min* and *max* specify the lower and upper bounds of the dynamic gap, respectively. 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 any built-in data type except `int64` and `uint64`. 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.

## 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 **Advanced Pane** of the **Simulation 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

### Output Assertion Signal

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

# Check Dynamic Gap

---

## 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.

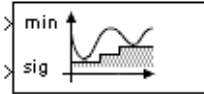
<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Check Dynamic Lower Bound

**Purpose** Check that one signal is always less than another signal

**Library** Model Verification

## Description



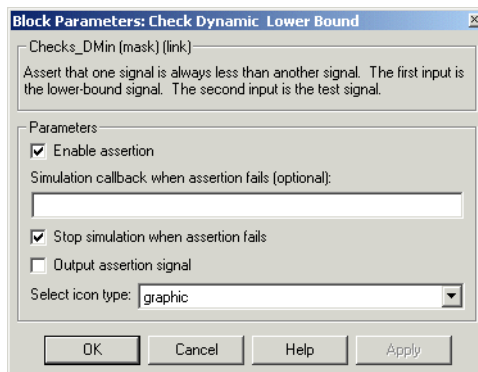
The Check Dynamic Lower Bound block checks that the amplitude of a test signal is less than the amplitude of a reference signal at the current time step. The test signal is the signal connected to the input labeled *sig*. If the verification condition is true, the block does nothing. If not, the block 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 dimensions and any built-in data type except `int64` and `uint64`. 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.

## Parameters and Dialog Box



# Check Dynamic Lower Bound

---

## 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 **Advanced Pane** of the **Simulation 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

## Output Assertion Signal

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

# Check Dynamic Lower Bound

---

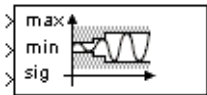
<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Check Dynamic Range

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

**Library** Model Verification

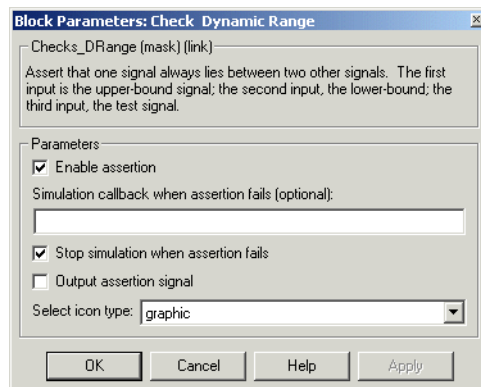
**Description** The Check Dynamic Range block checks that a test signal falls inside a range of amplitudes at each time step. The width of the range can vary from time step to time step. The input labeled *sig* is the test signal. The inputs labeled *min* and *max* are the lower and upper bounds of the 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 any built-in data type except `int64` and `uint64`. 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.

## 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 **Advanced Pane** of the **Simulation 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

## Output Assertion Signal

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

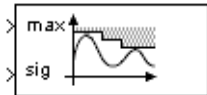
<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Check Dynamic Upper Bound

**Purpose** Check that one signal is always greater than another signal

**Library** Model Verification

## Description



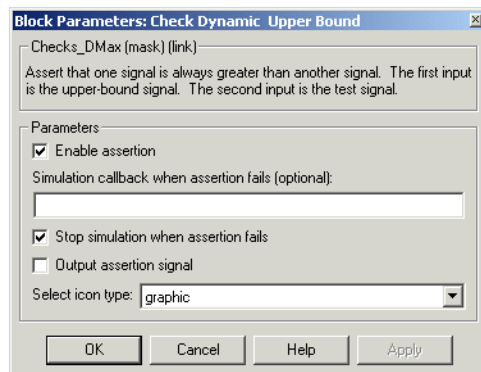
The Check Dynamic Upper Bound block checks that the amplitude of a test signal is greater than the amplitude of a reference signal at the current time step. The test signal is the signal connected to the input labeled *sig*. If the verification condition is true, the block does nothing. If not, the block 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 any built-in data type except `int64` and `uint64`. 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.

## Parameters and Dialog Box



## Enable Assertion

# Check Dynamic Upper Bound

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 **Advanced Pane** of the **Simulation 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

## Output Assertion Signal

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

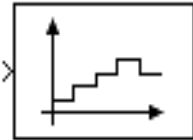
<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Check Input Resolution

**Purpose** Check that the input signal has a specified resolution

**Library** Model Verification

## Description



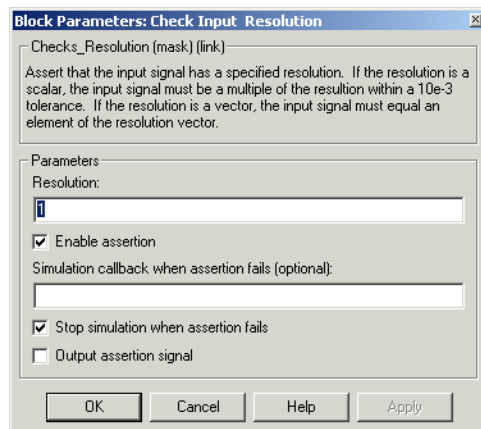
The Check Input Resolution block checks whether the input signal has a specified scalar or vector resolution (see “Resolution” on page 2-47). If the resolution is a scalar, the input signal must be a multiple of the resolution within a 10e-3 tolerance. If the resolution is a vector, the input signal 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 any built-in data type except `int64` and `uint64`. If the input signal is nonscalar, the block checks the resolution of each element of the input test signal.

## 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 **Advanced Pane** of the **Simulation 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

## Output Assertion Signal

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

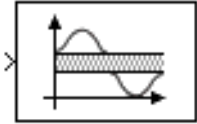
<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

# Check Static Gap

**Purpose** Check that a gap exists in a signal's range of amplitudes

**Library** Model Verification

## Description



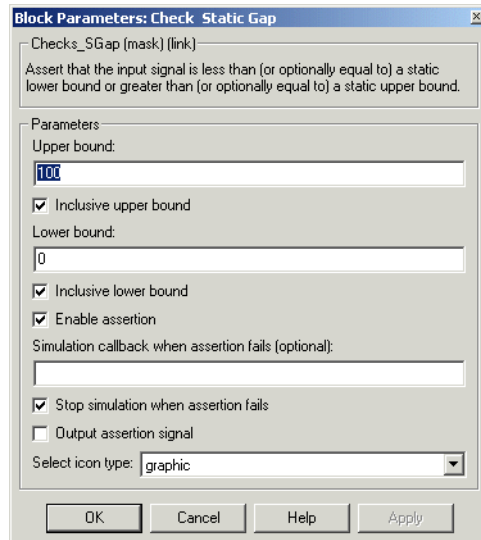
The Check Static Gap block checks that each element of the input signal is less than (or optionally equal to) a static lower bound or greater than (or optionally equal to) a static upper bound at the current time step. If the verification condition is true, the block does nothing. If not, the block halts 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 any built-in data type except int64 and uint64.

## 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 **Advanced Pane** of the **Simulation 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.

## 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

## Output Assertion Signal

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

# Check Static Gap

---

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.

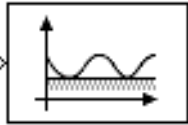
<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No



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

**Library** Model Verification

## Description

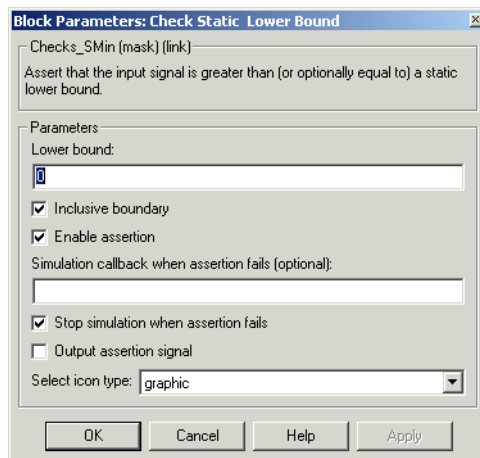


The Check Static Lower Bound block checks that each element of the input signal is greater than (or optionally equal to) a specified lower bound at the current time step. The block's parameter dialog box allows you to specify the value of the lower bound and whether the lower bound is inclusive. If the 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 any built-in data type except int64 and uint64.

## Parameters and Dialog Box



# Check Static Lower Bound

---

## 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 **Advanced Pane** of the **Simulation 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

## Output Assertion Signal

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

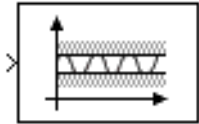
# Check Static Range

---

**Purpose** Check that a signal falls inside a fixed range of amplitudes

**Library** Model Verification

## Description



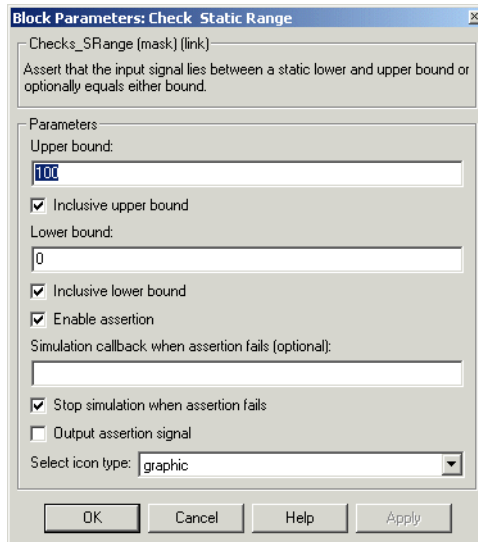
The Check Static Range block checks that each element of the input signal falls inside the same range of amplitudes at each time step. The block's parameter dialog box allows you to specify the upper and lower bounds of the valid amplitude range and whether the range includes the bounds. If the verification 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 any built-in data type except `int64` and `uint64`.

## 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 **Advanced Pane** of the **Simulation 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

## Output Assertion Signal

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

## Select Icon Type

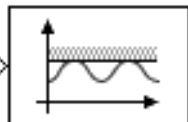
Type of icon used to display this block in a block diagram: either graphic or text. The graphic option displays a graphical representation of the assertion condition on the icon. The text option displays a mathematical expression that represents the assertion condition. If the icon is too small to display 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 signal is greater than (or optionally equal to) a static lower bound

**Library** Model Verification

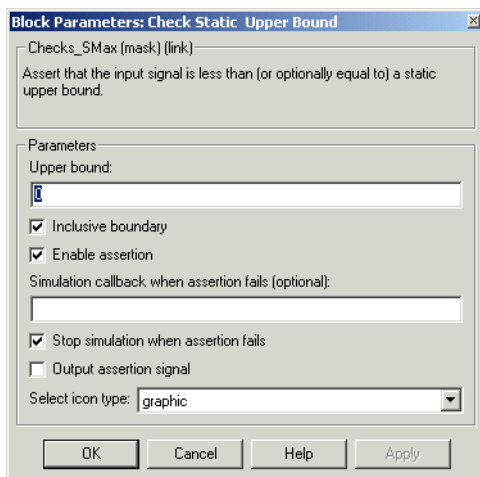
**Description** The Check Static Upper Bound block checks that each element of the input signal is less than (or optionally equal to) a specified lower bound at the current time step. The block's parameter dialog box allows you to specify the value of the upper bound and whether the bound is inclusive. 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 any built-in data type except int64 and uint64.

## Parameters and Dialog Box



# Check Static Upper Bound

---

## 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 **Advanced Pane** of the **Simulation 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 Simulink's **Simulation Diagnostics** viewer. Otherwise, the block displays a warning message in the MATLAB command window and continues the simulation.

## Output Assertion Signal

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

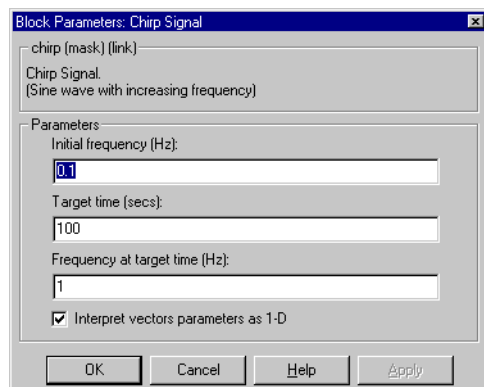
**Description** The Chirp Signal block generates a sine wave whose frequency increases at a linear rate with time. You can use this block for spectral analysis of nonlinear systems. The block generates a scalar or vector output.



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 the **Interpret vector parameters as 1-D** option is selected. If this option is selected and the parameters are row or column vectors, the block outputs a vector (1-D array) signal.

**Data Type Support** A 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	Of parameters
	Dimensionalized	Yes
	Zero Crossing	No

# Clock

---

**Purpose** Display and provide the simulation time

**Library** Sources

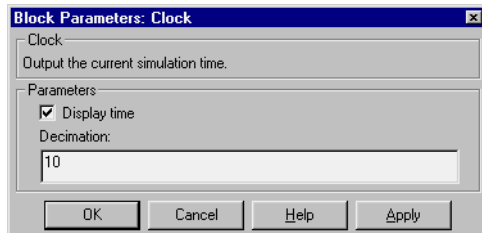
**Description** The Clock block outputs the current simulation time at each simulation step. This block is useful for other blocks that need the simulation time.



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

**Data Type Support** A 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 block 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. Note that if this parameter is not zero, the simulation must use a fixed-step solver to ensure accurate clock updates.

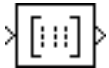
<b>Characteristics</b>	Sample Time	Continuous
	Scalar Expansion	N/A
	Dimensionalized	No
	Zero Crossing	No

# Combinatorial Logic

**Purpose** Implement a truth table

**Library** Math 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

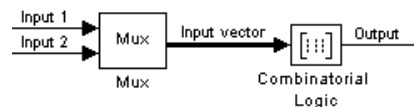
$$\text{number of rows} = 2^{\text{(number of inputs)}}$$

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,

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

## Example of Two-Input AND Function

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



The following table indicates the combination of inputs that generate each output. The input signal labeled “Input 1” corresponds to the column in the table labeled Input 1. Similarly, the input signal “Input 2” corresponds to the

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

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

$$(2^1 * 1 + 1)$$

The output value is 0.

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

## Example of Circuit

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

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

# Combinatorial Logic

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

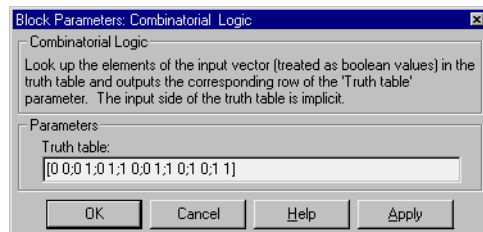
To implement this adder with the Combinatorial Logic block, you enter the 8-by-2 matrix formed by columns **c'** and **s** as the **Truth table** parameter.

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 Simulink's Boolean logic signals option (see "Enabling Strict Boolean Type Checking" in *Using Simulink*). If this option is enabled, the block accepts real signals of type `boolean` or `double`. The truth table can have Boolean values (0 or 1) of any data type. If the table contains non-Boolean values, the table's data type must be `double`. The type of the output is the same as that of the input except that the block outputs `double` if the input is `boolean` and the truth table contains non-Boolean values. If Boolean compatibility mode is disabled, the Combinatorial Logic block accepts only signals of type `boolean`. The block outputs `double` if the truth table contains non-Boolean values of type `double`. Otherwise, the output is `boolean`.

## Parameters and Dialog Box



### Truth table

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



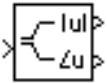
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	Dimensionalized	Yes; the output width is the number of columns of the <b>Truth table</b> parameter
	Zero Crossing	No

# Complex to Magnitude-Angle

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

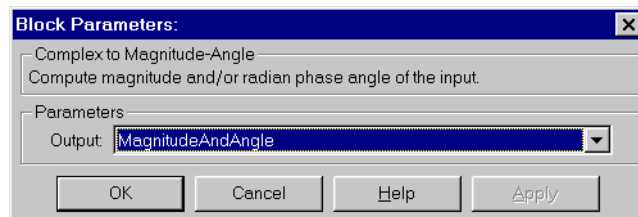
**Library** Math Operations

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



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

## Parameters and Dialog Box



## Output

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

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

**Purpose** Output the real and imaginary parts of a complex input signal

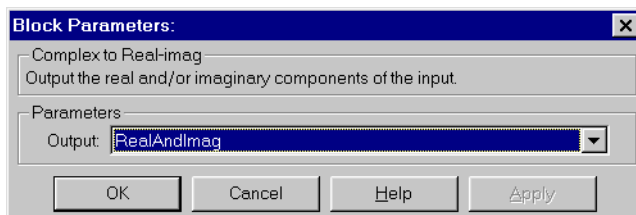
**Library** Math Operations

**Description** The Complex to Real-Imag block accepts a complex-valued signal of any data type, including fixed-point data types, except `int64` and `uint64`. 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.

**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).

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

# Configurable Subsystem

---

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

**Library** Ports & Subsystems

**Description** A 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.

- 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:

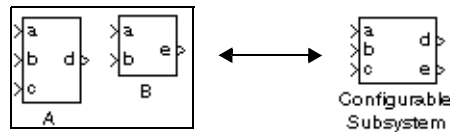
- Map each uniquely named input/output port in the library to a separate input/output port of the same name on the Configurable Subsystem block.
- Map all identically named input/output ports in the library to the same input/output ports on the Configurable Subsystem block.
- Terminate any input/output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

# Configurable Subsystem

---

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.

---

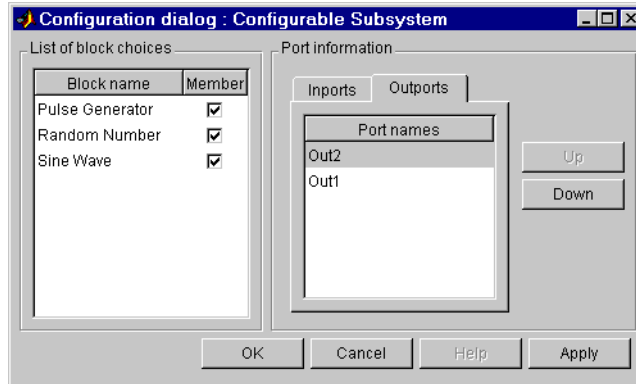
**Note** A Configurable Subsystem block does not provide ports that correspond to non-I/O ports, such as the trigger and enable ports on triggered and enabled subsystems. Thus, you cannot use a Configurable Subsystem block directly to represent blocks that have such ports. You can do so indirectly, however,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

---

## Data Type Support

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

## Parameters and Dialog Box



### List of block choices

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

### Port information

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

## Characteristics

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

# Constant

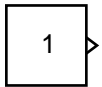
## Purpose

Generate a constant value

## Library

Simulink Sources and Fixed-Point Blockset Sources

## Description



Constant



Constant

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

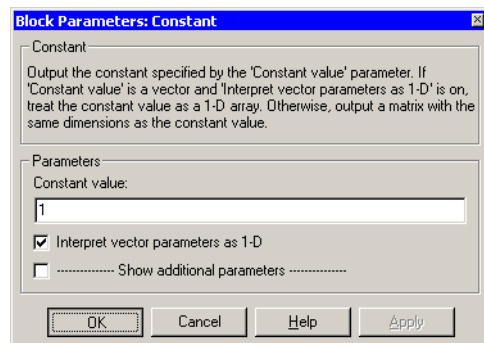
The output of the block has the same dimensions and elements of the **Constant value** parameter. If you specify a vector for this parameter, and you want the block to interpret it as 1-D, select the **Interpret vector parameters as 1-D** parameter.

When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to “Block Parameters” in the Fixed-Point Blockset documentation.

## Data Type Support

By default, a Constant block outputs a signal whose data type and complexity is the same as that of the block’s **Constant value** parameter. However, you can specify the output to be any supported data type except int64 and uint64.

## Parameters and Dialog Box



### Constant value

Constant value output by the block. It can be a scalar, vector, or matrix.

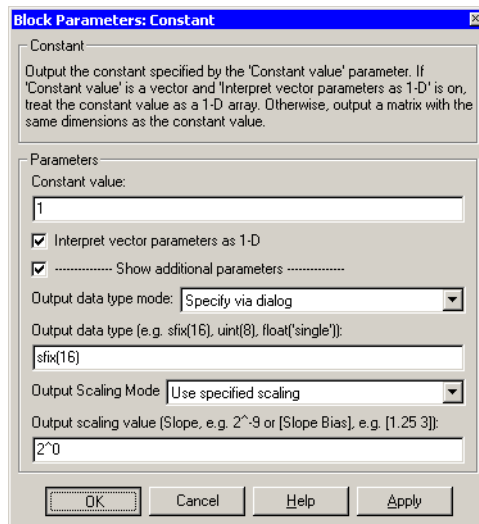


## Interpret vector parameters as 1-D

If selected, a vector specified for the **Constant value** parameter results in a 1-D signal.

## Show implementation details

If selected, additional parameters specific to implementation of the block become visible as shown.



## 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 drop-down list. Lastly, if you choose *Specify via dialog*, the **Output data type**, **Output Scaling Mode**, and **Output scaling value** parameters become visible.

## Output data type

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

# Constant

---

## 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 Specify via dialog is selected for the **Output data type mode** parameter.

## Output scaling value

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

## Conversions and Operations

The **Constant value** parameter is converted from its data type to the specified output data type offline using round-to-nearest and saturation. Refer to “Parameter Conversions” in the Fixed-Point Blockset documentation for more information about parameter conversions.

## Characteristics

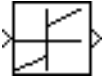
Dimensionalized	Yes
Direct Feedthrough	No
Sample Time	Constant
Scalar Expansion	No
Zero Crossing	No

# 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



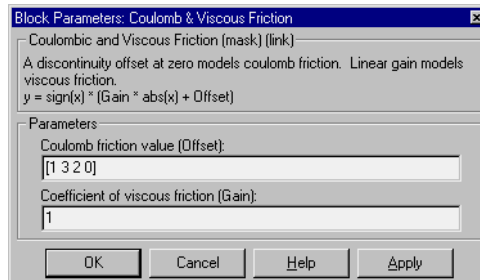
$$y = \text{sign}(u) * (\text{Gain} * \text{abs}(u) + \text{Offset})$$

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** A 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.

# Coulomb and Viscous Friction

---

<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

**Purpose** Define a data store

**Library** Signal Routing

**Description**



Data Store  
Memory

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:

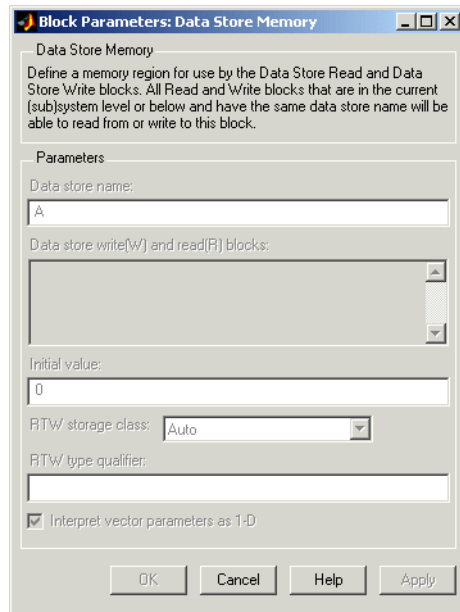
- If the Data Store Memory block is in the *top-level system*, the data store can be accessed by Data Store Read and Data Store Write blocks located anywhere in the model.
- If the Data Store Memory block is in a *subsystem*, the data store can be accessed by Data Store Read and Data Store Write blocks 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** A Data Store Memory block stores real or complex signals of any data type, including fixed-point data types, except int64 and uint64.

# Data Store Memory

## Parameters and Dialog Box



### 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.

### Data store write(W) and read(R) 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.

## **RTW storage class**

Specify the RTW storage class of the data store. For more information, refer to the Real-Time Workshop documentation.

## **RTW type qualifier**

Specify a RTW type qualifier for the data store. This parameter is only enabled if a value other than auto is selected for the **RTW storage class** parameter. For more information, refer to the Real-Time Workshop documentation.

## **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.

<b>Characteristics</b>	Dimensionalized	Yes
	Sample Time	N/A

**See Also** 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 2-79.

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

---

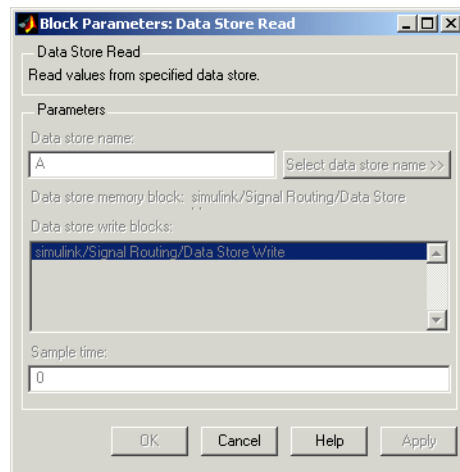
**Note** Be careful when setting an execution priority on a Data Store Read block. Make sure that the block reads from the data store after the store is updated by any Data Store Write blocks that write to the store in the same time step.

---

## Data Type Support

A Data Store Read block can output a real or complex signal of any data type, including fixed-point data types, except int64 and uint64.

## Parameters and Dialog Box





**Data store name**

Specify the name of the data store from which this block reads data.

**Data store memory block**

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

**Data store write blocks**

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

**Sample time**

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

<b>Characteristics</b>	Dimensionalized	Yes
	Sample Time	Continuous or discrete

**See Also** Data Store Memory, Data Store Write

# Data Store Write

**Purpose** Write data to a data store

**Library** Signal Routing

## Description



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

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

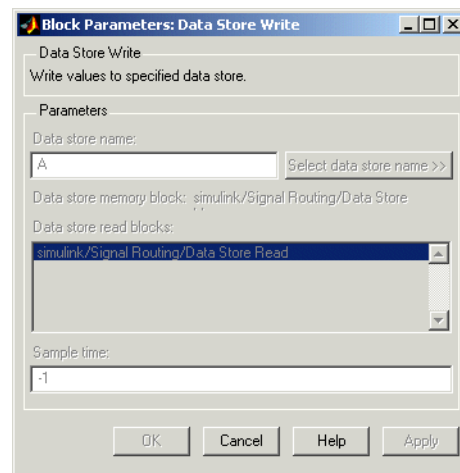
The data store to which this block writes is determined by the location of the Data Store Memory block that defines the data store. For more information, see “Data Store Memory” on page 2-79. The size of the data store is set by the Data Store Memory block that defines and initializes the data store. 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

A Data Store Write block accepts a real or complex signal of any data type, including fixed-point data types, except int64 and uint64.

## Parameters and Dialog Box



**Data store name**

Specify the name of the data store to which this block writes data.

**Data store memory block**

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

**Data store read blocks**

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

**Sample time**

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

**Characteristics**

Dimensionalized	Yes
Sample Time	Continuous or discrete

**See Also**

Data Store Memory, Data Store Read

# Data Type Conversion

---

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

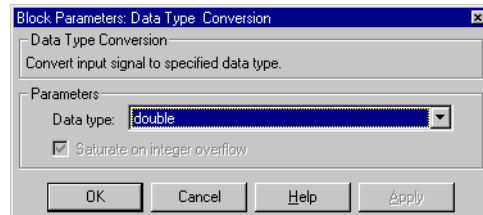
**Library** Signal Attributes

**Description** The Data Type Conversion block converts an input signal to the data type specified by the block's **Data type** parameter. The input can be any real- or complex-valued signal. If the input is real, the output is real. If the input is complex, the output is complex.



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

## Parameters and Dialog Box



### Data type

Specifies the type to which to convert the input signal. The auto option converts the input signal to the type required by the input port to which the Data Type Conversion block's output port is connected.

### Saturate on integer overflow

This parameter is enabled only for integer output. If selected, this option causes the output of the Data Type Conversion block to saturate on integer overflow. In particular, if the output data type is an integer type, the block output is the maximum value that can be represented by the output type or the converted output, whichever is smaller in the absolute sense. If the option is not selected, Simulink takes the action specified by the **Data overflow** event option on the **Diagnostics** page of the **Simulation Parameters** dialog box (see "The Diagnostics Pane" in Using Simulink.).

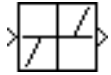
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	N/A
	Dimensionalized	Yes
	Zero Crossing	No

# Dead Zone

**Purpose** Provide a region of zero output

**Library** Discontinuities

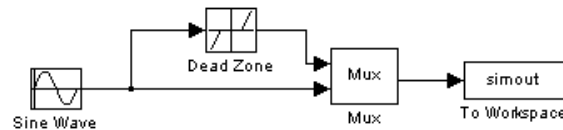
## Description



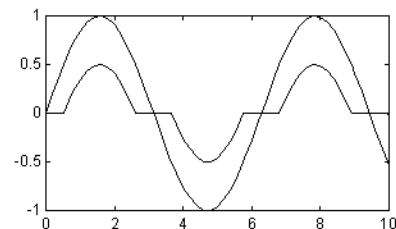
The Dead Zone block generates zero output within a specified region, called its dead zone. The lower and upper limits of the dead zone are specified as the **Start of dead zone** and **End of dead zone** parameters. The block output depends on the input and dead zone:

- 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.



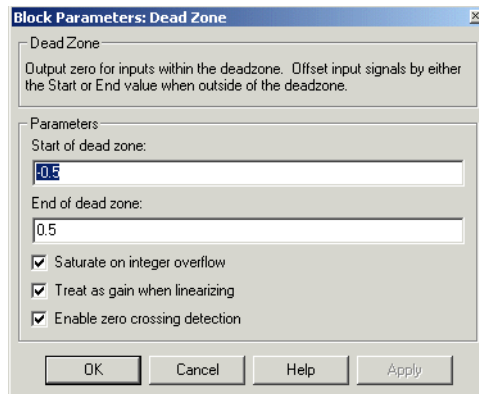
This plot shows the effect of the Dead Zone block on the sine wave. While the input (the sine wave) is between -0.5 and 0.5, the output is zero.



## Data Type Support

A Dead Zone block accepts and outputs a real signal of any data type except `int64` and `uint64`.

## Parameters and Dialog Box



### Start of dead zone

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

### End of dead zone

The upper limit of the dead zone. The default is 0.5.

### Treat as gain when linearizing

Simulink's linearization commands treat this block as a gain in state space. Selecting this option causes 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.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Of parameters
	Dimensionalized	Yes

# Demux

---

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

**Library** Signal Routing

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



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

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

## Specifying the Number of Outputs in Vector Mode

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



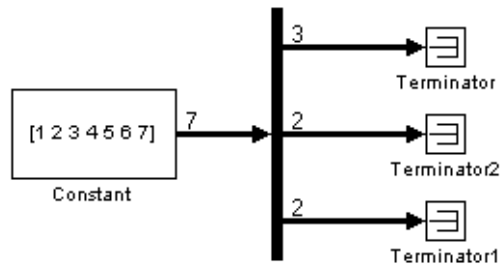
The following table summarizes how the block determines the outputs for an input vector of width  $n$ .

Parameter Value	Block outputs...	Comments
$p = n$	$p$ scalar signals	For example, if the input is a three-element vector and you specify three outputs, the block outputs three scalar signals.
$p > n$	Error	
$p < n$ $n \bmod p = 0$	$p$ 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.

# Demux

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	

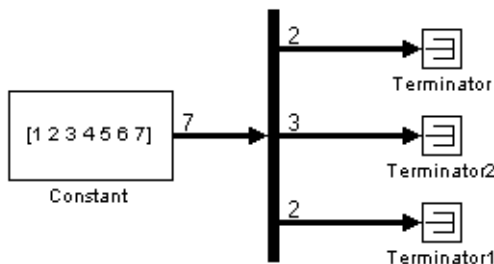
Note that you can specify the number of outputs as fewer than the number of input elements, in which case the block distributes the elements as evenly as possible over the outputs as illustrated in the following example.



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.

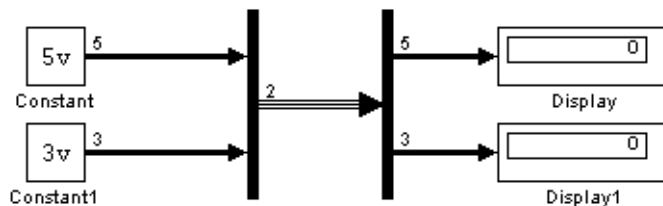


### Specifying the Number of Outputs in Bus Selection Mode

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.



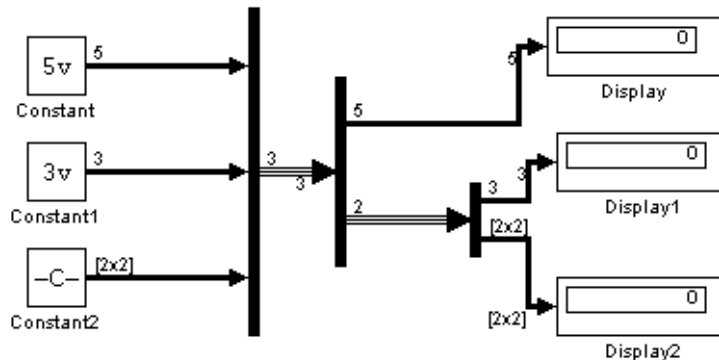
- Vector each of whose elements specifies the number of signals to output on the corresponding port

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

# Demux

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.

---

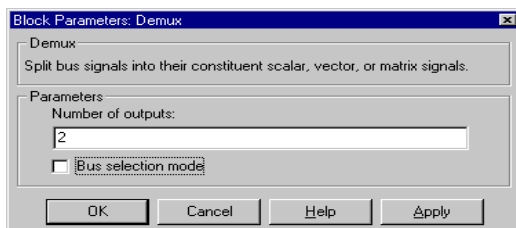
**Note** Simulink hides the name of a Demux block when you copy it from the Simulink library to a model.

---

## Data Type Support

A Demux block accepts and outputs complex or real signals of any data type, including fixed-point data types, except `int64` and `uint64`.

## Parameters and Dialog Box



### **Number of outputs**

The number and dimensions of outputs.

### **Bus selection mode**

Enable bus selection mode.

# Derivative

---

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

**Library** Continuous

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



$$\frac{\Delta u}{\Delta t}$$

where  $\Delta u$  is the change in input value and  $\Delta t$  is the change in time since the previous simulation time step. The block accepts one input and generates one output. The value of the input signal before the start of the simulation is assumed to be zero. 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

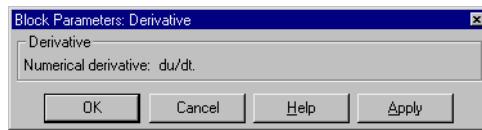
$$\frac{Y(z)}{u(z)} = \frac{1 - z^{-1}}{\Delta t} = \frac{z - 1}{\Delta t \cdot z}$$

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 Using Simulink.

## Data Type Support

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

## Dialog Box



## Characteristics

Direct Feedthrough	Yes
Sample Time	Continuous
Scalar Expansion	N/A
States	0
Dimensionalized	Yes
Zero Crossing	No

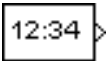
# Digital Clock

---

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

**Library** Sources

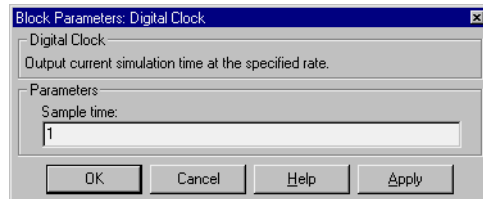
**Description** The Digital Clock block outputs the simulation time only at the specified sampling interval. At other times, the output is held at the previous value.



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** A 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	Discrete
	Scalar Expansion	No
	Dimensionalized	No
	Zero Crossing	No



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

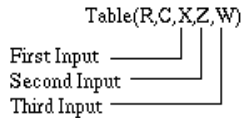
**Library** Look-Up Tables

**Description**



The Direct Look-Up 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.

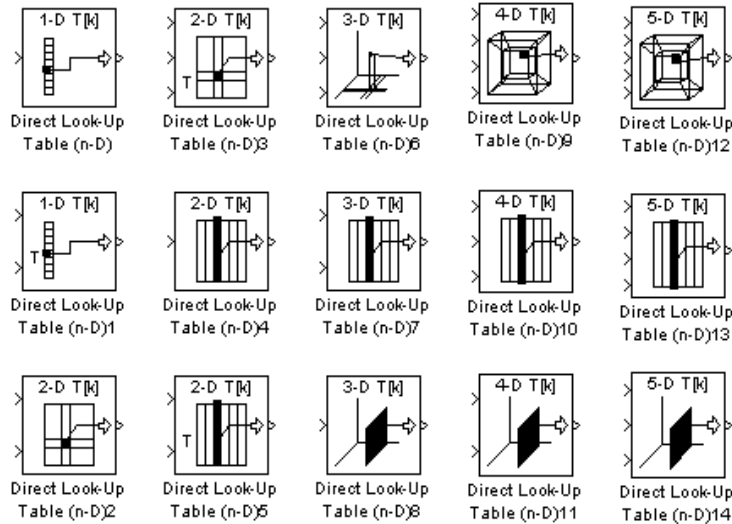
You define a set of output values as the **Table data** parameter. You specify what the output shape is: a scalar, a vector, or a 2-D matrix. The first input specifies the zero-based index to the first dimension higher than the number of dimensions in the output, the second input specifies the index 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 Look-Up Table (n-D)

This figure shows the set of all the different icons that the Direct Look-Up 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

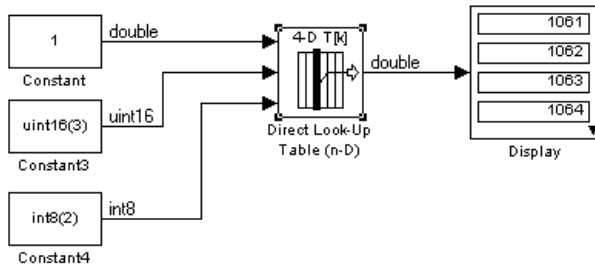
```
Invalid input value: "Clip and Warn"  
Output shape:      "Vector"  
Table data:        int16(a)
```

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

# Direct Look-Up Table (n-D)

```
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
```

# Direct Look-Up Table (n-D)

---

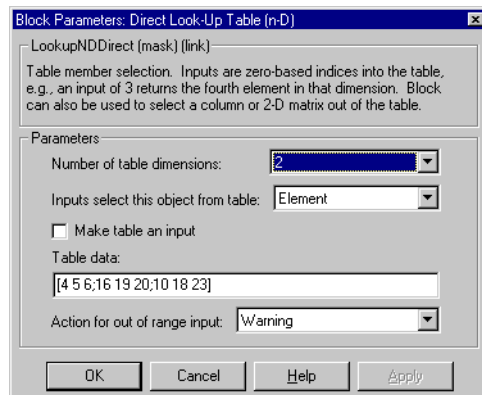
1075  
1076  
1077  
1078  
1079  
1080

## Data Type Support

The Direct Look-Up Table (n-D) block accepts mixed-type signals of type double, single, int8, uint8, int16, uint16, int32, and uint32. 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.

## 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,” following).

## 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's 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 Look-Up 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 MATLAB's 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 the Real-Time Workshop to generate clipping code with no code included to generate warnings. Code generated for the 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

# Direct Look-Up Table (n-D)

---

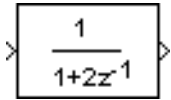
cases. This assumption helps the code generated by the Real-Time Workshop to be highly efficient.

<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** 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 2-116 for more information about coefficients.

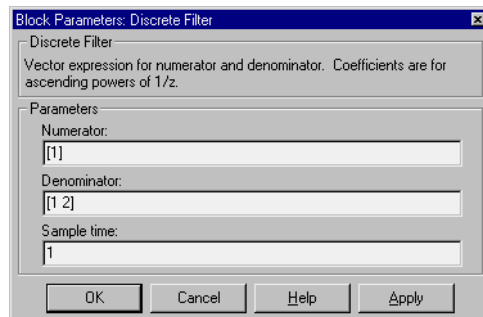
The Discrete Filter block represents the method often used by signal processing engineers, who describe digital filters using polynomials in  $z^{-1}$  (the delay operator). The Discrete Transfer Fcn block represents the method often used by control engineers, who represent a discrete system as polynomials 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 icon 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 2-355.

**Data Type Support**

A 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 ].

# Discrete Filter

---

## 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.

## Characteristics

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



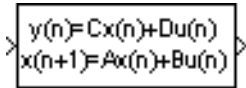
**Purpose** Implement a discrete state-space system

**Library** Discrete

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

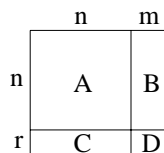
$$x(n+1) = Ax(n) + Bu(n)$$

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



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

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



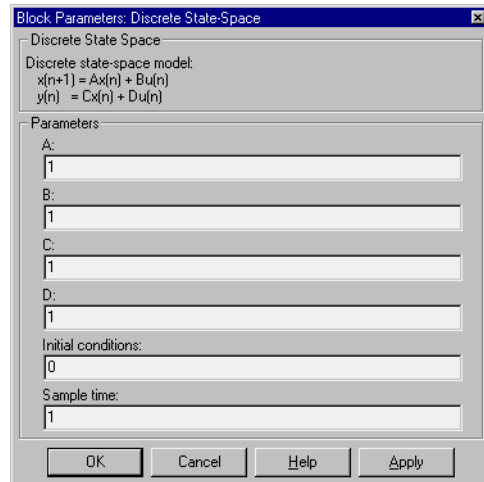
The block accepts one input and generates one output. The input vector width is determined by the number of columns in the  $B$  and  $D$  matrices. The output vector width is determined by the number of rows in the  $C$  and  $D$  matrices.

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

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

# Discrete State-Space

## 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.

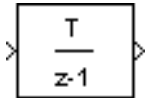
## Characteristics

Direct Feedthrough	Only if $D \neq 0$
Sample Time	Discrete
Scalar Expansion	Of the initial conditions
States	Determined by the size of $A$
Dimensionalized	Yes
Zero Crossing	No

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

**Library** Discrete

**Description** The Discrete-Time Integrator block can be used 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 Methods

The block can integrate using these methods: Forward Euler, Backward Euler, and Trapezoidal. For a given step  $k$ , Simulink updates  $y(k)$  and  $x(k+1)$ .  $T$  is the sampling period (delta  $T$  in the case of triggered sampling time). Values are clipped according to upper or lower limits. In all cases,  $y(0)=x(0)=IC$  (clipped if necessary), i.e., the initial output of the block is always the initial condition.

- 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  $k$  is

$$y(k) = y(k-1) + T * u(k-1)$$

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

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

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

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

# Discrete-Time Integrator

---

$$x(k+1) = x(k) + T*u(k) \text{ (clip if necessary)}$$

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

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

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

$$y(k) = y(k-1) + T * u(k)$$

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

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

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

$$\begin{aligned} \text{Step } k: y(k) &= x(k) + T*u(k) \\ x(k+1) &= y(k) \end{aligned}$$

With this method, input port 1 has direct feedthrough.

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

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

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

$$x(k) = y(k-1) + T/2 * u(k-1)$$

The block uses the following steps to compute its output:

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

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

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

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

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

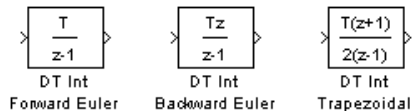
$$\begin{aligned} \text{Step 0:} \quad y(0) &= x(0) = \text{IC (clipped if necessary)} \\ x(1) &= y(0) \end{aligned}$$

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

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

With this method, input port 1 has direct feedthrough.

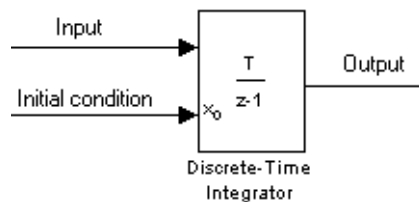
The block icon reflects the selected integration method, as this figure shows.



## 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.



# Discrete-Time Integrator

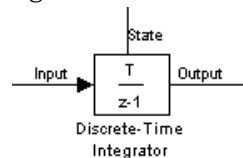
## Using the State Port

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

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

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

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

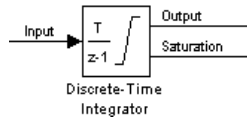


## Limiting the Integral

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

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

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



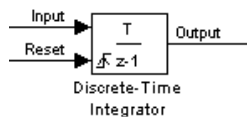
The signal has one of three values:

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

When the **Limit output** option is selected, the block has three zero crossings: one to detect when it enters the upper saturation limit, one to detect when it enters the lower saturation limit, and one to detect when it leaves saturation.

## Resetting the State

The block can reset its state to the specified initial condition, based on an external signal. To cause the block to reset its state, select one of the **External** reset choices. A trigger port appears below the block's input port and indicates the trigger type, as shown in this figure.



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

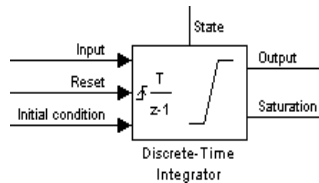
The reset port has direct feedthrough. If the block output is fed back into this port, either directly or through a series of blocks with direct feedthrough, an

# Discrete-Time Integrator

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

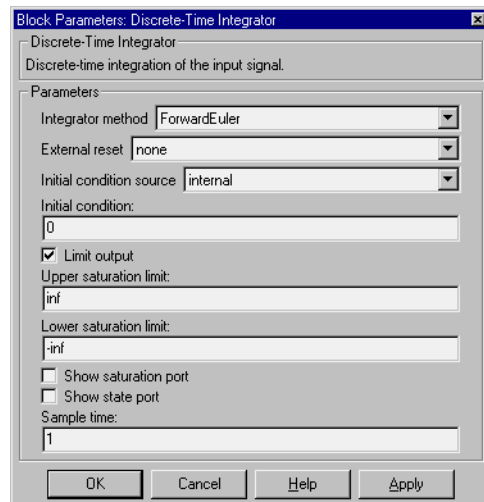
When all options are selected, the icon looks like this.



## Data Type Support

A Discrete-Time Integrator block accepts and outputs real signals of type double.

## Parameters and Dialog Box



### Integrator method

The integration method. The default is ForwardEuler.

### External reset

Resets the states to their initial conditions when a trigger event (rising, falling, either, 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 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. 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.

## Sample time

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

<b>Characteristics</b>	Direct Feedthrough	Yes, of the reset and external initial condition source ports
	Sample Time	Discrete
	Scalar Expansion	Of parameters
	States	Inherited from driving block and parameter
	Dimensionalized	Yes
	Zero Crossing	One for detecting reset, one each to detect upper and lower saturation limits, one when leaving saturation

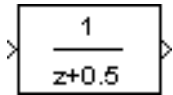
# Discrete Transfer Fcn

---

**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.

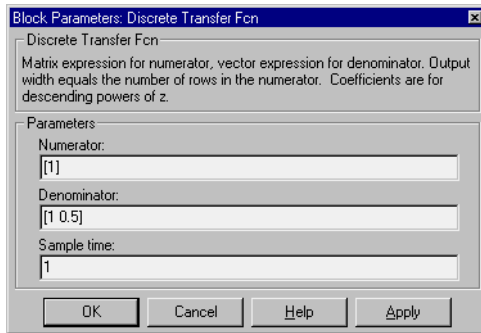
The Discrete Transfer Fcn block represents the method typically used by control engineers, representing discrete systems as polynomials in  $z$ . The Discrete Filter block represents the method typically used by signal processing engineers, who describe digital filters using polynomials in  $z^{-1}$  (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 2-355 for more information.

## Data Type Support

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

## Parameters and Dialog Box



### Numerator

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

### Denominator

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

### Sample time

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

## Characteristics

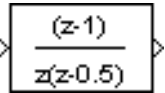
Direct Feedthrough	Only if the lengths of the <b>Numerator</b> and <b>Denominator</b> parameters are equal
Sample Time	Discrete
Scalar Expansion	No
States	Length of <b>Denominator</b> parameter -1
Dimensionalized	No
Zero Crossing	No

# Discrete Zero-Pole

**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



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

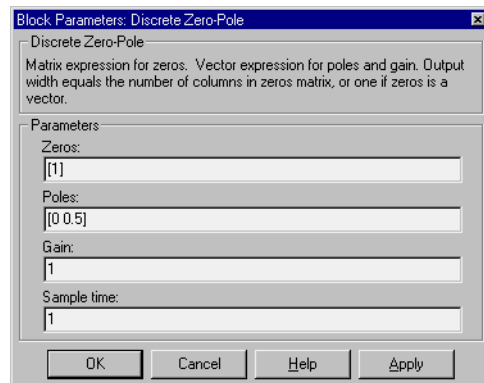
where  $Z$  represents the zeros vector,  $P$  the poles vector, and  $K$  the gain. The number of poles must be greater than or equal to the number of zeros ( $n \geq m$ ). If the poles and zeros are complex, they must be complex conjugate pairs.

The block icon displays the transfer function depending on how the parameters are specified. See Zero-Pole on page 2-387 for more information.

## Data Type Support

A Discrete Zero-Pole block accepts and outputs real signals of type double.

## Parameters and Dialog Box



### Zeros

The matrix of zeros. The default is [1].

**Poles**

The vector of poles. The default is [0 0.5].

**Gain**

The gain. The default is 1.

**Sample time**

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

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

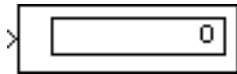
# Display

---

**Purpose** Show the value of the input

**Library** Sinks

**Description** The Display block shows the value of its input.



You can control the display format by selecting a **Format** choice:

- short, which displays a 5-digit scaled value with fixed decimal point
- long, which displays a 15-digit scaled value with fixed decimal point
- short\_e, which displays a 5-digit value with a floating decimal point
- long\_e, which displays a 16-digit value with a floating decimal point
- bank, which displays a value in fixed dollars and cents format (but with no \$ or commas)

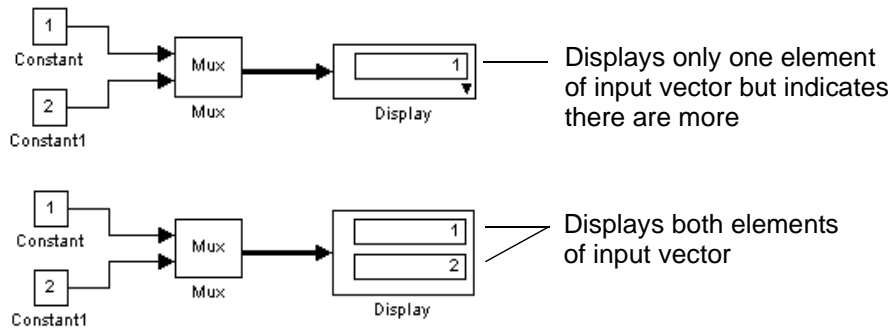
To use the block as a floating display, select the **Floating display** check box. The block's input port disappears and the block displays the value of the signal on a selected line. If you select the **Floating display** option, you must turn off Simulink's signal storage reuse feature. See "Signal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

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 nth 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

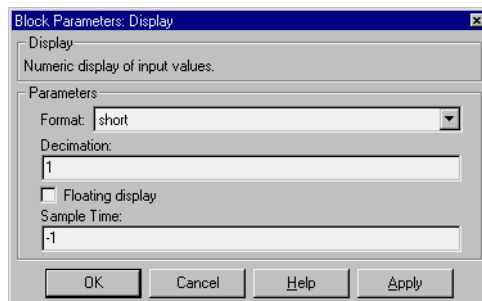
top model shows the block before it is resized; notice the black triangle. The bottom model shows the resized block displaying both input elements.



## Data Type Support

A Display block accepts and outputs real or complex signals of any data type, including fixed-point data types, except int64 and uint64.

## Parameters and Dialog Box



### Format

The format of the data displayed. The default is short.

### Decimation

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.

# Display

---

## Sample time

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

## Characteristics

Sample Time	Inherited from driving block
Dimensionalized	Yes



**Purpose** Create text that documents the model and save the text with the model

**Library** Model-Wide Utilities

**Description**



The DocBlock allows you to create and edit text that documents a model and save that text with the model. Double-clicking an instance of 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 **Preferences** 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.

**Data Type Support** Not applicable.

**Dialog Box** The DocBlock does not have a parameter dialog box.

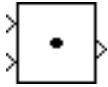
**Characteristics** Not applicable

# 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



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

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 of data type `double`. The signal type (complex or real) of the output depends on the signal types of the inputs.

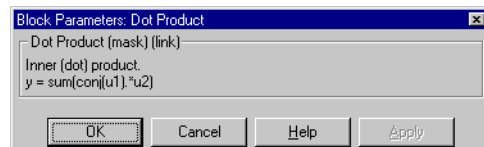
Input 1	Input 2	Output
real	real	real
real	complex	complex
complex	real	complex
complex	complex	complex

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

## Data Type Support

A Dot Product block accepts and outputs signals of type `double`.

## Dialog Box



<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	No
	States	0
	Dimensionalized	Yes
	Zero Crossing	No

# Enable

---

**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:

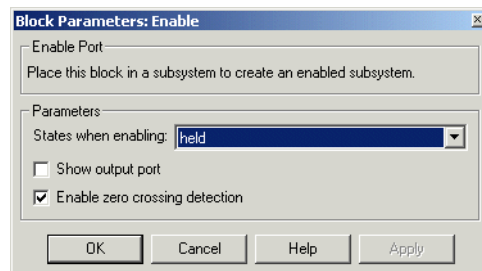
- reset resets the states to their initial conditions (zero if not defined).
- held holds the states at their previous values.

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 except int64 and uint64. See “Creating Conditionally Executed Subsystems” in the online Simulink help for more information about enabled subsystems.

## Parameters and Dialog Box



### 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.

<b>Characteristics</b>	Sample Time	Determined by the signal at the enable port
	Dimensionalized	Yes

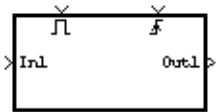
# Enabled and Triggered Subsystem

---

**Purpose** Represent a subsystem whose execution is enabled and triggered by external input

**Library** Ports & Subsystems

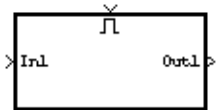
**Description** This block is a Subsystem block that is preconfigured to serve as the starting point for creating an enabled and triggered subsystem. For more information, see “Triggered and Enabled Subsystem” in the online Simulink help.



**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 *Using Simulink*.



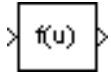
# Fcn

---

**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).

The rules of precedence obey the C language standards:

- 1 ( )
- 2 + - (unary)
- 3 pow (exponentiation)
- 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.

---

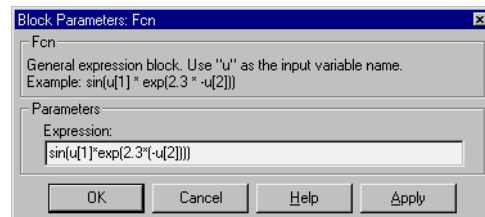
**Note** Simulink does not allow you to change the value of the block's **Expression** parameter while running a model in accelerated mode (see “The Simulink Accelerator”). Furthermore, Simulink does not update the value of the Fcn expression to reflect changes in the workspace while running in accelerated mode.

---

## Data Type Support

A 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.).

# Fcn

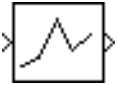
---

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

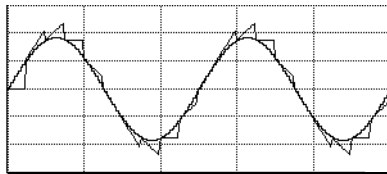
**Purpose** Implement a first-order sample-and-hold

**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.

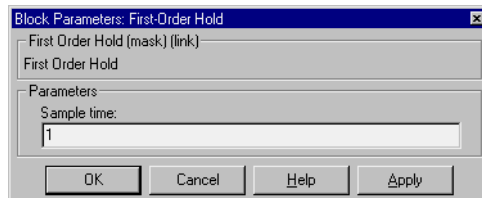


You can see the difference between the Zero-Order Hold and First-Order Hold blocks by running the demo program fohdemo. This figure compares the output from a Sine Wave block and a First-Order Hold block.



**Data Type Support** A 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.

# First-Order Hold

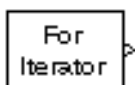
---

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Continuous
	Scalar Expansion	No
	States	1 continuous and 1 discrete per input element
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Implement a C-like for control flow statement in Simulink

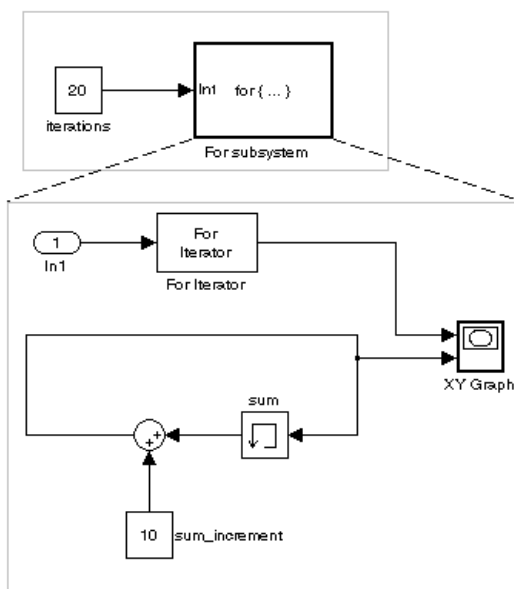
**Library** Ports & Subsystems/For Subsystem

## Description



The For Iterator block, when placed in a subsystem, implements a C-like for control flow statement in Simulink as a For subsystem. In the For subsystem, the For Iterator block has iterative control over any Simulink blocks present. For each iteration value of the For Iterator block, the accompanying blocks execute. The number of iterations is set internally for the For Iterator block or externally with data input.

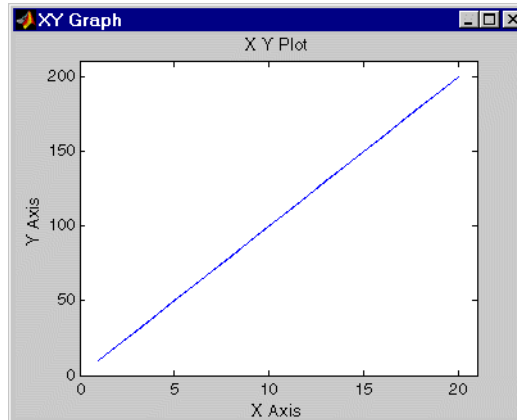
The following example shows a completed for control flow statement that increments an initial value of zero by 10 over 20 iterations.



In the preceding example, a For subsystem receives an input, which it passes to the For Iterator block inside. The For Iterator block uses this input to determine the number of times it executes the blocks of its subsystem, in this case, 20 times. Each time the blocks execute, a value of 10 is added to a sum, which is initially zero. In addition, for each time the blocks of the subsystem execute, the For Iterator block outputs a value equal to the number of times that the blocks have executed, including the current execution. This is referred

# For Iterator

to as the iterator value. This value, along with the sum value, is sent to an XY Graph block with the following result.



Points:  
(1,10)  
(2,20)  
etc.

The preceding for control flow statement example can be represented by the following pseudocode.

```
sum = 0;
iterations = 20;
sum_increment = 10;
for (i = 0; i < iterations; i++) {
    sum = sum + sum_increment;
}
```

You construct a For subsystem like the preceding example as follows:

- 1 Create a subsystem and place a For Iterator block in it.

This changes the subsystem icon to the text `for{...}`.

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

- 2 Double-click the For Iterator block to open its **Block Parameters** dialog and enter as follows:

- Specify external for the **Source of number of iterations** field.  
A data input marked N now appears on the For Iterator block. The source for this data input must be outside the For subsystem.  
In the preceding example, the source for the N port is a constant of value 20 that resides outside the For subsystem.  
If you specify internal for the **Source of number of iterations** field, the **Number of iterations** field appears for you to specify the number of iterations.
- Make sure that the **Show iteration number port** check box is selected if you want the For Iterator block to provide an optional data output port for the iteration value. This value begins at 1 for the first iteration and increments by 1 for each succeeding iteration.  
When the **Show iteration number port** check box is selected (the default), the **Output data type** field is enabled (grayed out otherwise). This allows you to select the type for the iterations number data output. The default type is int32.
- The setting of the **States when starting** field applies only to cases in which the For subsystem is called repeatedly. If you select reset (the default value) for this field, every time the For subsystem is called, its states are reset to their initial values. If you select held, the states of the For subsystem are retained between calls. In the preceding example, the For subsystem is called only once.

## Data Type Support

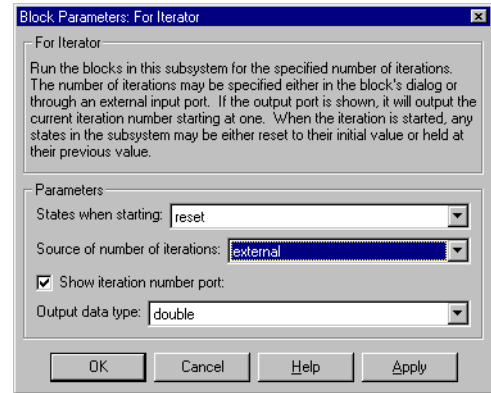
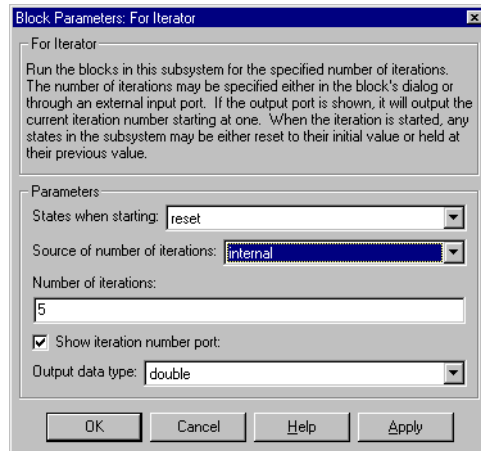
The following rules apply to the data type of the number of iterations (N) input port:

- 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 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.

# For Iterator

## Parameters and Dialog Box



### States when starting

Set this field to `reset` if you want the states of the For subsystem to be reinitialized for each iteration. Otherwise, set this field to `held` (the default) to make sure that these subsystem states retain their values from the previous iteration.

### Source of number of iterations

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 signal at the For Iterator block's N port determines the number of iterations.

### Number of iterations

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.

### Show iteration number port

If this is selected, the For Iterator block outputs its iteration value. This value starts at 1 and is incremented by 1 for each iteration.

### Output data type

Set the type for the iteration value output from the iteration number port to `double`, `int32`, `int16`, or `int8`.



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

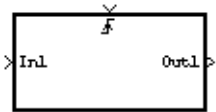
# For Iterator Subsystem

---

**Purpose** Represent a subsystem that executes repeatedly during a simulation time step

**Library** Ports & Subsystems

**Description** This 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 *Using Simulink*.



**Purpose** Accept input from a Goto block

**Library** Signal Routing

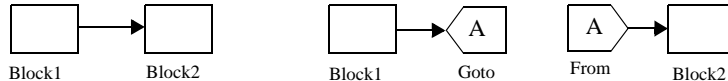
**Description**



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 *Using Simulink*.

The visibility of a Goto block tag determines the From blocks that can receive its signal. For more information, see Goto on page 2-159 and Goto Tag Visibility on page 2-162. The block icon 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.

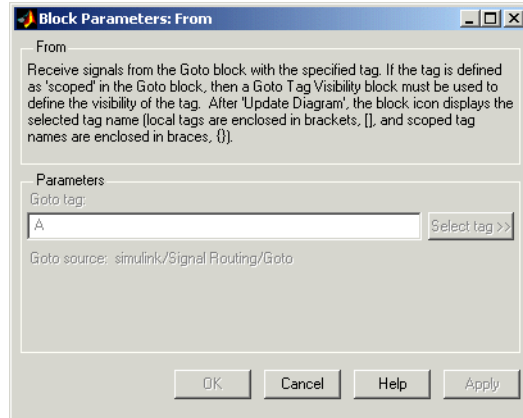
# From

---

## Data Type Support

A From block outputs real or complex signals of any data type, including fixed-point data types, except int64 and uint64.

## 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.

## Characteristics

Sample Time                      Inherited from block driving the Goto block

Dimensionalized                  Yes

**Purpose** Read data from a file

**Library** Sources

**Description** The From File block outputs data read from a file. The block icon displays the pathname of the file supplying the data.



The 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 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 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 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.

# From File

---

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

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

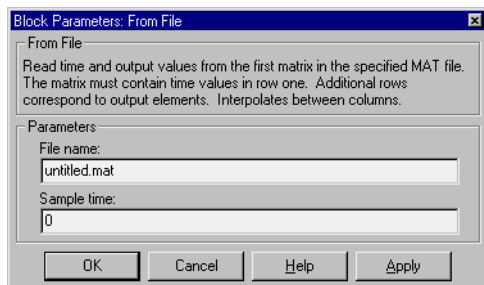
The From File block can read data written by a To File block without any modifications. To read data written by a To Workspace block and saved to a file:

- 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 **Workspace I/O** page of the **Simulation Parameters** dialog box. See “The WorkspaceF I/O Pane” for more information.
- The form of the data as it is written to the workspace is different from the form expected by the From File block. Before saving the data to a file, transpose it. When it is read by the From File block, it will be in the correct form.

## Data Type Support

A From File block outputs real signals of type double.

## Parameters and Dialog Box



### File name

The fully qualified pathname or file name of the file that contains the data used as input. On UNIX, the pathname can start with a tilde (~) character signifying your home directory. The default file name is `untitled.mat`. If you specify an unqualified file name, Simulink assumes that the file resides in MATLAB's 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	Inherited from driven block
Scalar Expansion	No
Dimensionalized	1-D array only
Zero Crossing	No

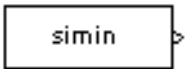
# From Workspace

---

**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 the **Interpolate data** option is selected, 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.

<b>Form Output Option</b>	<b>Interpolate Option</b>	<b>Block Output After Final Data</b>
Extrapolate	On	Extrapolated from final data value
Extrapolate	Off	Error
SettingToZero	On	Zero
SettingToZero	Off	Zero



---

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 2-351) if the output is in structure-with-time format (see “Loading Input from the Base Workspace” for a description of these formats).

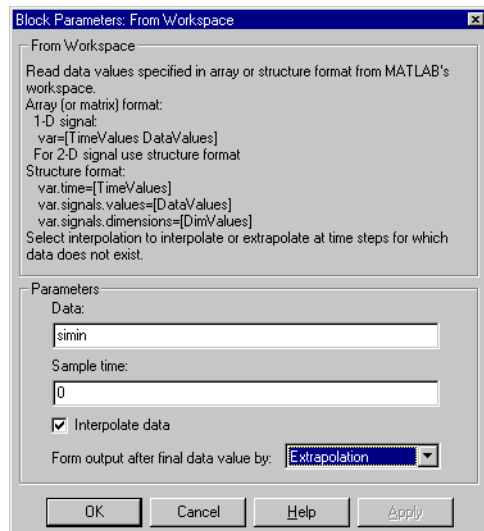
---

## Data Type Support

A From Workspace block accepts real or complex signals of any type except `int64` and `uint64` from the workspace. 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.

# From Workspace

## 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.

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

# Function-Call Generator

---

**Purpose** Execute a function-call subsystem a specified number of times at a specified rate

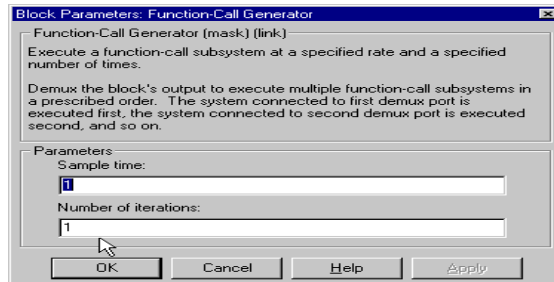
**Library** Ports & Subsystems

**Description** The Function-Call Generator block executes a function-call subsystem (for example, a Stateflow 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** A 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.

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	User-specified
	Scalar Expansion	No
	Dimensionalized	Yes
	Zero Crossing	No

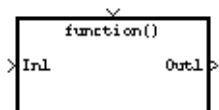
# Function-Call Subsystem

---

**Purpose** Represent a subsystem that can be invoked as a function by another block

**Library** Ports & Subsystems

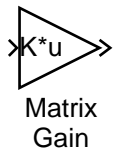
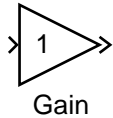
**Description** This block is a Subsystem block that is preconfigured to serve as a starting point for creating a function-call subsystem. For more information, see “Function-Call Subsystems” in *Using Simulink*.



**Purpose** Multiply the input by a constant

**Library** Simulink Math Operations and Fixed-Point Blockset Math

**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.



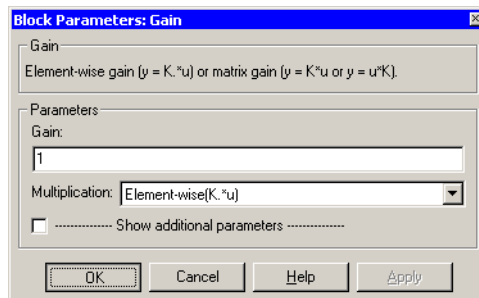
You specify the value of the gain in the **Gain** parameter. The **Multiplication** parameter lets you specify element-wise or matrix multiplication. For matrix multiplication, this parameter also lets you indicate the order of the multiplicands.

When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to “Block Parameters” in the Fixed-Point Blockset documentation.

The Matrix gain block is an implementation of the Gain block with different default settings.

**Data Type Support** The input and gain of the Gain block can be a real or complex scalar, vector, or matrix of any data type except `boolean`, `int64`, and `uint64`. If the input is real and the gain is complex, the output is complex.

## Parameters and Dialog Box



### Gain

Specify the value by which to multiply the input. The gain may be a scalar, vector, or matrix.

# Gain, Matrix Gain

---

## Multiplication

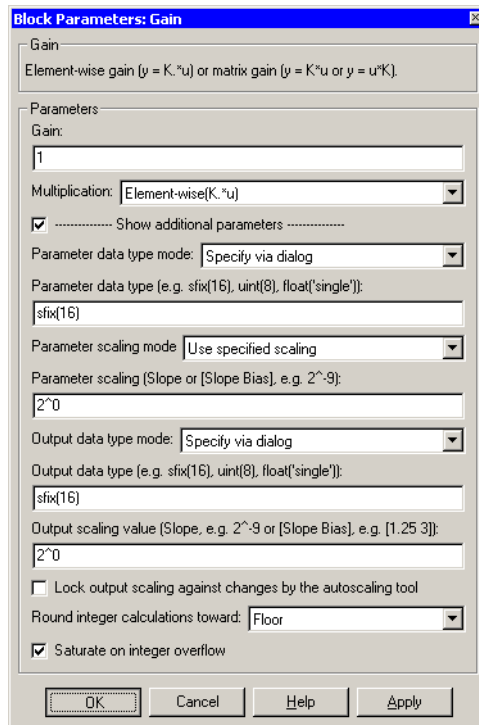
Specify the multiplication mode:

- **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.

## Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.





## 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

Set the gain data type. This parameter is only visible if **Specify via dialog** is selected 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.

# Gain, Matrix Gain

---

- **Best Precision: Element-wise**—This mode sets radix 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 radix 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 radix 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 radix point for all the elements of the gain such that the largest element has the best possible precision.

This parameter is only visible if `Specify via dialog` is selected for the **Parameter data type mode** parameter.

## Parameter scaling

Set the gain scaling using either radix point-only or [Slope Bias] scaling. This parameter is only visible if `Specify via dialog` is selected for the **Parameter data type mode** parameter, and if `Use specified scaling` is selected for the **Parameter scaling mode** parameter.

## 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 backpropagation. 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. If the **Production hardware characteristics** parameter on the **Advanced** pane of the **Simulation Parameters** dialog is set to `Unconstrained integer sizes`, Simulink chooses the output data type without regard to hardware constraints. If the parameter is set to `Microprocessor`, 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 `Unconstrained integer sizes` is

specified, the output data type is `sfixed24`. If `Microprocessor` is specified and the microprocessor supports 8-bit, 16-bit, and 32-bit words, 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 Simulink Diagnostic Viewer.

## **Output data type**

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

## **Output scaling value**

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

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

If selected, scaling of outputs is locked. This parameter is only visible if `Specify via dialog` is selected for the **Output data type mode** parameter.

## **Round integer calculations toward**

Select the rounding mode for integer output.

## **Saturate on integer overflow**

If selected, overflows saturate.

## **Conversions and Operations**

The gain is converted from doubles to the specified data type offline using round-to-nearest and saturation. Refer to “Parameter Conversions” in the Fixed-Point Blockset documentation for more information about parameter conversions. The input and gain are then multiplied, and the result is converted to the output data type using the specified rounding and overflow modes. Refer to “Rules for Arithmetic Operations” in the Fixed-Point Blockset documentation for more information about the rules that this block obeys when performing fixed-point operations.

# Gain, Matrix Gain

---

<b>Characteristics</b>	Dimensionalized	Yes
	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Of input and <b>Gain</b> parameter for Element-wise multiplication
	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 2-141. 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.

---

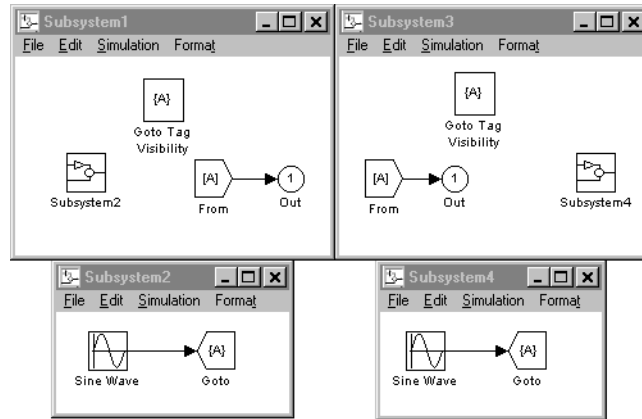
**Note** A scoped Goto block in a masked system is visible only in that subsystem and in the subsystems it contains. Simulink generates an error if you run or update a diagram that has a Goto Tag Visibility block at a higher level in the block diagram than the corresponding scoped Goto block in the masked 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

# Goto

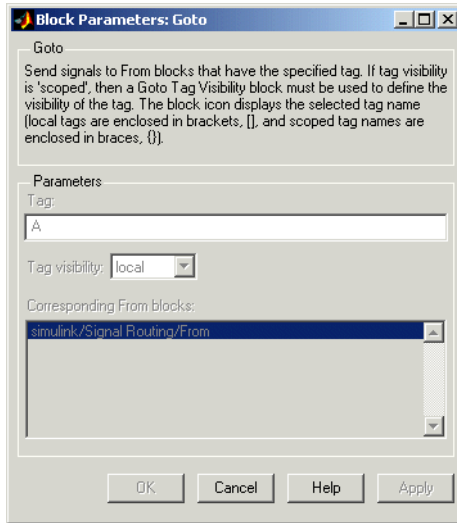
defined as scoped can be used in more than one place in the model. This example shows a model that uses two scoped tags with the same name (A).



## Data Type Support

A Goto block accepts real or complex signals of any data type, including fixed-point data types, except int64 and uint64.

## 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.

### Characteristics

Sample Time	Inherited from driving block
Dimensionalized	Yes

# Goto Tag Visibility

**Purpose** Define scope of Goto block tag

**Library** Signal Routing

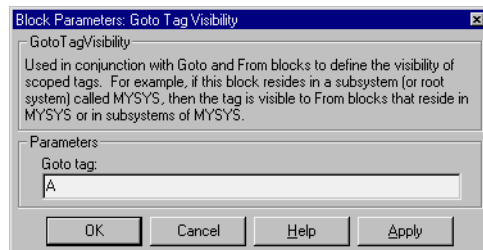
**Description** The Goto Tag Visibility block defines the accessibility of Goto block tags that have scoped visibility. The tag specified as the **Goto tag** parameter is accessible by From blocks in the same subsystem that contains the Goto Tag Visibility block and in subsystems below it in the model 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 icon 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

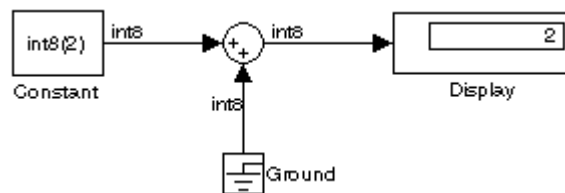
## Description



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

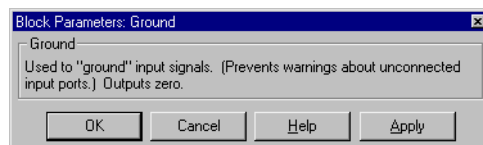
A Ground block outputs a signal of the same numeric type and data type as the port to which it is connected. For example, consider the following model.



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 fixed-point data types.

## Parameters and Dialog Box



**Characteristics** Sample Time Inherited from driven block

Dimensionalized Yes

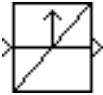
# Hit Crossing

---

**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.



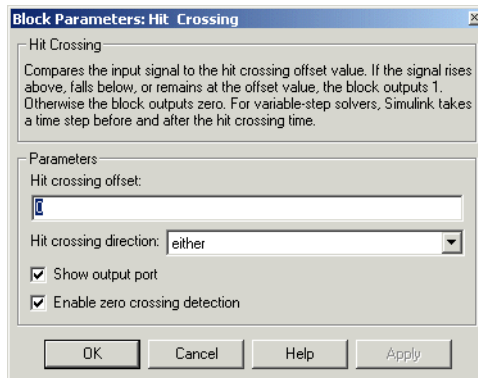
The block accepts one input of type `double`. If the **Show output port** check box is selected, the block output indicates when the crossing occurs. If the input signal is exactly the value of the offset value after the hit crossing is detected, the block continues to output a value of 1. If the input 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** A 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

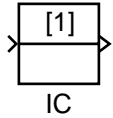
Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes
Dimensionalized	Yes

# IC

**Purpose** Set the initial value of a signal

**Library** Signal Attributes

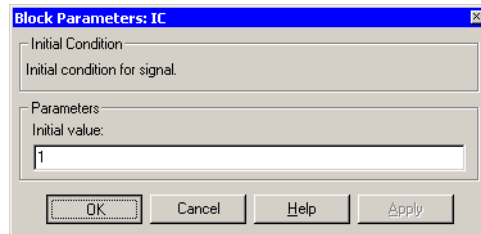
**Description** The IC block sets the value of the signal at its output port at  $t=0$ .



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 *Using Simulink*.

**Data Type Support** An IC block accepts and outputs a signal of type double.

## Dialog Box

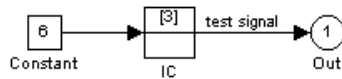


## Initial value

Specify the initial value for the input signal.

## 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.

---

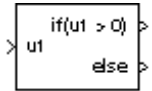
<b>Characteristics</b>	Dimensionalized	Yes
	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Of parameter only
	Zero Crossing	No

# If

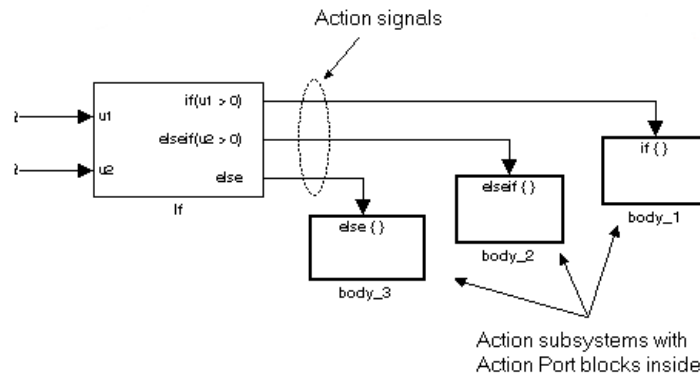
**Purpose** Implement a C-like if -else control flow statement in Simulink

**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`.

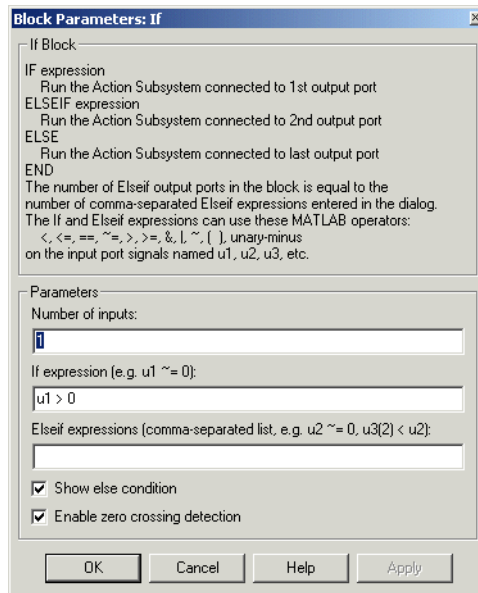
## Data Type Support

Inputs  $u_1, u_2, \dots, u_n$  can be scalar or vector of any data type, including fixed-point data types, except `int64` and `uint64`.

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 2-5.



## 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 the **Show else condition** check box is selected. 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.

# If

---

## Show else condition

If this box is selected, 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.

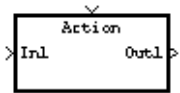
## Characteristics

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** This 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 *Using Simulink*.



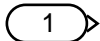
# Inport

---

**Purpose** Create an input port for a subsystem or an external input

**Library** Ports & Subsystems, Sources

**Description** Inport blocks are the links from outside a system into the system.



In1

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 block 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 **Simulation 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 **Simulation 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 *Using Simulink*.

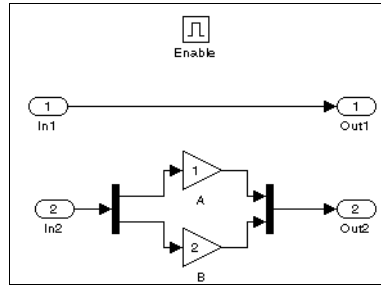
## Data and Numeric Type Support

An Inport block accepts complex or real signals of any data type including fixed-point data types. 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

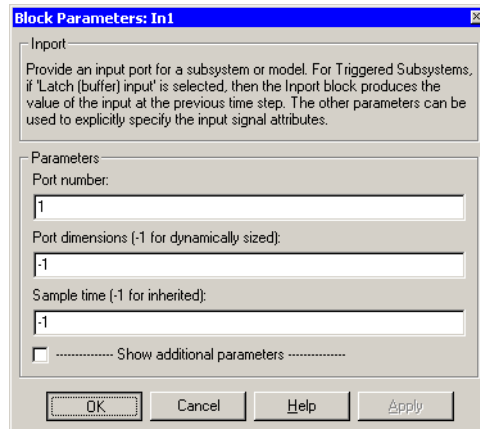
# Inport

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



### Port number

Specify the port number of the Inport 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. Valid values are

- -1—Any sample time accepted
- 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 “Specifying Sample Time” in the online documentation for more information.

## Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown:

**Block Parameters: In1**

Inport  
Provide an input port for a subsystem or model. For Triggered Subsystems, if 'Latch (buffer) input' is selected, then the Inport block produces the value of the input at the previous time step. The other parameters can be used to explicitly specify the input signal attributes.

Parameters

Port number:

Port dimensions (-1 for dynamically sized):

Sample time (-1 for inherited):

Show additional parameters

Latch (buffer) input:

Data type:

Output data type (e.g. sfix(16), uint(8), float('single')):

Output scaling value (Slope, e.g. 2^-9 or [Slope Bias], e.g. [1.25 3]):

Signal type:

Sampling mode:

Interpolate data

OK Cancel Help Apply

# Inport

---

## **Latch (buffer) input**

This field is enabled only if the Inport block resides in a triggered subsystem. If selected, the block outputs the value of the input signal at the previous time step.

## **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 Specify via dialog is selected for the **Data type** parameter.

## **Output scaling value**

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

## **Signal type**

Specify the numeric type (real or complex) of the external input. To accept any numeric 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.

## **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.

## **Characteristics**

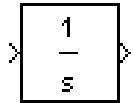
Dimensionalized	Yes
Sample Time	Inherited from driving block



**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 **Simulation 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.

# Integrator

---

The parameter dialog box also allows you to

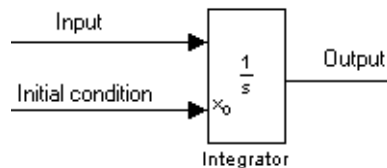
- Define upper and lower limits on the integral
- Create an input that resets the block's output (state) to its initial value, depending on how the input changes
- Create an optional state output that allows you to use the value of the block's output to trigger a block reset

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.

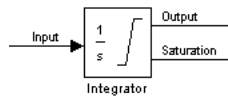
---

## 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.

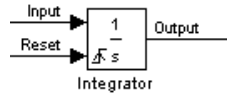
When this option is selected, the block has three zero crossings: one to detect when it enters the upper saturation limit, one to detect when it enters the lower saturation limit, and one to detect when it leaves saturation.

## 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**

# Integrator

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

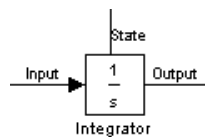


- Select **rising** to trigger the state reset when the reset signal has a rising edge.
- Select **falling** to trigger the state reset when the reset signal has a falling edge.
- Select **either** to trigger the reset when either a rising or falling signal occurs.
- Select **level1** to trigger the 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 2-183)
- Handing off a state from one enabled subsystem to another (see “Handing Off States Between Enabled Subsystems” on page 2-184)

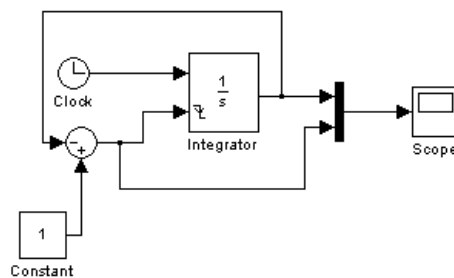
---

**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

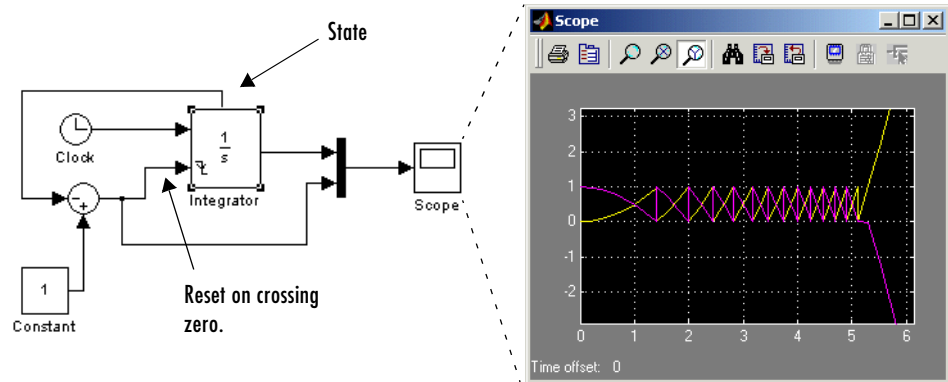
The Integrator block’s state port allows you to avoid creating algebraic loops when creating an integrator that resets itself based on the value of its output. Consider, for example, the following model.



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.

# Integrator

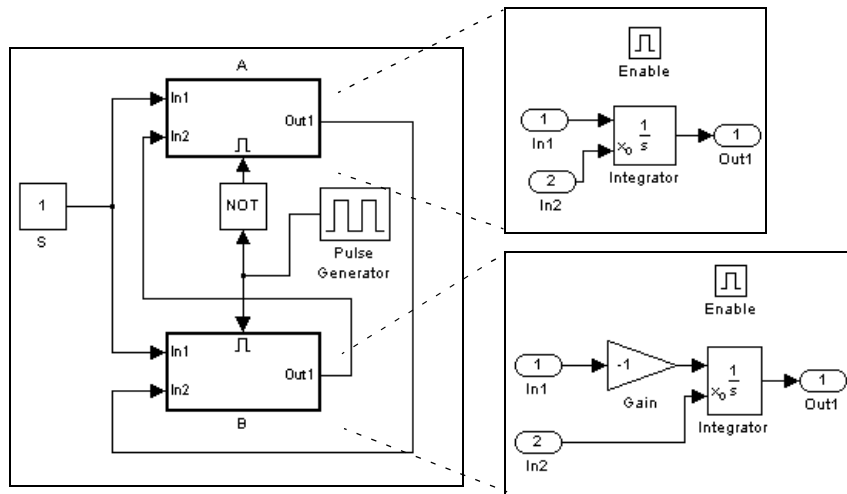
The following model uses the integrator's state port to avoid the algebraic loop.



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.

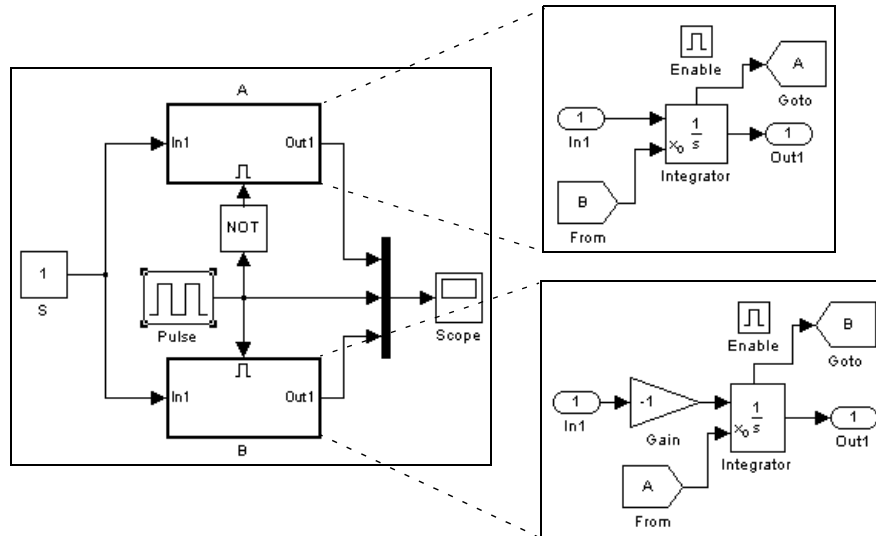


In this model, a constant input signal drives two enabled subsystems that integrate the signal. A pulse generator generates an enabling signal that causes execution to alternate between the two subsystems. The enable port of each subsystem is set to reset. This causes the subsystem to reset its integrator 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.

# Integrator

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

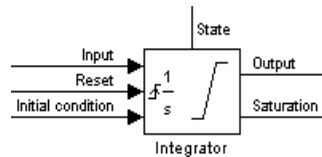
By default Simulink uses the absolute tolerance value specified in the **Simulation Parameters** dialog box (see “Error Tolerances”) to compute the output of the Integrator block. If this value does not provide sufficient error



control, specify a more appropriate value in the **Absolute tolerance** field of the Integrator block's dialog box. The value that you specify is used to compute all of the block's outputs.

## Choosing All Options

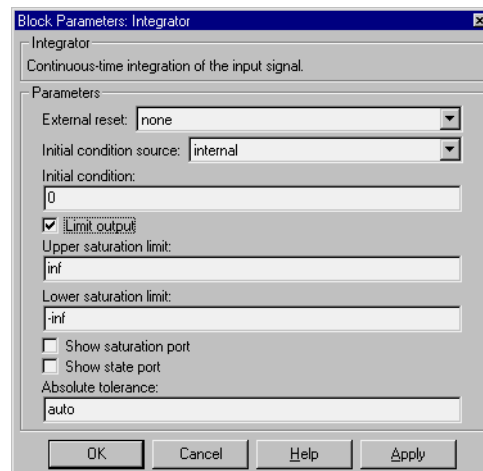
When all options are selected, the icon looks like this.



## Data Type Support

An 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).

# Integrator

---

## 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 `inf`.

## Lower saturation limit

The lower limit for the integral. The default is `-inf`.

## Show saturation port

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

## Show state port

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

## Absolute tolerance

Absolute tolerance used to compute the block's outputs. You can enter `auto` or a numeric value. If you enter `auto`, Simulink determines the absolute tolerance (see "Error Tolerances"). If you enter a numeric value, Simulink uses the specified value to compute the block's outputs. Note that a numeric value overrides the setting for the absolute tolerance in the **Simulation Parameters** dialog box.

## Characteristics

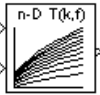
Direct Feedthrough	Yes, of the reset and external initial condition source ports
Sample Time	Continuous
Scalar Expansion	Of parameters
States	Inherited from driving block or parameter
Dimensionalized	Yes
Zero Crossing	If the <b>Limit output</b> option is selected, one for detecting reset, one each to detect upper and lower saturation limits, one when leaving saturation

# Interpolation (n-D) Using PreLook-Up

**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 PreLook-Up Index Search block

**Library** Look-Up Tables

## Description



The Interpolation (n-D) Using PreLook-Up block uses the precalculated indices and interval fractions from the PreLook-Up Index Search block to perform the equivalent operation that the Look-Up Table (n-D) performs. This combination of blocks allows multiple Interpolation (n-D) blocks to feed a set of PreLook-Up 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 PreLook-Up 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 PreLook-Up Index Search block.

The block generates output based on the input values:

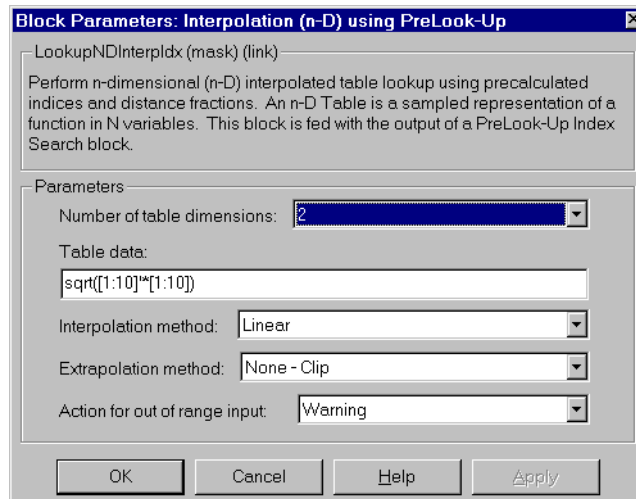
- If the inputs match breakpoint parameter values, the output is the table value at the intersection of the row, column, and higher dimensions' breakpoints.
- If the inputs do not match row and column parameter values, the block generates output by interpolating between the appropriate table values. 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.

## Data Type Support

An Interpolation (n-D) Using PreLook-Up 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.

# Interpolation (n-D) Using PreLook-Up

## 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 MATLAB's online documentation](#).

### Interpolation method

None (flat) or Linear.

# Interpolation (n-D) Using PreLook-Up

---

## Extrapolation method

None (clip) or Linear.

## Action for out of range input

None, Warning, Error.

## Characteristics

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

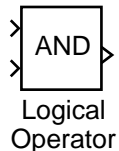
# Logical Operator

---

**Purpose** Perform the specified logical operation on the input

**Library** Simulink Math Operations and Fixed-Point Blockset Logic & Comparison

## 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 icon 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.

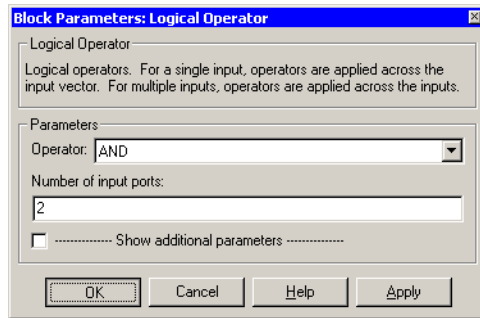
When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to “Block Parameters” in the Fixed-Point Blockset documentation.

## Data Type Support

A Logical Operator block accepts real or complex signals of any data type except `int64` and `uint64`. However, if the **Output data type mode** parameter is set to `Logical`, the input may only be `boolean` or `double`.

# Logical Operator

## Parameters and Dialog Box



### Operator

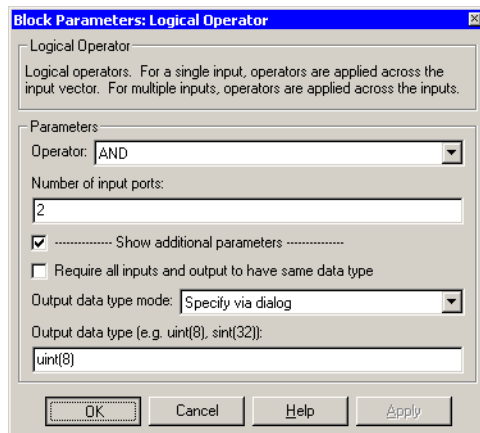
The logical operator to be applied to the block inputs. Valid choices are the operators listed previously.

### Number of input ports

The number of block inputs. The value must be appropriate for the selected operator.

### Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.





## 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 **Boolean Logic Signals** parameter in the **Advanced** tab of the Simulation Parameters Interface. If you select Logical and **Boolean Logic Signals** is on, then the output data type is always Boolean. If you select Logical and **Boolean Logic Signals** 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 Specify via dialog is selected for the **Output data type mode** parameter.

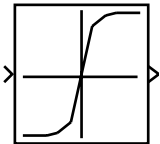
<b>Characteristics</b>	Dimensionalized	Yes
	Direct Feedthrough	Yes
	Sample Time	Inherited from the driving block
	Scalar Expansion	Of inputs
	Zero Crossing	No

# Look-Up Table

**Purpose** Approximate a one-dimensional function using the specified lookup method

**Library** Simulink Look-Up Tables and Fixed-Point Blockset LookUp

**Description** The Look-Up Table block computes an approximation to some function  $y=f(x)$  given data vectors  $x$  and  $y$ .



Look-Up  
Table

---

**Note** To map two inputs to an output, use the Look-Up Table (2-D) block.

---

The length of the  $x$  and  $y$  data vectors provided to this block must match. Also, the  $x$  data vector must be strictly monotonically increasing after conversion to the input's fixed-point data type, except in the following case. If the input  $x$  and the output signal are both either single or double, and if the lookup method is Interpolation-Extrapolation, then  $x$  may be monotonically increasing rather than strictly monotonically increasing. Note that due to quantization, the  $x$  data vector may be strictly monotonic in doubles format, but not so after conversion to a fixed-point data type.

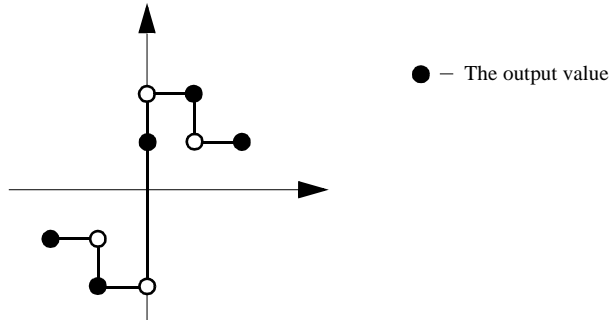
You define the table by specifying the **Vector of input values** parameter as a 1-by- $n$  vector and the **Vector of output values** parameter as a 1-by- $n$  vector. The block generates output based on the input values using one of these methods selected from the **Look-up method** parameter list:

- Interpolation-Extrapolation—This is the default method; it performs linear interpolation and extrapolation of the inputs.
  - If a value matches the block's input, the output is the corresponding element in the output vector.
  - If no value matches the block's input, then the block performs linear interpolation between the two appropriate elements of the table to determine an output value. If the block input is less than the first or greater than the last input vector element, then the block extrapolates using the first two or last two points.
- Interpolation-Use End Values—This method performs linear interpolation as described above but does not extrapolate outside the end points of the input vector. Instead, the end-point values are used.

- Use Input Nearest—This method does not interpolate or extrapolate. Instead, the element in  $x$  nearest the current input is found. The corresponding element in  $y$  is then used as the output.
- Use Input Below—This method does not interpolate or extrapolate. Instead, the element in  $x$  nearest and below the current input is found. The corresponding element in  $y$  is then used as the output. If there is no element in  $x$  below the current input, then the nearest element is found.
- Use Input Above—This method does not interpolate or extrapolate. Instead, the element in  $x$  nearest and above the current input is found. The corresponding element in  $y$  is then used as the output. If there is no element in  $x$  above the current input, then the nearest element is found.

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.

# Look-Up Table

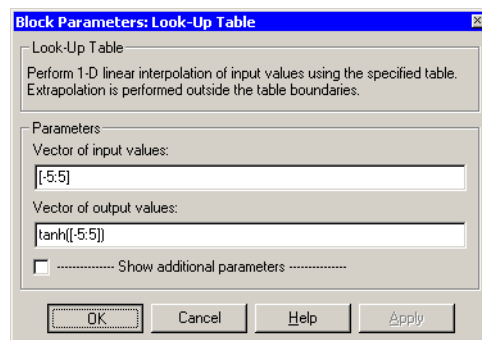
- When  $u$  is greater than zero, the output is the value connected with the point first encountered when moving away from the origin in a positive direction. In this example, when  $u$  is 1,  $y$  is 2, marked with a solid circle.
- When  $u$  is at the origin and there are two output values specified 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 Look-Up Table block 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.

When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to "Block Parameters" in the Fixed-Point Blockset documentation.

To avoid parameter saturation errors, the automatic scaling script `autofixexp` employs a special rule for the Look-Up Table block. `autofixexp` modifies the scaling by using the output look-up values in addition to the logged minimum and maximum simulation values. This prevents the data from being saturated to different values. The look-up values are given by the **Vector of output values** parameter (the `YDataPoints` variable).

## Parameters and Dialog Box



## 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.

## Vector of output values

Specify the vector of output values. The output values vector must be the same size as the input values vector.

## Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.

**Block Parameters: Look-Up Table**

Look-Up Table  
Perform 1-D linear interpolation of input values using the specified table.  
Extrapolation is performed outside the table boundaries.

Parameters:

Vector of input values:  
[-5:5]

Vector of output values:  
tanh([-5:5])

Show additional parameters

Look-up method: Interpolation-Extrapolation

Output data type mode: Specify via dialog

Output data type (e.g. fix(16), uint(8), float('single')):  
fix(16)

Output scaling value (Slope, e.g. 2^-9 or [Slope Bias], e.g. [1.25 3]):  
2^0

Lock output scaling against changes by the autoscaling tool

Round integer calculations toward: Floor

Saturate on integer overflow

OK Cancel Help Apply

# Look-Up Table

---

## **Look-up method**

Specify the lookup method. See “Description” on page 2-196 for a discussion of the options for this parameter.

## **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 *Specify via dialog* is selected for the **Output data type mode** parameter.

## **Output scaling value**

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

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

If selected, scaling of outputs is locked. This parameter is only visible if *Specify via dialog* is selected for the **Output data type mode** parameter.

## **Round integer calculations toward**

Select the rounding mode for the fixed-point output.

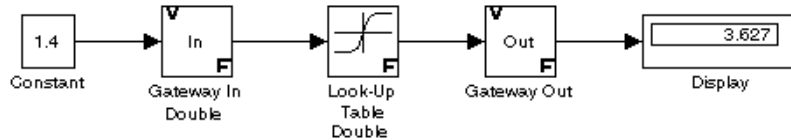
## **Saturate on integer overflow**

If selected, overflows saturate.

## **Conversions and Operations**

The **Vector of input values** parameter is converted from doubles to the input data type. The **Vector of output values** parameter is converted from doubles to the output data type. Both conversions are performed offline using round-to-nearest and saturation. Refer to “Parameter Conversions” in the Fixed-Point Blockset documentation for more information about parameter conversions.

## Examples



Suppose the Look-Up 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.

Look-Up Method	Input	Output	Comment
Interpolation-Extrapolation	1.4	2.153	N/A
	5.2	83.59	N/A
Interpolation-Use End Values	1.4	2.153	N/A
	5.2	74.2	The value for $\sinh(5.0)$ was used.
Use Input Above	1.4	3.627	The value for $\sinh(2.0)$ was used.
	5.2	74.2	The value for $\sinh(5.0)$ was used.
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.

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

**See Also** Look-Up Table (2-D), Look-Up Table (n-D)

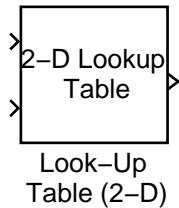
# Look-Up Table (2-D)

---

**Purpose** Approximate a two-dimensional function using a selected look-up method

**Library** Simulink Look-Up Tables and Fixed-Point Blockset LookUp

## Description



The Look-Up 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.

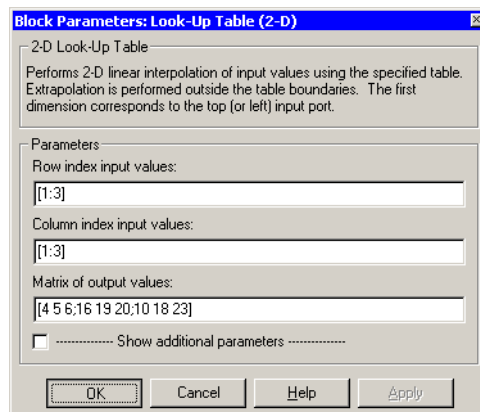


- Use **Input Below**—This method does not interpolate or extrapolate. Instead, the elements in  $x$  and  $y$  nearest and below the current inputs are found. The corresponding element in  $z$  is then used as the output. If there are no elements in  $x$  or  $y$  below the current inputs, then the 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.

To avoid parameter saturation errors, the automatic scaling script `autofixexp` employs a special rule for the Look-Up Table (2-D) block. `autofixexp` modifies the scaling by using the output look-up values in addition to the logged minimum and maximum simulation values. The output look-up values are converted to the specified output data type. This prevents the data from being saturated to different values.

When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to “Block Parameters” in the Fixed-Point Blockset documentation.

## Parameters and Dialog Box



# Look-Up 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

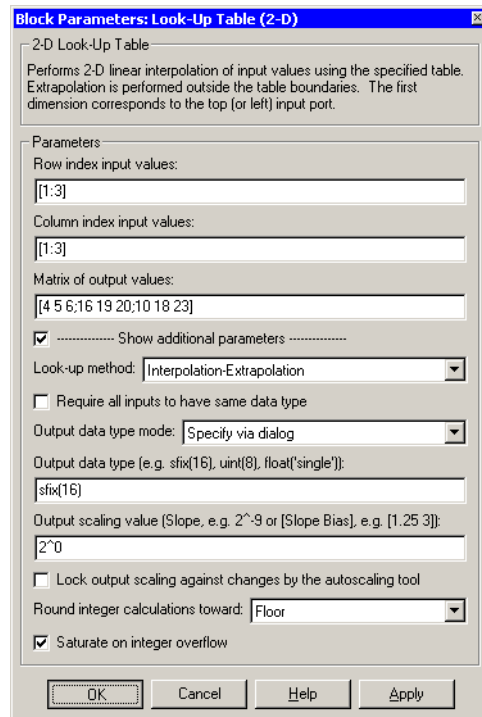
The column values for the table, entered as a vector. The vector values must increase monotonically.

## Matrix of output values

The table of output values. The matrix size must match the dimensions defined by the **Row** and **Column** parameters.

## Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.



### **Look-up method**

Specify the lookup method. See “Description” on page 2-202 for a discussion of the options for this parameter.

### **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 Specify via dialog is selected for the **Output data type mode** parameter.

### **Output scaling value**

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

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

If selected, scaling of outputs is locked. This parameter is only visible if Specify via dialog is selected for the **Output data type mode** parameter.

### **Round integer calculations toward**

Select the rounding mode for the fixed-point output.

### **Saturate on integer overflow**

If selected, overflows saturate.

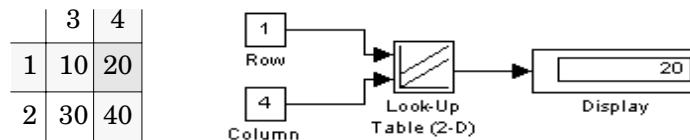
# Look-Up Table (2-D)

## Examples

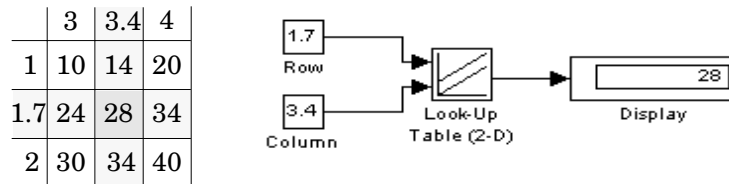
In this example, the block parameters are defined as

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.



## Characteristics

Dimensionalized	Yes
Direct Feedthrough	Yes
Sample Time	Inherited from driving blocks
Scalar Expansion	Of one input if the other is a vector
Zero Crossing	No

## See Also

Look-Up Table, Look-Up 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

Look-Up Tables

## Description

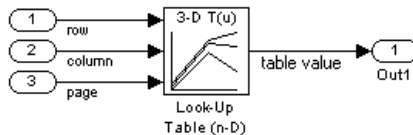


The Look-Up Table (n-D) block evaluates a sampled representation of a function in N variables by interpolating between samples to give an approximate value for  $y = F(x_1, x_2, x_3, \dots, x_n)$ , even when the function  $F$  is known only empirically. The block efficiently maps the block inputs to 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 MATLAB's online documentation](#).

# Look-Up Table (n-D)

---

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 Look-Up Table (n-D) block with the PreLook-Up Index Search block to have more flexibility and potentially much higher performance for linear interpolations in certain circumstances.

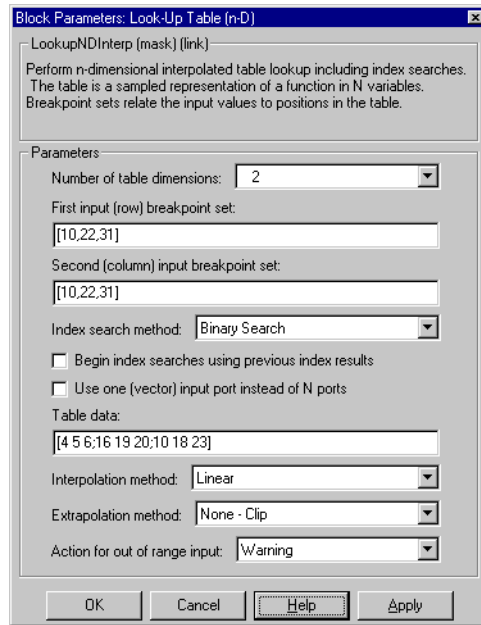
---

For noninterpolated table lookups, use the Direct Look-Up 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,  $y = table(k)$ .

## Data Type Support

A Look-Up Table (n-D) block accepts signals of types `double` or `single`, but for any given Look-Up 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.

## 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.

# Look-Up Table (n-D)

---

## **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.



## **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 “Warning” is selected, and the simulation halts with an error message if “Error” is selected.

# Look-Up Table (n-D)

---

## Characteristics

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

# Magnitude-Angle to Complex

**Purpose** Convert a magnitude and/or a phase angle signal to a complex signal

**Library** Math Operations

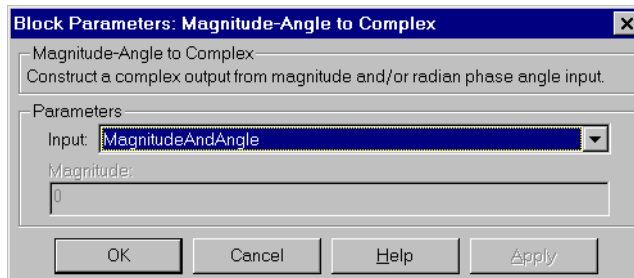
**Description** The Magnitude-Angle to Complex block converts magnitude and/or phase angle inputs to a complex-valued output signal. The inputs must be real-valued signals of type `double`. The angle input is assumed to be in radians. The data type of the complex output signal is `double`.



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.

# Magnitude-Angle to Complex

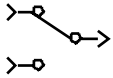
---

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Of the input when the function requires two inputs
	Dimensionalized	Yes
	Zero Crossing	No

**Purpose** Switch between two inputs

**Library** Signal Routing

**Description**



Manual Switch

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 icon (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** A Manual Switch block accepts signals of any complexity and data type, including fixed-point data types, except int64 and uint64.

**Parameters and Dialog Box** None

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

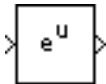
# Math Function

---

**Purpose** Perform a mathematical function

**Library** Math Operations

**Description** The Math Function block performs numerous common mathematical functions.



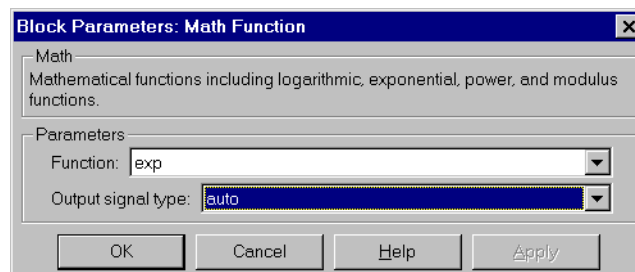
You can select one of these functions from the **Function** list: exp, log,  $10^u$ , log10, magnitude2, square, sqrt, pow, conj, reciprocal, hypot, rem, mod, transpose, and hermitian. The block output is the result of the operation of the function on the input or inputs.

The name of the function appears on the block icon. 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 can produce only scalar output.

**Data Type Support** A Math Function block accepts complex or real-valued signals or signal vectors of type double. The output signal type is real or complex, depending on the setting of the **Output signal type** parameter.

## Parameters and Dialog Box



## Function

The mathematical function.

## Output signal type

The dialog allows you to select the output signal type of the Math Function block as real, complex, or auto.

Function	Input	Output Signal Type		
	Signal	Auto	Real	Complex
Exp, log, 10u, log10, square, sqrt, pow, reciprocal, conjugate, transpose, hermitian	real complex	real complex	real error	complex complex
magnitude squared	real complex	real real	real real	complex complex
hypot, rem, mod	real complex	real error	real error	complex error

## Characteristics

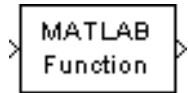
Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Of the input when the function requires two inputs
Dimensionalized	Yes
Zero Crossing	No

# 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)
```

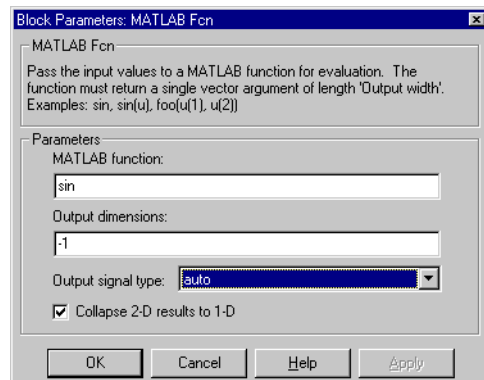
---

**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** A MATLAB Fcn block accepts one complex- or real-valued 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.

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

# Matrix Concatenation

**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 = [u1 u2 u3 ... un]           % Equivalent MATLAB code
```

For horizontal concatenation, inputs must all have the same row dimension,  $M$ , but can have different column dimensions. The output matrix has dimension  $M$ -by- $\Sigma N_i$ , 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 = [u1;u2;u3;...;un]          % Equivalent MATLAB code
```

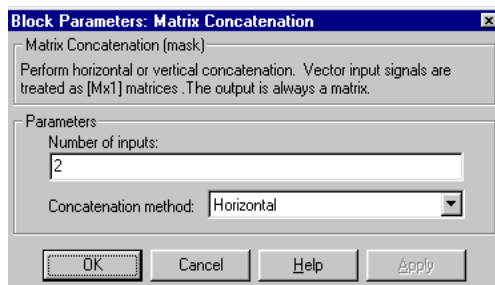
For vertical concatenation, inputs must all have the same column dimension,  $N$ , but can have different row dimensions. The output matrix has dimension  $\Sigma M_i$ -by- $N$ , where  $M_i$  is the number of rows in input  $u_i$  ( $i = 1, 2, \dots, n$ ).

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.)

## 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.

## Dialog Box



### Number of inputs

The number of matrices to concatenate.

### Concatenation method

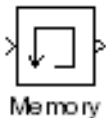
The dimension along which to concatenate the inputs.

# Memory

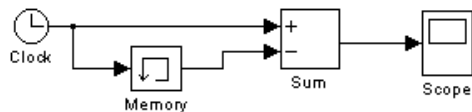
**Purpose** Output the block input from the previous integration step

**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.



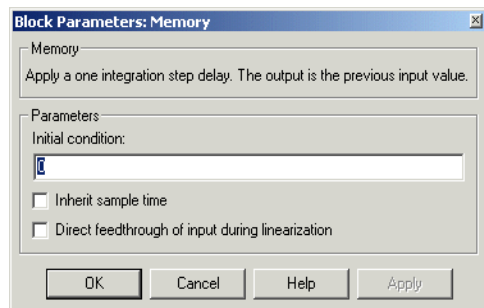
---

**Note** Avoid using the Memory block when integrating with ode15s or ode113, unless the input to the block does not change.

---

**Data Type Support** A Memory block accepts signals of any data type and complexity, including fixed-point data types, except int64 and uint64.

## 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.

## **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.

## **Characteristics**

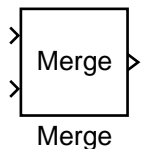
Dimensionalized	Yes
Direct Feedthrough	No, except when <b>Direct feedthrough of input during linearization is enabled</b> .
Sample Time	Continuous, but inherited from the driving block if the <b>Inherit sample time</b> check box is selected
Scalar Expansion	Yes, of the <b>Initial condition</b> parameter
Zero Crossing	No

# Merge

**Purpose** Combine multiple signals into a single signal

**Library** Signal Routing

## Description



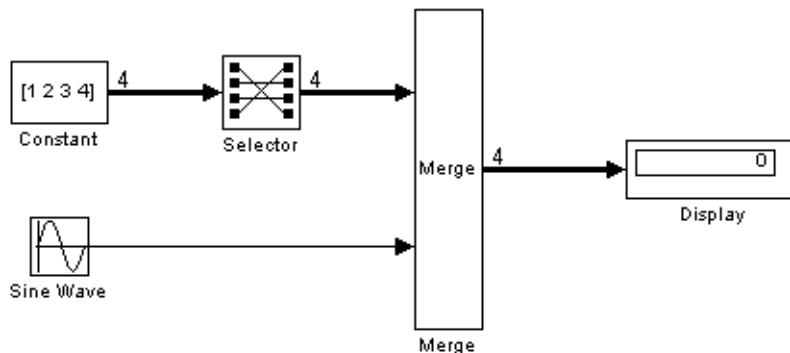
The Merge block combines its inputs into a single output line whose value at any time is equal to the most recently computed output of its driving blocks. You can specify any number of inputs by setting the block's **Number of inputs** parameter.

---

**Note** Merge blocks facilitate creation of alternately executing subsystems. See “Creating Alternately Executing Subsystems” for an application example.

---

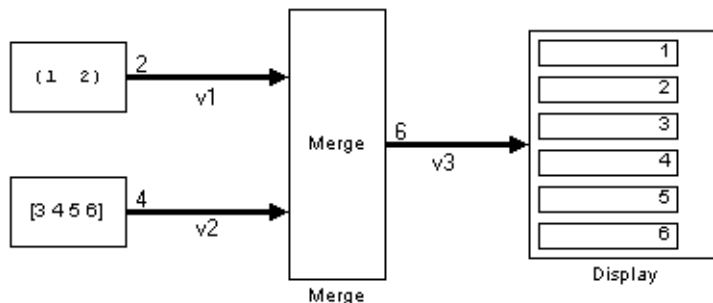
A Merge block does not accept signals whose elements have been reordered. For example, in the following diagram, the Merge block does not accept the output of the Selector block because the Selector block interchanges the first and fourth elements of the vector signal.



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 the **Allow unequal port widths** option is selected, 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.

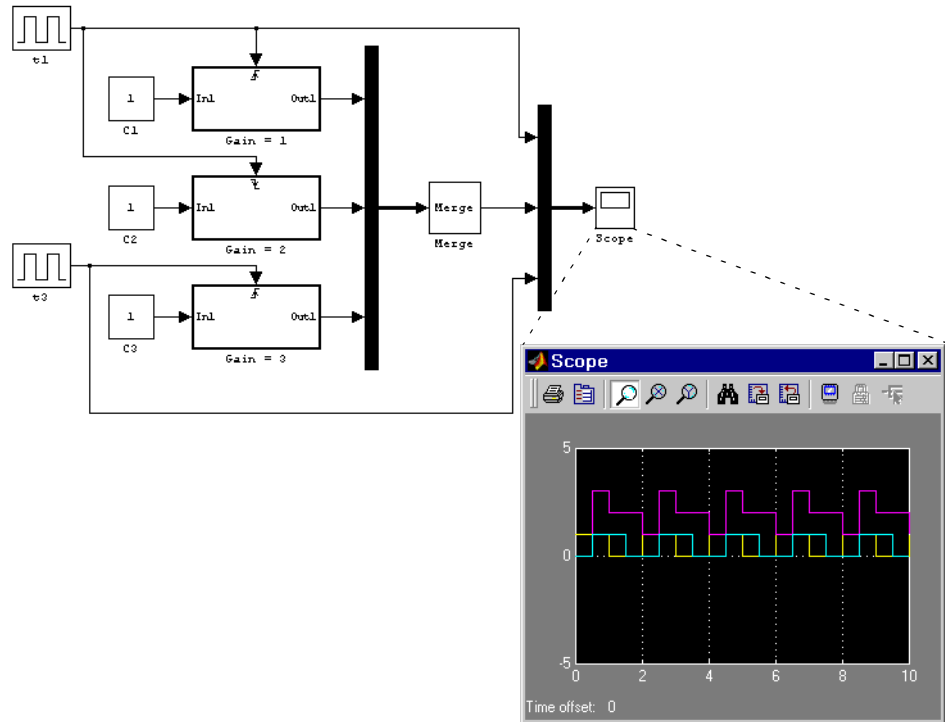
## Merging S-Function Outputs

The Merge block does not 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 `ssSetOutputPortReusable` for more information.

# Merge

## 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.



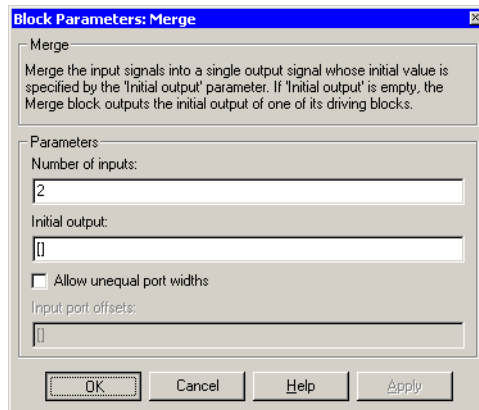
This example connects three amplifiers to a Merge block via a Mux block. The top and bottom amplifiers trigger on a rising pulse; the middle, on a falling pulse. The trigger signal connected to the bottom amplifier has a phase delay of .5 s compared to the trigger signal connected to the bottom amplifier. 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

A Merge block accepts signals of any complexity and data type, including fixed-point data types, except int64 and uint64.



## 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.

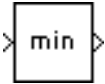
<b>Characteristics</b>	Dimensionalized	Yes
	Direct Feedthrough	Yes
	Sample Time	Inherited from the driving block
	Scalar Expansion	No
	Zero Crossing	No

# 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.



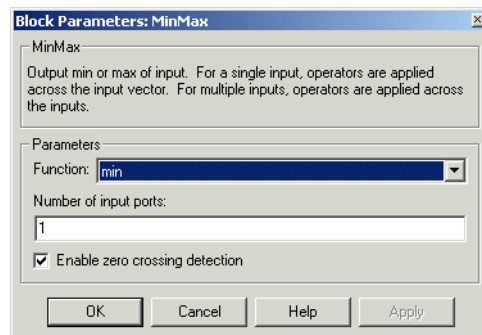
If the block has one input port, the input must be a scalar or a vector. The block outputs a scalar equal to the minimum or maximum element of the input vector.

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

A MinMax block accepts and outputs real-valued signals of any data type except int64 and uint64.

## Parameters and Dialog Box



## Function

The function (min or max) to apply to the input.

## Number of input ports

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.

## Characteristics

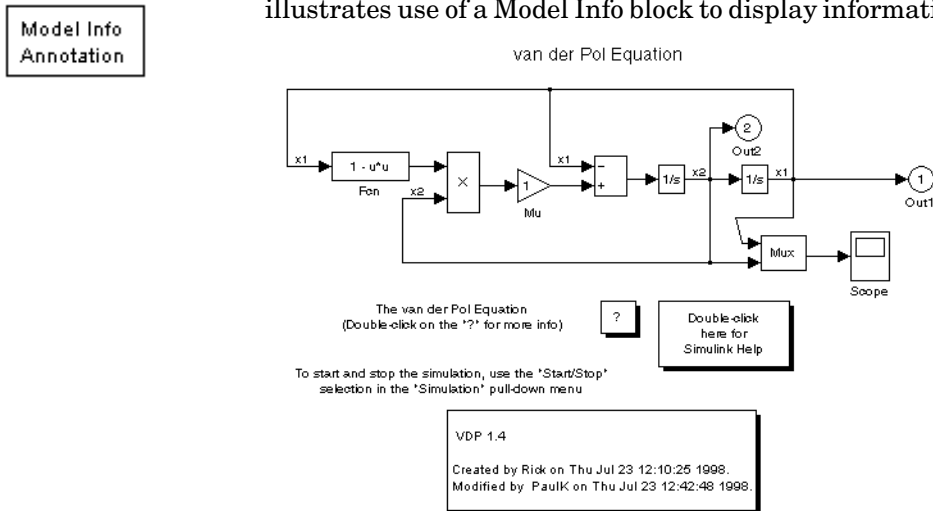
Direct Feedthrough	Yes
Sample Time	Inherited from the driving block
Scalar Expansion	Of the inputs
Dimensionalized	Yes

# Model Info

**Purpose** Display revision control information in a model

**Library** Model-Wide Utilities

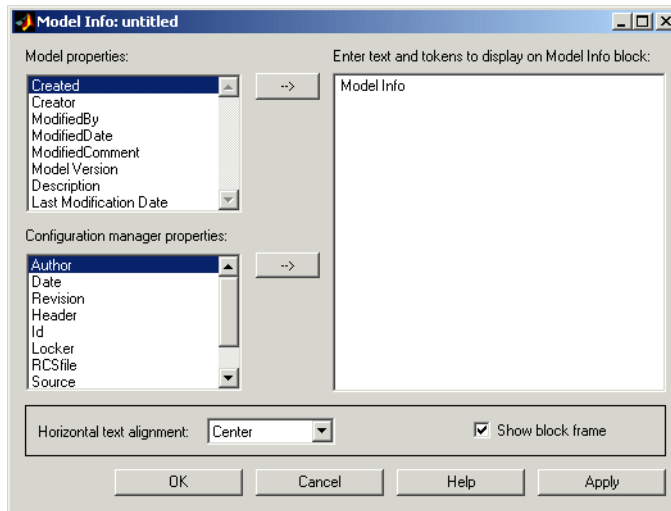
**Description** The Model Info block displays revision control information about a model as an annotation block in the model's block diagram. The following diagram illustrates use of a Model Info block to display information about the vdp model.



A Model Info block can show revision control information embedded in the model itself and/or information maintained by an external revision control or configuration management system. A Model Info block's dialog allows you to specify the content and format of the text displayed by the block.

**Data Type Support** Not applicable.

## 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.

---

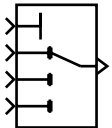
**Note** The selected item does not appear in the Model Info block until you check the model in or out of the repository maintained by the configuration manager and you have closed and reopened the model.

---

**Purpose** Choose between multiple block inputs

**Library** Simulink Signal Routing and Fixed-Point Blockset Select

## Description



Multiport  
Switch

The Multi-Port 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.

When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to “Block Parameters” in the Fixed-Point Blockset documentation.

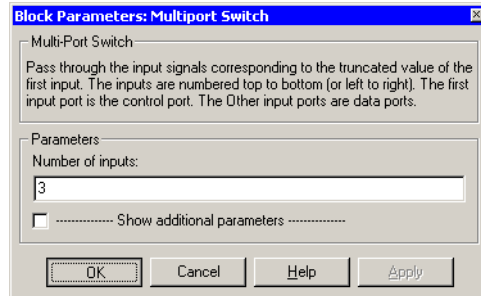
The Index Vector block, also in the Fixed-Point Blockset Select library, is another implementation of the Multi-Port Switch block that has different default parameter settings.

# Multi-Port Switch

## Data type support

The control input of a Multi-Port Switch block can be a real-valued signal of any data type, including fixed-point data types, except int64 and uint64. The data inputs can be of any complexity and data type, including fixed-point data types, except int64 and uint64.

## Parameters and Dialog Box

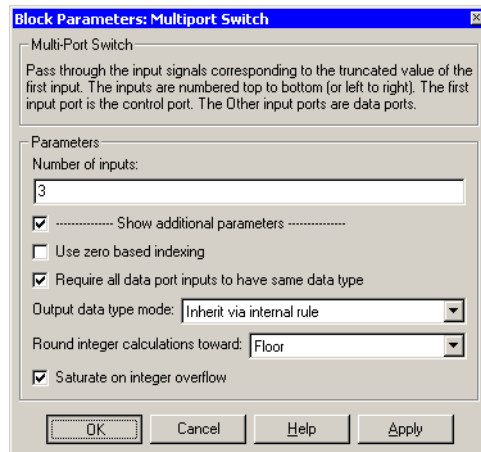


### Number of input ports

Specify the number of data inputs to the block.

### Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.





## Use zero based indexing

If selected, the block uses zero-based indexing. Otherwise, the block uses one-based indexing.

## 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 the fixed-point output.

## Saturate on integer overflow

If selected, overflows saturate.

## Characteristics

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

# Mux

---

**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

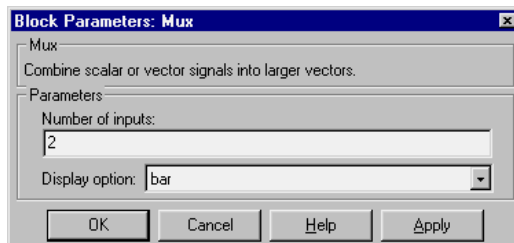
You can enter a list of signal names separated by commas. Simulink assigns each name to the corresponding port and signal. For example, if you enter position, velocity, the Mux block will have two inputs, named position and velocity.

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

**Data Type Support**

A Mux block accepts real or complex signals of any data type, including fixed-point data types, except int64 and uint64. The Mux block supports mixed-type vectors.

**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 icon in your model.

Display Option	Appearance of Block in Model
none	Mux appears inside block icon
signals	Displays signal names next to each port
bar	Displays the block icon in a solid foreground color

# Output

---

**Purpose** Create an output port for a subsystem or an external output

**Library** Ports & Subsystems, Sinks

**Description** Output blocks are the links from a system to a destination outside the system.

× 1

Out1

Simulink assigns Output block port numbers according to these rules:

- It automatically numbers the Output blocks within a top-level system or subsystem sequentially, starting with 1.
- If you add an Output block, it is assigned the next available number.
- If you delete an Output block, other port numbers are automatically renumbered to ensure that the Output blocks are in sequence and that no numbers are omitted.
- If you copy an Output block into a system, its port number is *not* renumbered unless its current number conflicts with an Output block already in the system. If the copied Output 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

Output blocks in a subsystem represent outputs from the subsystem. A signal arriving at an Output block in a subsystem flows out of the associated output port on that Subsystem block. The Output 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 Output 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 Output 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 Output block is included in the grouped blocks, Simulink automatically renumbers the ports on the blocks.

The Output block name appears in the Subsystem block 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 **Simulation 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 **Simulation 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”)

# Output

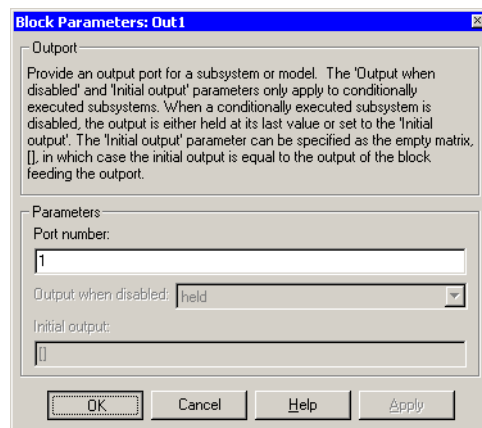
## Data Type Support

An Output block accepts complex or real signals of any data type except `int64` and `uint64` as input, with the following exception. An Output block can only accept fixed-point data types if it is not a root-level output. The complexity and data type of the block's output are the same as those of its input.

The elements of a signal array connected to an Output block can be of differing complexity and data types except in the following circumstance: If the output is in a conditionally executed subsystem and the initial output is specified, all elements of an input array must be of the same complexity and data types.

Typical Simulink data type conversion rules apply to an output'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



### Port number

Specify the port number of the Output block.

### Output when disabled

For conditionally executed subsystems, specify 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.

## Characteristics

Dimensionalized

Yes

Sample Time

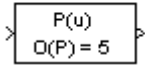
Inherited from driving block

# Polynomial

**Purpose** Perform evaluation of polynomial coefficients on input values

**Library** Math Operations

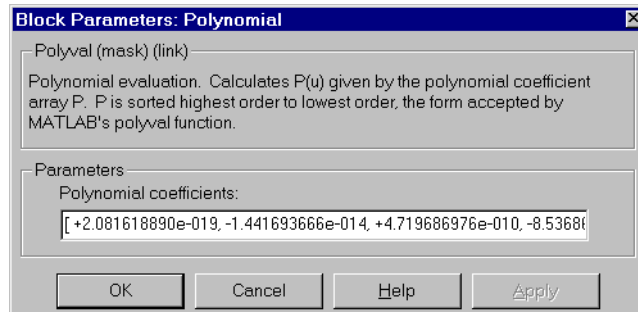
**Description** The Polynomial block uses a coefficients parameter to evaluate a real polynomial for the input value.



You define a set of polynomial coefficients in the form accepted by MATLAB's `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

# Prelook-Up 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** Look-Up Tables

## Description



The PreLook-Up 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 PreLook-Up 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 PreLook-Up 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 icon. Note that the index value is zero-based.

---

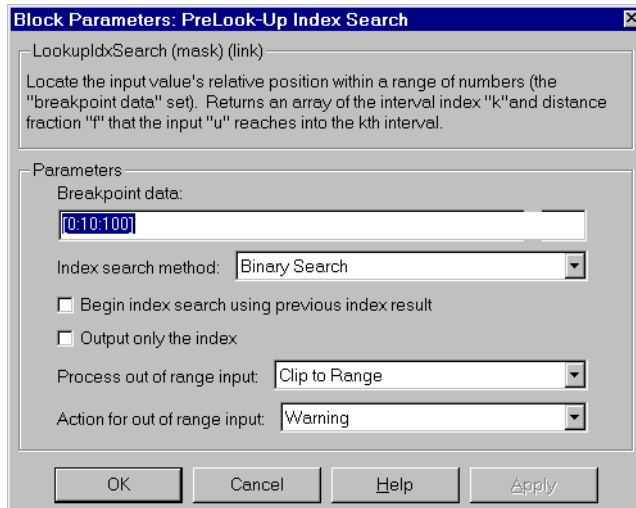
**Note** The interval fraction can be negative or greater than 1 for out-of-range input. See the documentation for the block's **Process out of range input parameter** for more information.

---

## Data Type Support

A PreLook-Up 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 PreLook-Up block, the interval fraction output can be dropped and the resulting index value output is a uint32 instead.

# Prelook-Up Index Search

---

## 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 this option is selected. 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 this option is selected. 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 Simulink Diagnostic Viewer).

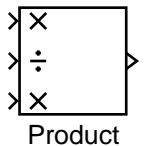
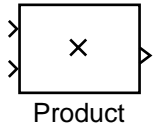
## Characteristics

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

**Purpose** Multiply or divide inputs

**Library** Simulink Math Operations and Fixed-Point Blockset Math

**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,

# Product

---

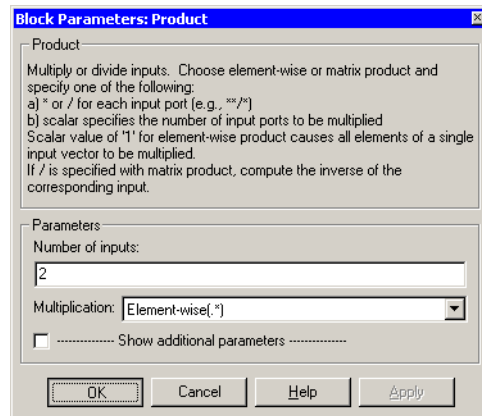
however, the **Multiplication** parameter is set to **Matrix**, then a single “\*” will cause the block to output the matrix unchanged, and a single “/” will cause the block to output the inverse of the matrix.

When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to “Block Parameters” in the Fixed-Point Blockset documentation.

## Data Type Support

The Product block accepts signals of any complexity and data type, including fixed-point data types except `int64` and `uint64`. All input signals must be of the same data type.

## Parameters and Dialog Box



### Number of inputs

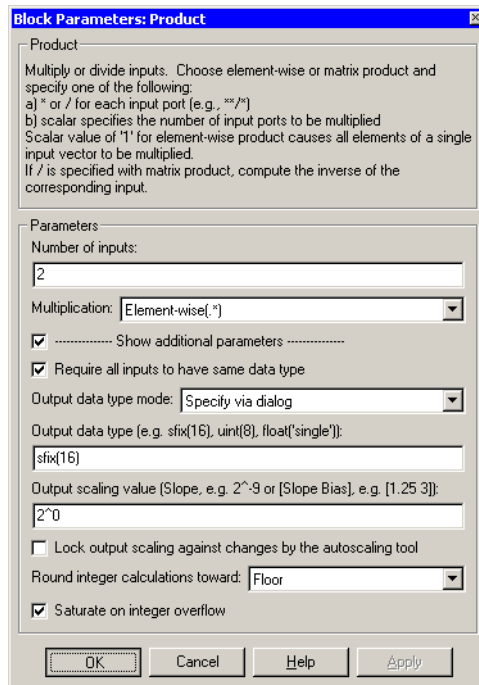
Enter the number of inputs or a combination of “\*” and “/” symbols. See “Description” above for a complete discussion of this parameter.

### Multiplication

Specify element-wise or matrix multiplication. See “Description” above for a complete discussion of this parameter.

### Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.



### 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 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 **Production hardware characteristics**

parameter on the **Advanced** pane of the **Simulation Parameters** dialog is set to `Unconstrained integer sizes`, Simulink chooses the data type without regard to hardware constraints. If the parameter is set to `Microprocessor`, 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 `Unconstrained integer sizes` is specified, the output data type is `sfixed24`. If `Microprocessor` is specified and the microprocessor supports 8-bit, 16-bit, and 32-bit words, 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 Simulink Diagnostic Viewer.

### **Output data type**

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

### **Output scaling value**

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

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

If selected, **Output scaling** is locked. This parameter is only visible if `Specify via dialog` is selected for the **Output data type mode** parameter.

### **Round integer calculations toward**

Select the rounding mode for fixed-point output.

### **Saturate on integer overflow**

If selected, overflows saturate.



## **Conversions and Operations**

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. Refer to “Rules for Arithmetic Operations” in the Fixed-Point Blockset documentation for more information about the rules that this block obeys when performing fixed-point operations.

## **Characteristics**

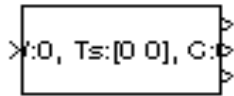
Dimensionalized	Yes
Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes
Zero Crossing	No

# Probe

**Purpose** Output a signal's attributes, including width, dimensionality, sample time, and/or complex signal flag

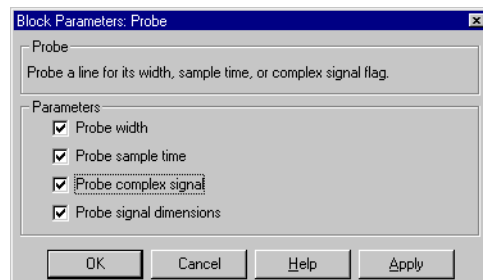
**Library** Signal Attributes

**Description** The Probe block outputs selected information about the signal on its input. The block can output the input signal's width, dimensionality, sample time, and/or a flag indicating whether the input is a complex-valued signal. The block has one input port. The number of output ports depends on the information that you select for probing, that is, signal dimensionality, sample time, and/or complex signal flag. Each probed value is output as a separate signal on a separate output port. The block accepts real or complex-valued signals of any built-in data type. It outputs signals of type `double`. During simulation, the block's icon displays the probed data.



**Data Type Support** A Probe block accepts and outputs any data type, including fixed-point data types, except `int64` and `uint64`.

## Parameters and Dialog Box



### Probe width

If selected, output the width (number of elements) of the probed signal.

### Probe sample time

If selected, output the sample time of the probed signal.

### Probe complex signal

If selected, output 1 if the probed signal is complex; otherwise, 0.

### Probe signal dimensions

If selected, output the dimensions of the probed signal.

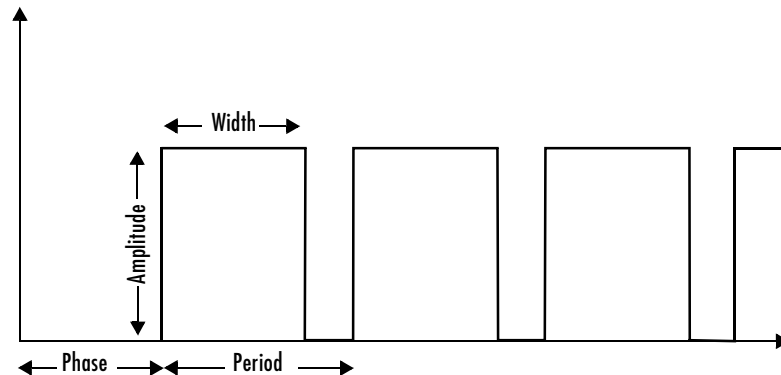
<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Yes
	Dimensionalized	Yes
	Zero Crossing	No

# 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 **Simulation Parameters** dialog box.

---

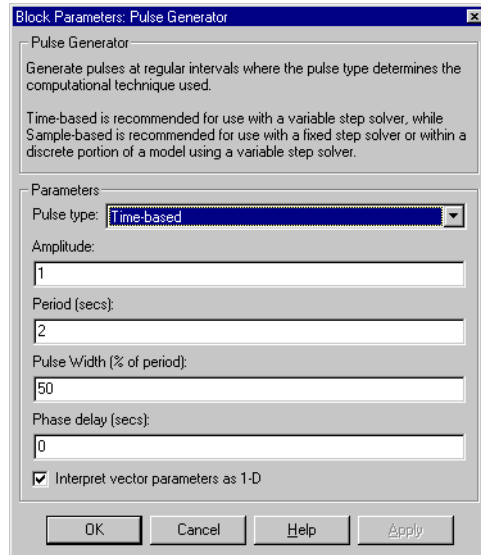
If you select time-based as the block's pulse type, you must specify the pulse's phase delay and period in units of seconds. If you specify sample-based, you must specify the block's sample time in seconds, using the **Sample Time** parameter, then specify the block's phase delay and period as integer 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

A Pulse Generator block outputs real signals of any data type except `int64` and `uint64`. The data type of the output signal is the same as that of the **Amplitude** parameter.

# Pulse Generator

## Parameters and Dialog Box



### Pulse type

The pulse type for this block: time-based or sample-based. The default is time-based.

### 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.

## 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 this option is selected 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.

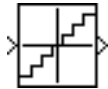
<b>Characteristics</b>	Sample Time	Inherited
	Scalar Expansion	Of parameters
	Dimensionalized	Yes
	Zero Crossing	No

# Quantizer

**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.

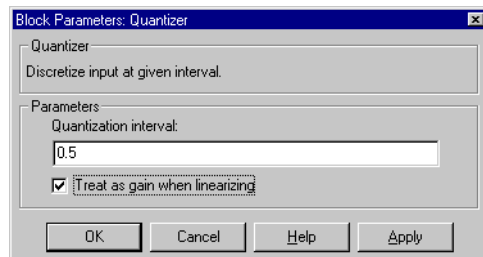
$$y = q * \text{round}(u/q)$$

where  $y$  is the output,  $u$  the input, and  $q$  the **Quantization interval** parameter.

## Data Type Support

A 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.



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

# Ramp

---

**Purpose** Generate constantly increasing or decreasing signal

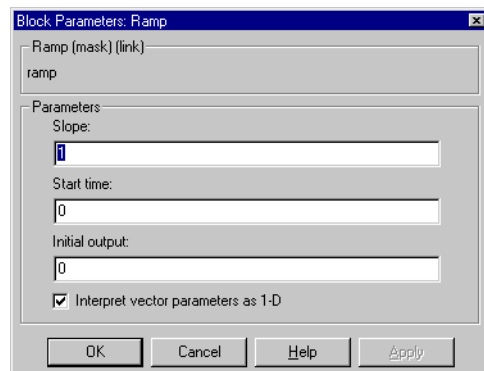
**Library** Sources

**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** A 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 this option is selected 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	Inherited from driven block
	Scalar Expansion	Yes
	Dimensionalized	Yes
	Zero Crossing	Yes

# Random Number

---

**Purpose** Generate normally distributed random numbers

**Library** Sources

**Description** The Random Number block generates normally distributed random numbers. The seed is reset to the specified value each time a simulation starts.



By default, the sequence produced has a mean of 0 and a variance of 1, although you can vary these parameters. The sequence of numbers is repeatable and can be produced by any Random Number block with the same seed and parameters. To generate a vector of random numbers with the same mean and variance, specify the **Initial seed** parameter as a vector.

To generate uniformly distributed random numbers, use the Uniform Random Number block.

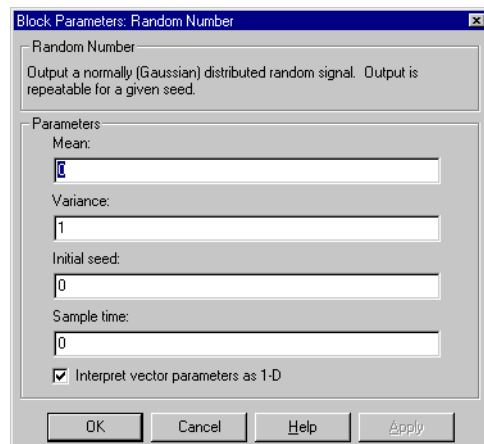
Avoid integrating a random signal, because solvers are meant to integrate relatively smooth signals. Instead, use the Band-Limited White Noise block.

All the block's numeric parameters must be of the same dimension after scalar expansion.

## Data Type Support

A Random Number block accepts and outputs signals of type double.

## Parameters and Dialog Box



**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 this option is selected 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	Continuous or discrete
Scalar Expansion	Of parameters
Dimensionalized	Yes
Zero Crossing	No

# Rate Limiter

**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

A Rate Limiter block accepts and outputs signals of type double.

## Parameters and Dialog Box



**Rising slew rate**

The limit of the derivative of an increasing input signal.

**Falling slew rate**

The limit of the derivative of a decreasing input signal.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Continuous
	Scalar Expansion	Of input and parameters
	Dimensionalized	Yes
	Zero Crossing	No

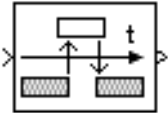
# Rate Transition

---

**Purpose** Handle transfer of data between blocks operating at different rates

**Library** Signal Attributes

**Description** 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.

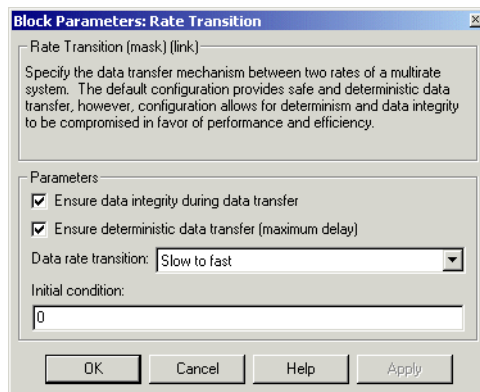


**Note** The Zero-Order Hold and Unit Delay blocks also enable transfer of data between blocks operating at different rates. However, you should use the Rate Transition block for this purpose because it is designed specifically for this purpose, offers a wider range of options, and is easier to use.

## Data Type Support

A Rate Transition block accepts and outputs signals of any data type, including fixed-point data types, except int64 and uint64.

## 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 this option is selected 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.

# Rate Transition

---

## 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.

## Data rate transition

Select `Slow to fast` if the block connected to the input of the Rate Transition block operates at a slower rate than the block connected to the Rate Transition block. Otherwise, select `Fast to slow`.

## Initial condition

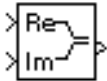
This parameter applies only to `Slow to fast` transitions. It specifies the Rate Transition’s initial output at the beginning of a transition when there is not yet any output from the slow block connected to the Rate Transition block’s input.

<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	Of input.
	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.

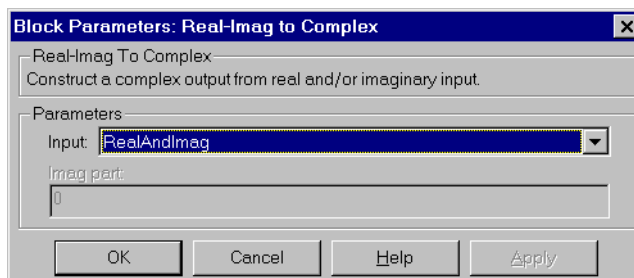


The inputs can both be arrays (vectors or matrices) of equal dimensions, or one input can be an array and the other a scalar. If the block has an array input, the output is a complex array of the same dimensions. The elements of the real input are mapped to the real parts of the corresponding complex 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, including fixed-point data types, except `int64` and `uint64`. The output is of the same type as the input or parameter that determines the output.

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

## Parameters and Dialog Box



### Input

Specifies the kind of input: a real input, an imaginary input, or both.

### Real (Imag) part

If the input is a real-part signal, this parameter specifies the constant imaginary part of the output signal. If the input is the imaginary part, this parameter specifies the constant real part of the output signal. Note that the title of this field changes to reflect its usage.

# Real-Imag to Complex

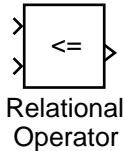
---

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Of the input when the function requires two inputs
	Dimensionalized	Yes
	Zero Crossing	No

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

**Library** Simulink Math Operations and Fixed-Point Blockset Logic & Comparison

**Description** The Relational Operator block performs the specified comparison of its two inputs.



The relational operator connecting the two inputs is selected with the **Relational Operator** parameter. The block icon updates to display the selected operator. The supported operations are given below.

Operation	Description
==	TRUE if the first input is equal to the second input
~=	TRUE if the first input is not equal to the second input
<	TRUE if the first input is less than the second input
<=	TRUE if the first input is less than or equal to the second input
>=	TRUE if the first input is greater than or equal to the second input
>	TRUE if the first input is greater than the second input

You can specify inputs as scalars, arrays, or a combination of a scalar and an array:

- For scalar inputs, the output is a scalar.
- For array inputs, the output is an array of the same dimensions, where each element is the result of an element-by-element comparison of the input arrays.
- For mixed scalar/array inputs, the output is an array, where each element is the result of a comparison between the scalar and the corresponding array element.

The 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.

# Relational Operator

---

---

**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.

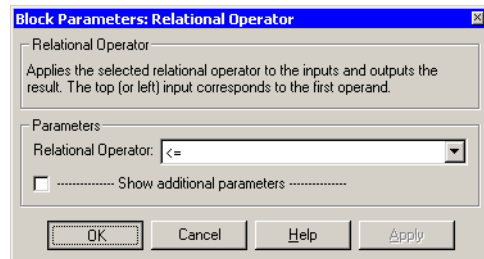
---

## Data Type Support

A Relational Operator block accepts real or complex signals of any data type except `int64` and `uint64`. 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 `!=`.

## Parameters and Dialog Box

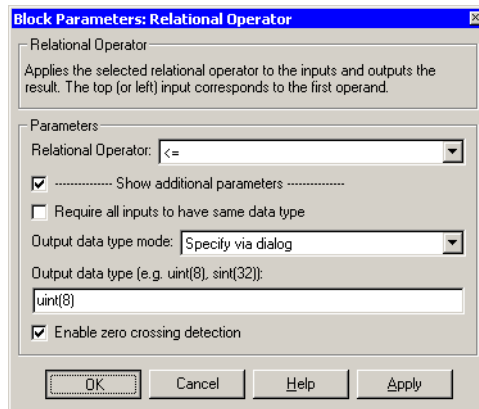


## Relational Operator

Designate the relational operator used to compare the two inputs.

## Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.



## 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 **Boolean Logic Signals** parameter in the **Advanced** tab of the Simulink Simulation Parameters Interface. If you select `Logical` and **Boolean Logic Signals** is on, then the output data type is always boolean. If you select `Logical` and **Boolean Logic Signals** is off, then the output data type will match the input data type is always double.

## 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 `Specify via dialog` is selected for the **Output data type mode** parameter.

## Enable zero crossing detection

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

# Relational Operator

---

## Conversions and Operations

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. Refer to “Parameter Conversions” in the Fixed-Point Blockset documentation for more information about parameter conversions.

## Characteristics

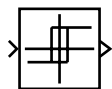
Dimensionalized	Yes
Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Of inputs
Zero Crossing	No, unless <b>Enable zero crossing detection</b> is selected.



**Purpose** Switch output between two constants

**Library** Simulink Discontinuities and Fixed-Point Blockset Nonlinear

## Description



Relay

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.

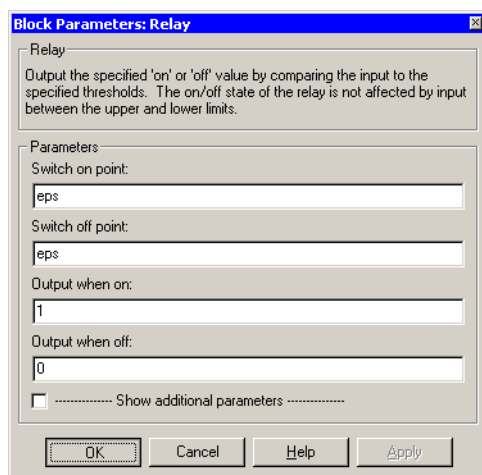
The **Switch on point** value must be greater than or equal to the **Switch off point**. Specifying a **Switch on point** value greater than the **Switch off point** value models hysteresis, whereas specifying equal values models a switch with a threshold at that value.

When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to “Block Parameters” in the Fixed-Point Blockset documentation.

## Data Type Support

This block supports any data type, including fixed-point, except int64 and uint64.

## Parameters and Dialog Box



## Switch on point

The “on” threshold for the relay.

## Switch off point

The “off” threshold for the relay.

## Output when on

The output when the relay is on.

## Output when off

The output when the relay is off.

## Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.

**Block Parameters: Relay**

Relay  
Output the specified 'on' or 'off' value by comparing the input to the specified thresholds. The on/off state of the relay is not affected by input between the upper and lower limits.

Parameters

Switch on point:  
eps

Switch off point:  
eps

Output when on:  
1

Output when off:  
0

..... Show additional parameters .....

Output data type mode: Specify via dialog

Output data type (e.g. `sfixed(16)`, `uint(8)`, `float('single')`):  
sfixed(16)

Output scaling value [Slope, e.g.  $2^{-9}$  or [Slope Bias], e.g. [1.25 3]]:  
 $2^0$

Parameter scaling Use specified scaling

Enable zero crossing detection

OK Cancel Help Apply

### 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 Specify via dialog is selected for the **Output data type mode** parameter.

### Output scaling value

Set the output scaling using radix point-only or [Slope Bias] scaling. This parameter is only visible if Specify via dialog is selected for the **Output data type mode** parameter, and is only enabled if Use specified scaling is selected 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 radix 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 Specify via dialog is selected for the **Output data type mode** parameter.

### 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.

## Conversions and Operations

The **Switch on point** and **Switch off point** parameters are converted to the input data type offline using round-to-nearest and saturation.

# Relay

---

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

**Purpose** Generate an arbitrarily shaped periodic signal

**Library** Sources

## Description

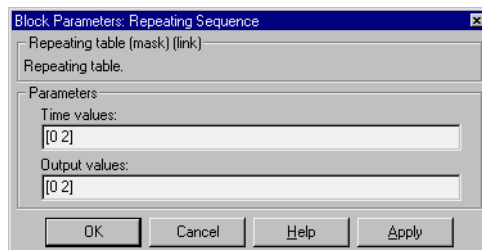


The Repeating Sequence block outputs a periodic scalar signal having a waveform that you specify. You can specify any waveform, using the block dialog's **Time values** and **Output values** parameters. The **Times value** parameter specifies a vector of sample times. The **Output values** parameter 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

A 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

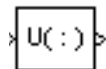
---

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

**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.

<b>Output Dimensionality</b>	<b>Description</b>
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.

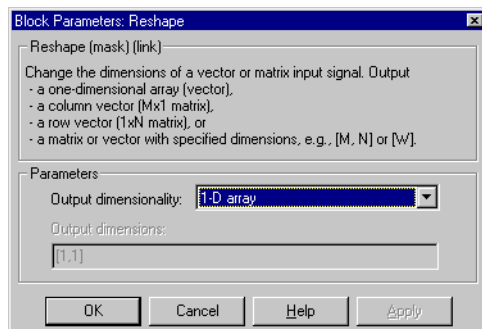
# Reshape

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, including fixed-point data types, except int64 and uint64.

## 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.



<b>Characteristics</b>	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 icon.

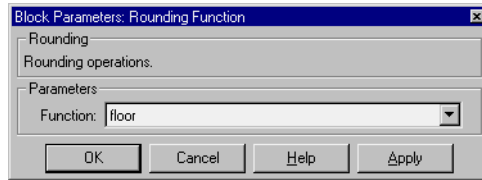
The input signal can be a scalar, vector, or matrix signal having real- or complex-valued elements of type `double`. The output signal has the same dimensions, data type, and numeric type as the input. Each element of the output signal is the result of applying the selected rounding function to the corresponding 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

A Rounding Function block accepts and outputs real signals of type `double` or `single`.

## Parameters and Dialog Box



### Function

The rounding function.

### Characteristics

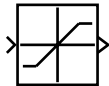
Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	N/A
Dimensionalized	Yes
Zero Crossing	No

# Saturation

**Purpose** Limit the range of a signal

**Library** Simulink Discontinuities and Fixed-Point Blockset Nonlinear

## Description



Saturation

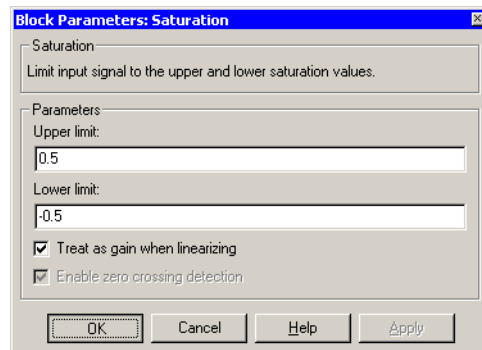
The Saturation block imposes upper and lower bounds on a signal. When the input signal is within the range specified by the **Lower limit** and **Upper limit** parameters, the input signal passes through unchanged. When the input signal is outside these bounds, the signal is clipped to the upper or lower bound.

When the **Lower limit** and **Upper limit** parameters are set to the same value, the block outputs that value.

## Data Type Support

A Saturation block accepts and outputs real signals of any data type, including fixed-point data types, except `int64` and `uint64`. The output data type is the same as the input data type.

## 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.

### 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.

## **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.

## **Conversions and Operations**

Both the **Upper limit** and **Lower limit** parameters are converted to the input data type offline using round-to-nearest and saturation.

## **Characteristics**

Dimensionalized	Yes
Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Of parameters and input
Zero Crossing	No, unless <b>Enable zero crossing detection</b> is selected

# Scope, Floating Scope

**Purpose** Display signals generated during a simulation

**Library** Sinks

## Description

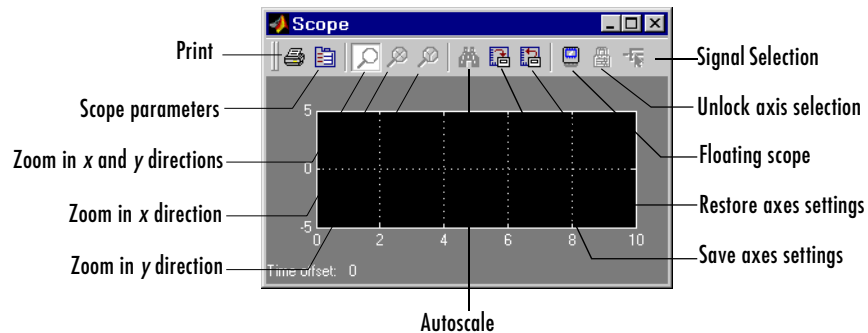


The Scope block displays its input with respect to simulation time. The Scope block can have multiple axes (one per port); all axes have a common time range with independent  $y$ -axes. The Scope allows you to adjust the amount of time and the range of input values displayed. You can move and resize 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.

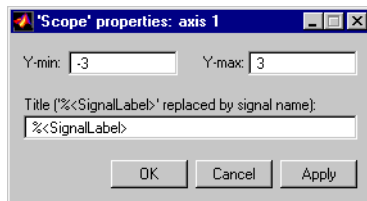
---

## 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.

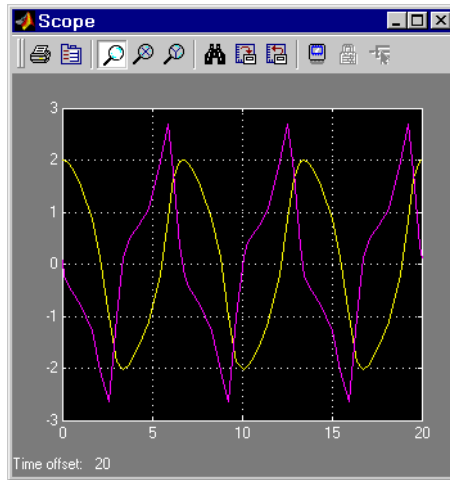
### Title

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

## Time Offset

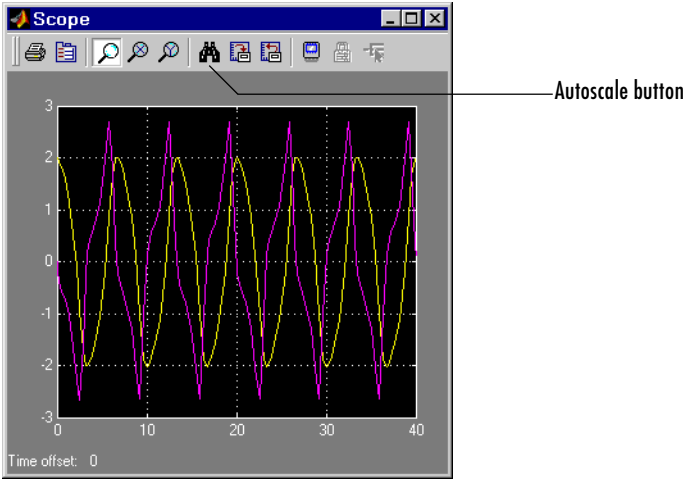
This figure shows the Scope block displaying the output of the vdp model. The simulation was run for 40 seconds. Note that this scope shows the final 20 seconds of the simulation. The **Time offset** field displays the time corresponding to 0 on the horizontal axis. Thus, you have to add the offset to 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

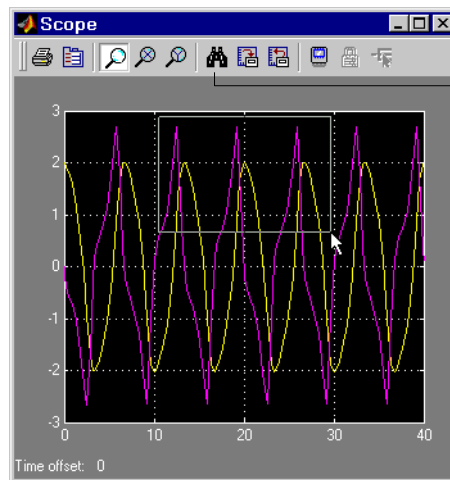
## Zooming

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.

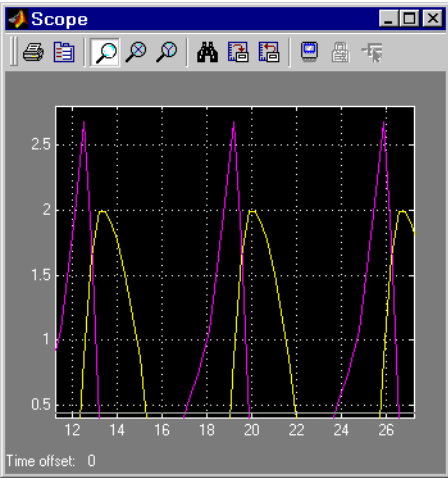
To zoom in on data in both directions at the same time, make sure the leftmost **Zoom** toolbar button is selected. Then, define the zoom region using a bounding box. When you release the mouse button, the Scope displays the data in that area. You can also click a point in the area you want to zoom in on.

If the scope has multiple  $y$ -axes, and you zoom in on one set of  $x$ - $y$  axes, the  $x$ -limits on all sets of  $x$ - $y$  axes are changed so that they match, because all  $x$ - $y$  axes must share the same time base ( $x$ -axis).

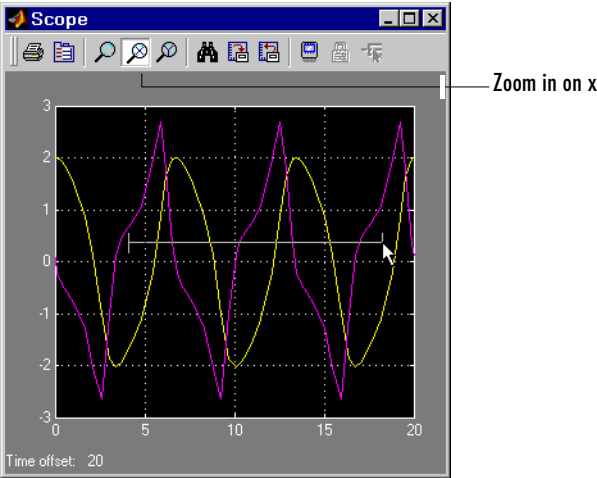
This figure shows a region of the displayed data enclosed within a bounding box.



This figure shows the zoomed region, which appears after you release the mouse button.



To zoom in on data in just the x direction, click the middle **Zoom** toolbar button. Define the zoom region by positioning the pointer at one end of the region, pressing and holding down the mouse button, then moving the pointer to the other end of the region. This figure shows the Scope after you define the zoom region, but before you release the mouse button.



# Scope, Floating 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.



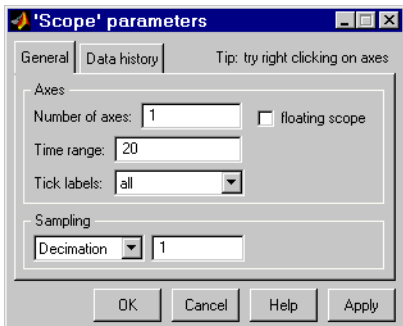
You might want to do this after zooming in on a region of the displayed data so you can see the same region in another simulation. The time range is inferred from the current  $x$ -axis limits.

## 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, this dialog box appears.



The dialog box has two panes: **General** and **Data history**.

## General Parameters

You can set the axis parameters, time range, and tick labels in the **General** tab. You can also choose the **floating scope** option with this tab.

### 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

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

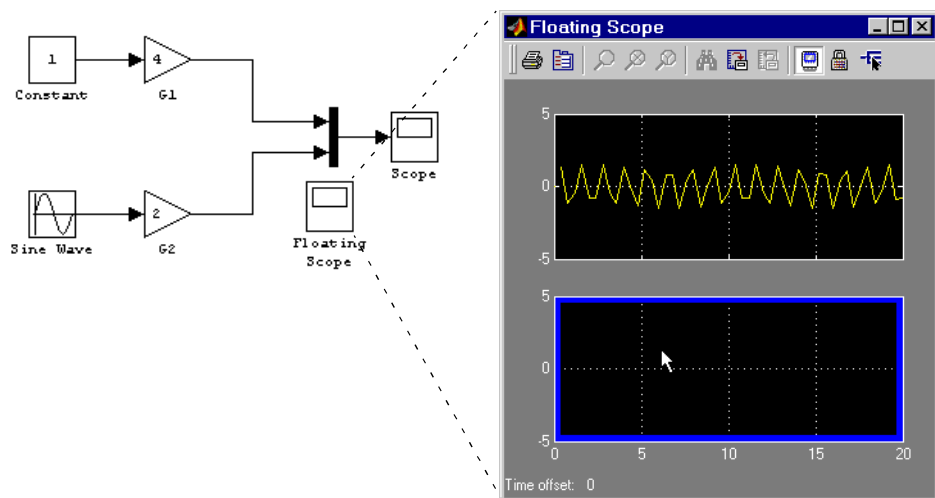
# Scope, Floating Scope

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.

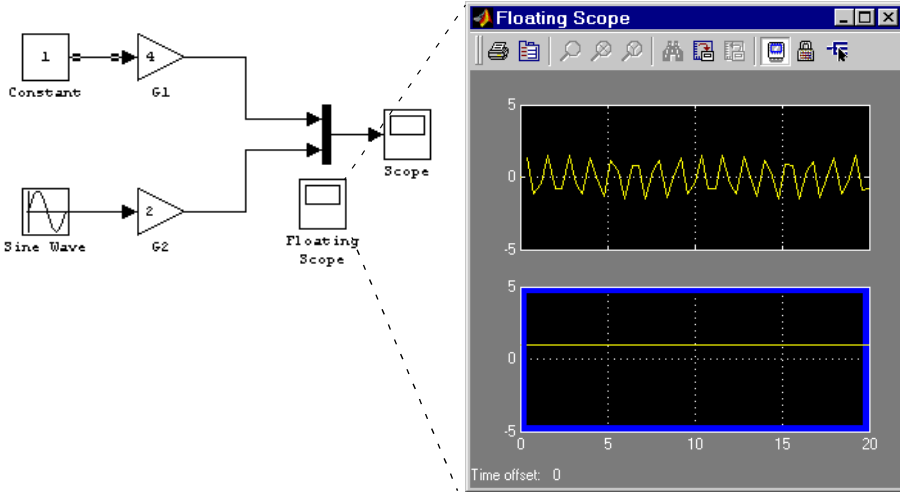
To use a floating scope during a simulation, first open the scope. To display the signals carried on a line, select the line. Hold down the **Shift** key while clicking another line to select multiple lines. It might be necessary to click the **Autoscale data** button on the floating scope's toolbar to find the signal and adjust the axes to the signal values. Or you can use the floating scope's Signal Selector (see "Signal Selector" on page 2-299) to select signals for display. The Signal Selector allows you to select signals anywhere in your model, including unopened subsystems.

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, nonactive 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.



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

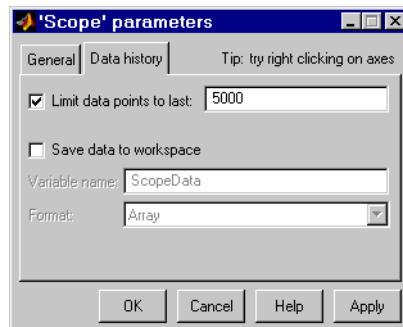
To specify a decimation factor, enter a number in the data field to the right of the **Decimation** choice. To display data at a sampling interval, select the **Sample time** choice and enter a number in the data field.

# Scope, Floating Scope

---

## Controlling Data Collection and Display

You can control the amount of data that the Scope stores and displays by setting fields on the **Data History** pane.



You can also choose to save data to the workspace in this tab. You apply the current parameters and options by clicking the **Apply** or **OK** button. The values that appear in these fields are the values that are used in the next simulation.

### Limit data points to last

You can limit the number of data points saved to the workspace by selecting the **Limit data points to last** check box and entering a value in its data field. The Scope relies on its data history for zooming and autoscaling operations. If the number of data points is limited to 1,000 and the simulation generates 2,000 data points, only the last 1,000 are available for regenerating the display.

### Save data to workspace

You can automatically save the data collected by the Scope at the end of the simulation by selecting the **Save data to workspace** check box. If you select this option, the **Variable name** and **Format** fields become active.

### Variable name

Enter a variable name in the **Variable name** field. The specified name must be unique among all data logging variables being used in the model. Other data logging variables are defined on other Scope blocks, To Workspace blocks, and simulation return variables such as time, states, and outputs. Being able to save Scope data to the workspace means that it



is not necessary to send the same data stream to both a Scope block and a To Workspace block.

### Format

Data can be saved in one of three formats: Array, Structure, or Structure with time. Use Array only for a Scope with one set of axes. For Scopes with more than one set of axes, use Structure if you do not want to store time data and use Structure with time if you want to store time data.

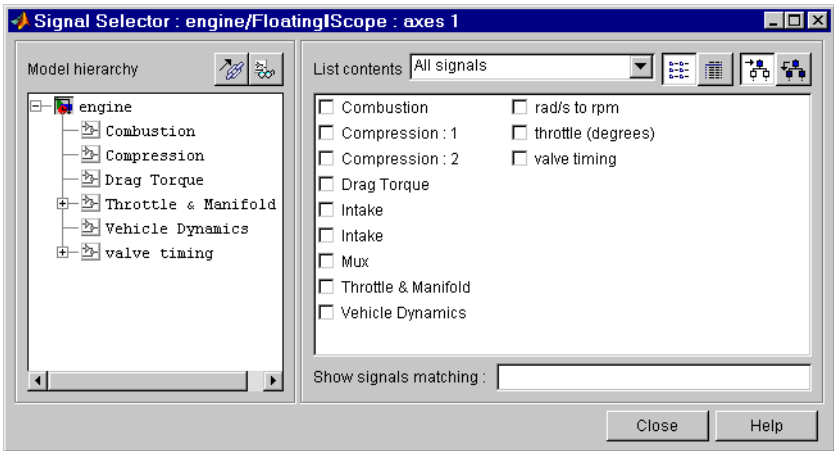
### Printing the Contents of a Scope Window

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.



### Signal Selector

The Signal Selector allows you to select the signals to be displayed in the floating scope. You can use it to select any signal in your model, including signals in unopened subsystems. To display the 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. The Signal Selector appears.



## Scope, Floating Scope

---

The Signal Selector contains two panes. The left pane allows you to display signals of any subsystem in your model. The signals appear in the right pane. The right pane allows you to select the signals to display in the floating scope.

To select a subsystem for viewing, click its entry in the **Model hierarchy** tree or use the up or down arrow keys to move the selection highlight to the entry. To show or hide the subsystems contained by the currently selected subsystem, click the +/- button next to the subsystem's name or press the forward or backward arrow keys on your keyboard. To view subsystems included as library links in your model, click the **Library Links** button at the top of the **Model hierarchy** pane. To view the subsystems contained by masked subsystems, click the **Look Under Masks** button at the top of the pane.

The **Signals** pane shows all the signals in the currently selected subsystem by default. To show named signals only, select `Named signals only` from the **List contents** control at the top of the pane. To show test point signals only, select `Test point signals only` from the **List contents** control. To show only signals whose signals match a specified string of characters, enter the characters in the **Show signals matching** control at the bottom of the **Signals** pane and press the **Enter** key. To show the selected types of signals for all subsystems below the currently selected subsystem in the model hierarchy, select the **Current and Below** button at the top of the **Signals** pane.

The **Signals** pane by default shows the name of each signal and the number of the port that emits the signal. To display more information on each signal, select the **Detailed view** button at the top of the pane. The detailed view shows the path and data type of each signal and whether the signal is a test point.

To select or deselect a signal in the **Signals** pane, click its entry or use the arrow keys to move the selection highlight to the signal entry and press the **Enter** key. You can also move the selection highlight to a signal entry by typing the first few characters of its name (enough to uniquely identify it).

---

**Note** You can continue to select and deselect signals on the block diagram with the Signal Selector open. For example, shift-clicking a line in the block diagram adds the corresponding signal to the set of signals that you previously selected with the Signal Selector. Simulink updates the Signal Selector to reflect signal selection changes you have made on the block diagram. However, the changes do not appear until you select the Signal Selector window itself.

---

## Data Type Support

A Scope block accepts real signals of any data type, including fixed-point data types, except `int64` and `uint64`. The Scope block accepts homogeneous vectors.

## 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.

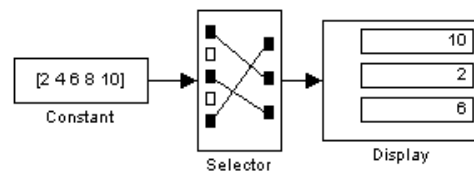


A Selector block accepts either vector or matrix signals as input. Set the **Input Type** parameter to the type of signal (vector or matrix) that the block should accept in your model. The parameter dialog box and the block icon change to reflect the type of input that you select. The way the block determines 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. 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 is being selected. For example, this model shows the Selector block icon 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 icon is large enough, it displays the ordering of input vector elements graphically.

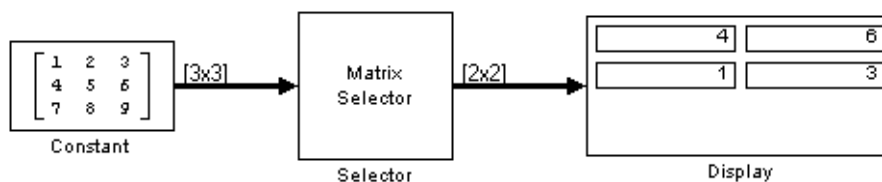
If you select external as the source for element indices, the block adds an input port for the element indices signal. The signal should specify the elements to be selected in the same way they are specified, using the **Elements** parameter.

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). For example, 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 matrix's second row and whose second row contains the first and third elements of the input matrix's first row.



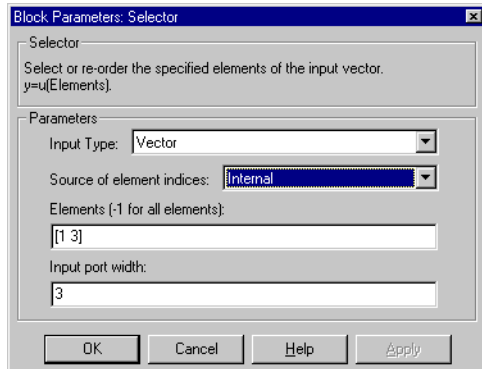
## Data Type Support

The Selector block accepts signals of any signal and data type, 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.

## Parameters and Dialog Box

The parameter dialog box appears as follows when vector input mode is selected.

# Selector



## Input Type

The type of the input signal: vector or matrix.

## 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.

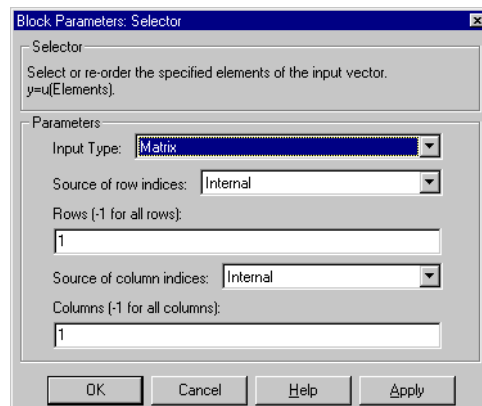
## Elements

The elements to be included in the output vector.

## Input port width

The number of elements in the input vector.

The dialog box appears as follows when matrix input mode is selected.



## **Input Type**

The type of the input signal: vector or matrix.

## **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.

## **Characteristics**

Sample Time	Inherited from driving block
Dimensionalized	Yes

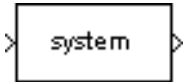
# S-Function

---

**Purpose** Access an S-function

**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 an M-file or MEX-file written as an S-function (see “Overview of S-Functions” in *Writing S-Functions* for information on how to create S-functions).



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.

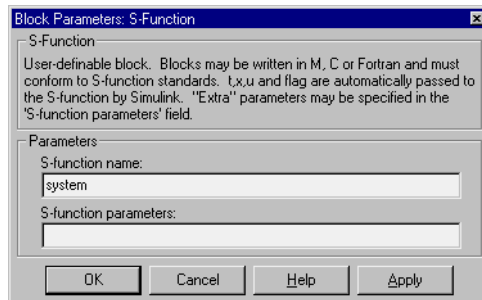
The S-Function block displays the name of the specified S-function and is always drawn with one input port and one output port, regardless of the number of inputs and outputs of the contained subsystem.

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 built-in data type except `int64` and `uint64`.

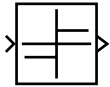
## Parameters and Dialog Box

See “S-Function Builder Dialog Box” in the online documentation for information on using the S-Function Builder block’s parameter dialog box.

**Purpose** Indicate the sign of the input

**Library** Simulink Math Operations and Fixed-Point Blockset Nonlinear

**Description** The Sign block indicates the sign of the input:

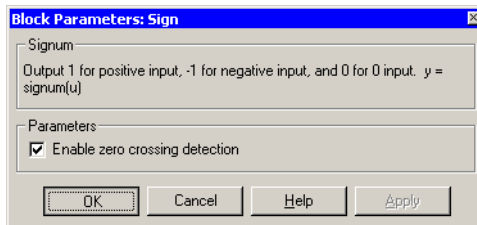


Sign

- 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 signals of any data type, 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).

### 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.

<b>Characteristics</b>	Dimensionalized	Yes
	Direct Feedthrough	Yes
	Sample Time	Inherited from the driving block
	Scalar Expansion	N/A

# Signal Builder

---

**Purpose** Create and generate interchangeable groups of signals whose waveforms are piecewise linear

**Library** Sources

**Description** The Signal Builder allows you to create interchangeable groups of piecewise linear signal sources and use them in a model. See “Working with Signal Groups” in *Using Simulink* for more information.



**Data Type Support** A Signal Builder block outputs a scalar or array of real signals of type double.

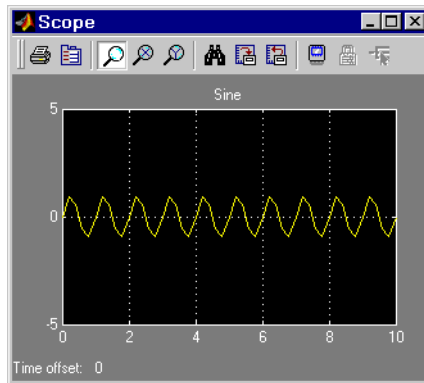
**Parameters and Dialog Box** The Signal Builder block has the same dialog box as that of a Subsystem block. To display the dialog box, select **Block Parameters** from the block’s context menu.

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

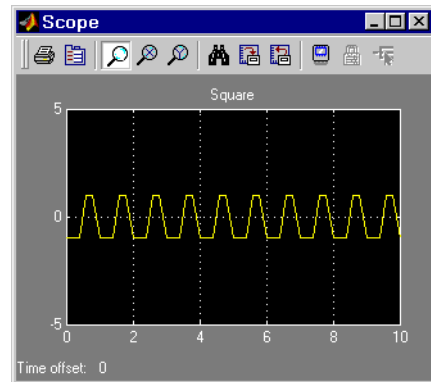
**Purpose** Generate various waveforms

**Library** Sources

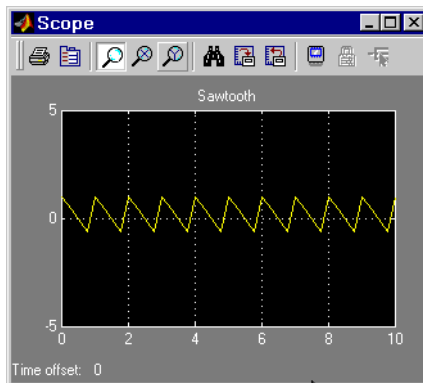
**Description** The Signal Generator block can produce one of three different waveforms: sine wave, square wave, and sawtooth wave. The signal parameters can be expressed in Hertz (the default) or radians per second. This figure shows each signal displayed on a Scope using default parameter values.



Sine Wave



Square Wave



Sawtooth Wave

# Signal Generator

---

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.

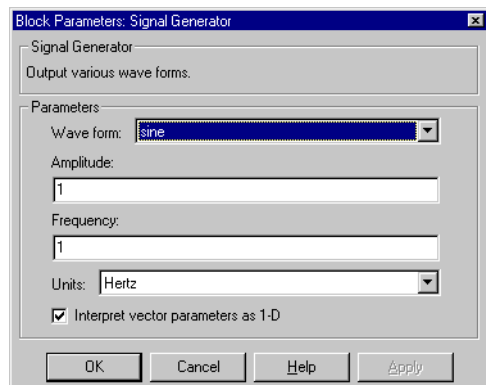
You can vary the output settings of the Signal Generator block while a simulation is in progress. This is useful to determine quickly the response of a system to different types of inputs.

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

A Signal Generator block outputs a scalar or array of real signals of type double.

## Parameters and Dialog Box



### 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.

**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	Of parameters
Dimensionalized	Yes
Zero Crossing	No

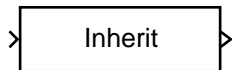
# Signal Specification

---

**Purpose** Verify that an input signal has the specified dimensions, sample time, data type, and numeric type

**Library** Signal Attributes

**Description** The Signal Specification block checks that the input signal has certain specified attributes. If so, the block outputs the input signal unchanged. Otherwise, it halts the simulation and displays an error message.



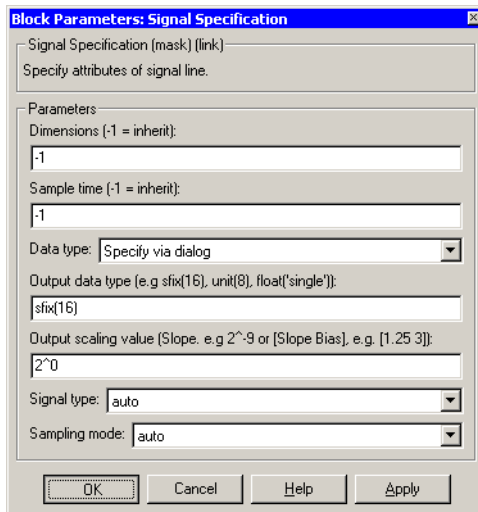
**Signal Specification** The Signal Specification block can be used as a mechanism to ensure that the attributes of a signal meet the desired attributes for certain sections of your model. For example, consider two people working on different parts of a model. The Signal Specification block is useful for indicating which attributes of various signals are needed by the different sections of the model. If there is a miscommunication and data types are changed unexpectedly, the attributes do not match and Simulink reports an appropriate error. Using the Signal Specification block helps ensure that you do not introduce unexpected problems into your models. If you are familiar with the assert mechanism in languages such as C, you can see that the Signal Specification block serves a similar purpose.

The Signal Specification block can also be used to ensure correct propagation of signal attributes throughout a model. Simulink's capability of allowing many attributes to be propagated from block to block is very powerful. However, when using user-written S-functions, 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** Accepts signals of any data type, including fixed-point data types but not int64 and uint64, that matches the data type specified by the **Data type** parameter.



## Parameters and Dialog Box



### Dimensions

Specify the dimensions that the input signal must match. Valid values are

- -1—Any dimension accepted
- n—Vector signal of width n accepted
- [m n]—Matrix signal having m rows and n columns accepted

### Sample Time

Specify the sample time that the input signal must match. Valid values are

- -1—Any sample time accepted
- 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 that the input signal must match. To accept any data type, set this parameter to auto.

# Signal Specification

---

## Output data type

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

## Output scaling value

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

## Signal type

Specify the numeric type (real or complex) that the input signal must match. To accept any numeric 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.

<b>Characteristics</b>	Dimensionalized	Yes
	Direct Feedthrough	Yes
	Sample Time	Continuous
	Scalar Expansion	No
	Zero Crossing	No

**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

$$y = Amplitude \times \sin(frequency \times time + phase) + bias$$

Time-based mode has two submodes: continuous mode or discrete mode. The value of the **Sample time** parameter determines whether the block operates in continuous mode or discrete mode:

- 0 (the default) causes the block to operate in continuous mode.
- >0 causes the block to operate in discrete mode.

See “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

# Sine Wave

---

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

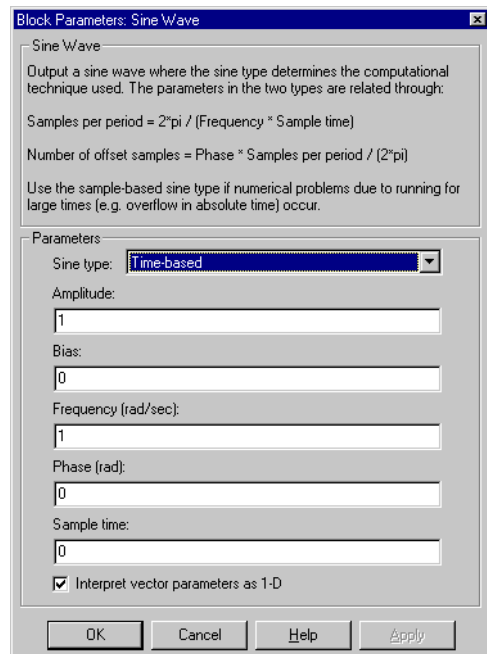
The block's numeric parameters must be of the same dimensions after scalar expansion. If the **Interpret vector parameters as 1-D** option is off, the block outputs a signal of the same dimensions and dimensionality as the parameters. If the **Interpret vector parameters as 1-D** option is on and the 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

A 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.

### 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.

<b>Characteristics</b>	Sample Time	Continuous, discrete, or inherited for time-based and discrete for sample-based
	Scalar Expansion	Of parameters
	Dimensionalized	Yes
	Zero Crossing	No

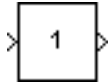
# Slider Gain

---

**Purpose** Vary a scalar gain using a slider

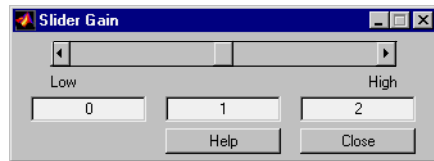
**Library** Math Operations

**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, Matrix Gain on page 2-153).

**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.



<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Of the gain
	States	0
	Dimensionalized	Yes
	Zero Crossing	No

# State-Space

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

**Library** Continuous

**Description** The State-Space block implements a system whose behavior is defined by

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

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.

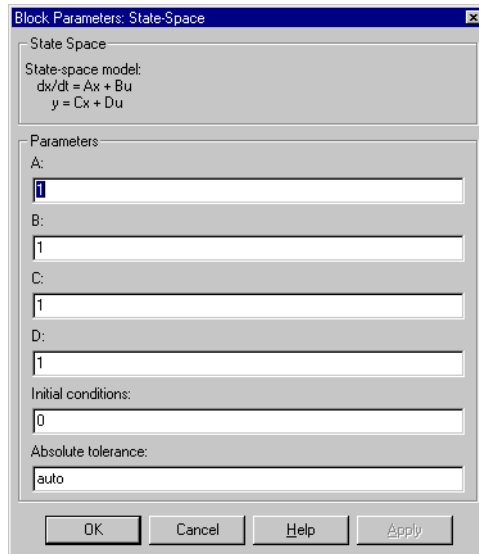
## Specifying the Absolute Tolerance for the Block's States

By default Simulink uses the absolute tolerance value specified in the **Simulation 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.

## 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 **Simulation Parameters** dialog box.

# State-Space

---

<b>Characteristics</b>	Direct Feedthrough	Only if $D \neq 0$
	Sample Time	Continuous
	Scalar Expansion	Of the initial conditions
	States	Depends on the size of A
	Dimensionalized	Yes
	Zero Crossing	No

**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

A Step block outputs real signals of type double.

### Parameters and Dialog Box

The dialog box titled "Block Parameters: Step" contains the following fields and options:

- Step**: Output a step.
- Parameters**:
  - Step time**: 1
  - Initial value**: 0
  - Final value**: 1
  - Sample time**: 0
  - Interpret vector parameters as 1-D
  - Enable zero crossing detection
- Buttons: OK, Cancel, Help, 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.

# Step

---

## **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	Inherited from driven block
Scalar Expansion	Of parameters
Dimensionalized	Yes

**Purpose** Stop the simulation when the input is nonzero

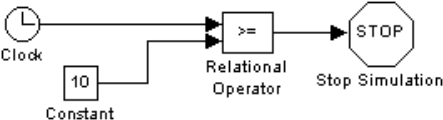
**Library** Sinks

**Description** The Stop Simulation block stops the simulation when the input is nonzero.



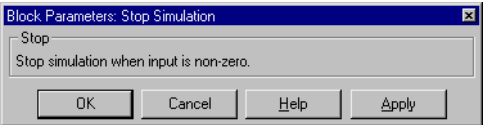
The simulation completes the current time step before terminating. If the block input is a vector, any nonzero vector element causes the simulation to stop.

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** A Stop Simulation block accepts real signals of type double or boolean.

**Dialog Box**



**Characteristics**

Sample Time	Inherited from driving block
Dimensionalized	Yes

# Subsystem, Atomic 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 (see “Atomic Versus Virtual Subsystems”), 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.

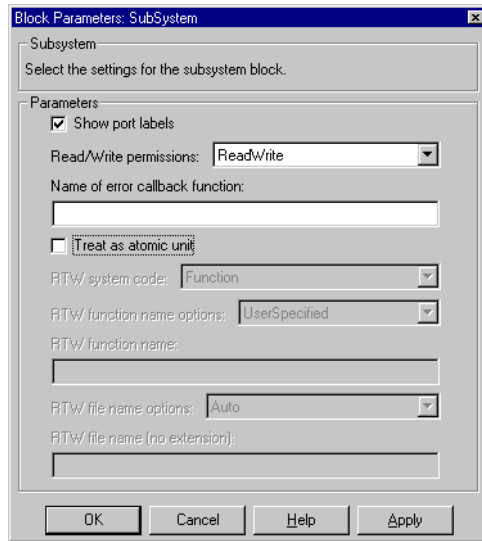
See “Creating Subsystems” for more information about subsystems.

## Data Type Support

A subsystem’s enable and trigger ports accept any data type, including fixed-point data types, except int64 and uint64. See Inport on page 2-174 for information on the data types accepted by a subsystem’s input ports. See Outport on page 2-238 for information on the data types output by a subsystem’s output ports.



## Parameters and Dialog Box



### Show port labels

Causes Simulink to display the labels of the subsystem's ports in the subsystem's icon.

### 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.

# Subsystem, Atomic Subsystem

---

## Access

Controls user access to the contents of the subsystem. You can select any of the following values.

Access	Description
ReadWrite	User can open and modify the contents of the subsystem.
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.

---

**Note** Parameters whose names begin with RTW are used by the Real-Time Workshop for code generation. See the Real-Time Workshop documentation for more information.

---

# Subsystem, Atomic Subsystem

---

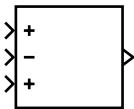
<b>Characteristics</b>	Sample Time	Depends on the blocks in the subsystem
	Dimensionalized	Depends on the blocks in the subsystem
	Zero Crossing	Yes, for enable and trigger ports if present

# Sum

**Purpose** Add or subtract inputs

**Library** Simulink Math Operations and Fixed-Point Blockset Math

## Description



Sum

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.

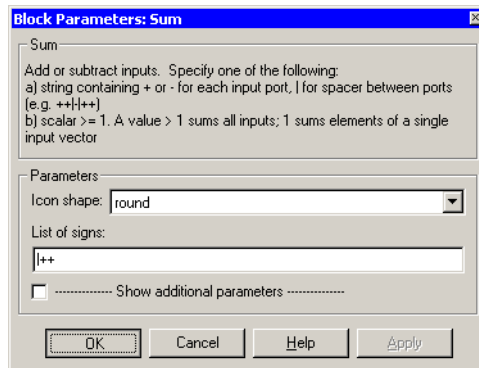
- A spacer character creates extra space between ports on the block’s icon.
- If only addition of all inputs is required, then a numeric parameter value equal to the number of inputs can be supplied instead of “+” characters.
- If only one vector is input, then a single “+” or “-” will collapse the vector using the specified operation.

When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to “Block Parameters” in the Fixed-Point Blockset documentation.

## Data Type Support

The Sum block accepts signals of any complexity and data type, including fixed-point data types, except int64 and uint64. The inputs may be of different data types unless the **Require all inputs to have same data type** parameter is selected.

## Parameters and Dialog Box



### Icon shape

Designate the icon shape of the block.

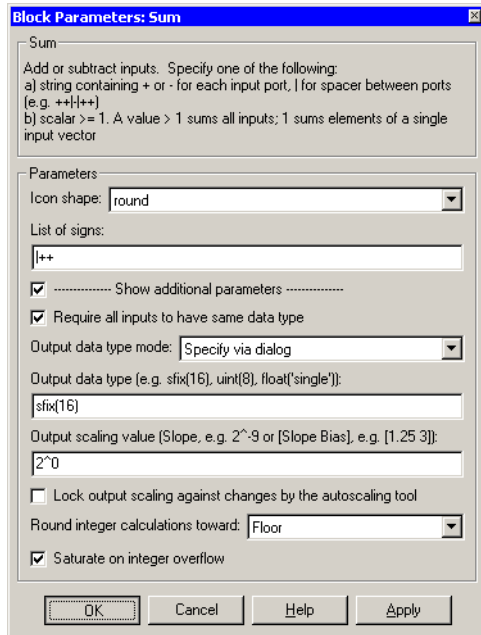
### List of signs

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 icon 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.

### Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.



## 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 **Specify via dialog** is selected for the **Output data type mode** parameter.

**Output scaling value**

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

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

If selected, scaling of outputs is locked. This parameter is only visible if Specify via dialog is selected for the **Output data type mode** parameter.

**Round integer calculations toward**

Select the rounding mode for fixed-point output.

**Saturate on integer overflow**

If selected, overflows saturate.

**Conversions and Operations**

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. Refer to “Rules for Arithmetic Operations” in the Fixed-Point Blockset documentation for more information about the rules that this block obeys when performing fixed-point operations.

**Characteristics**

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

# Switch

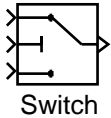
## Purpose

Switch output between the first input and the third input based on the value of the second input

## Library

Simulink Signal Routing and Fixed-Point Blockset Select

## 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*.

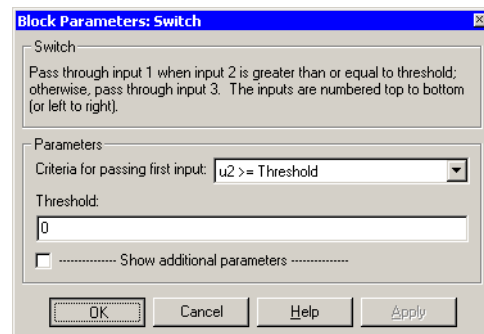
You select the conditions under which the first input is passed with the **Criteria for passing first input** parameter. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in the **Criteria for passing first input parameter**, then the first input is passed. Otherwise, the third input is passed.

When the **Show additional parameters** check box is selected, some of the parameters that become visible are common to many blocks. For a detailed description of these parameters, refer to “Block Parameters” in the Fixed-Point Blockset documentation.

## Data Type Support

A Switch block accepts real- or complex-valued signals of any data type except `int64` and `uint64` for data and control inputs. The data type of the threshold is `double`.

## Parameters and Dialog Box





### Criteria for passing first input

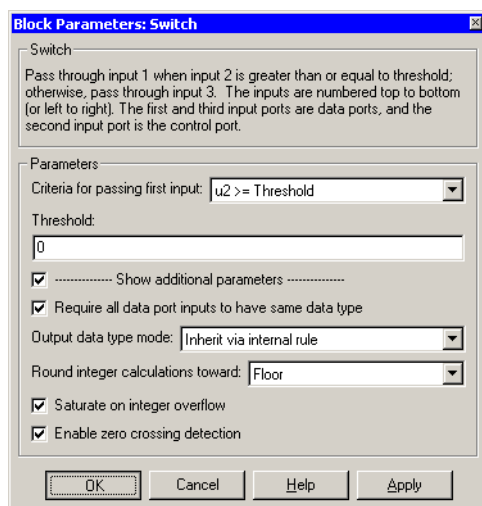
Select the conditions under which the first input is passed. You can make the block check whether the control input is greater than or equal to the threshold value, purely greater than the threshold value, or nonzero. If the control input meets the condition set in this parameter, then the first input is passed. Otherwise, the third input is passed.

### Threshold

Assign the switch threshold that determines which input is passed to the output.

### Show additional parameters

If selected, additional parameters specific to implementation of the block become visible as shown.



### Require all data port inputs to have same data type

Select to require all data inputs to have the same data type.

### Output data type mode

Choose to inherit the output data type and scaling by backpropagation or by an internal rule. The internal rule causes the output of the block to have the same data type and scaling as the input with the larger positive range.

# Switch

---

## **Round integer calculations toward**

Select the rounding mode for fixed-point output.

## **Saturate on integer overflow**

If selected, overflows saturate.

## **Enable zero crossing detection**

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

## **Characteristics**

Dimensionalized	Yes
Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	Yes
Zero Crossing	No, unless <b>Enable zero crossing detection</b> is selected

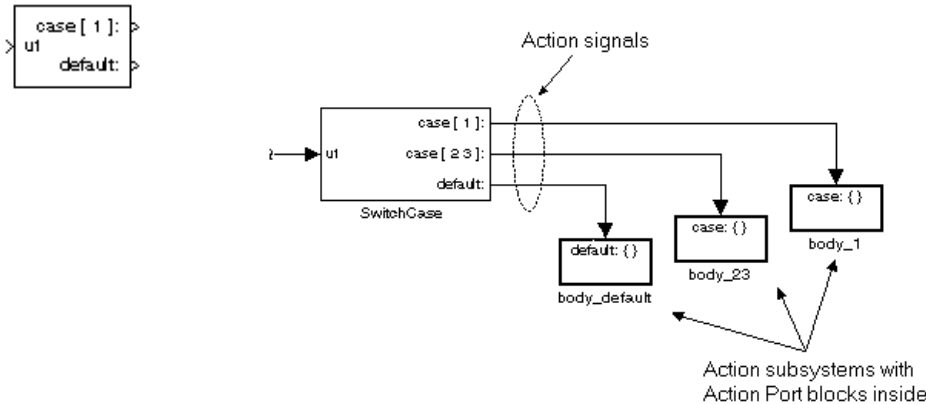
## **See Also**

Multi-Port 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.

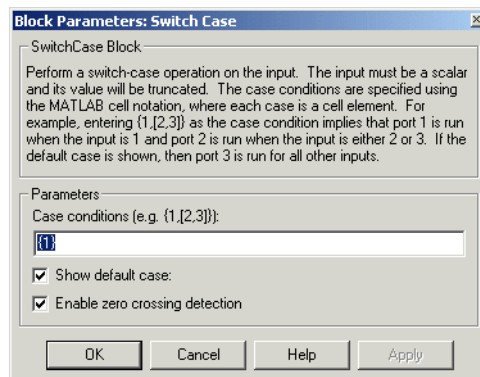
**Note** As demonstrated in the preceding pseudocode example, cases for the Switch Case block contain an implied break after their Switch Case Action subsystems are executed. There is no fall-through behavior for the Simulink switch control flow statement as found in standard C switch statements.

## Data Type Support

Input to the port labeled u1 of a Switch Case block can be a scalar value of any data type, including fixed-point data types, except boolean, int64, and uint64. Noninteger inputs are truncated.

Data outputs are action signals to Switch Case Action subsystems that are created with Action Port blocks and subsystems.

## Parameters and Dialog Box



## Case conditions

Case conditions are specified using MATLAB cell notation where each cell is a case condition consisting of integers or arrays of integers. In the preceding dialog example, entering `{1, [7,9,4]}` specifies that output port case[1] is run when the input value is 1, and output port case[7 9 4] is run when the input value is 7, 9, or 4.

You can use colon notation to specify a range of case conditions. For example, entering `{[1:5]}` specifies that output port case[1 2 3 4 5] is run when the input value is 1, 2, 3, 4, or 5.

Depending on block size, cases with long lists of conditions are displayed in shortened form in the Switch Case block, using a terminating ellipsis (...).

# Switch Case

---

## Show default case

If this check box is selected, 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.

## Characteristics

Direct Feedthrough	Yes
Sample Time	Inherited from driving block
Scalar Expansion	No
Dimensionalized	No
Zero Crossing	Yes, if zero-crossing detection is enabled.

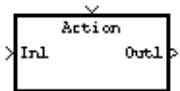
# Switch Case Action Subsystem

---

**Purpose** Represent a subsystem whose execution is triggered by a Switch Case block

**Library** Ports & Subsystems

**Description** This block is a Subsystem block that is preconfigured to serve as a starting point for creating a subsystem whose execution is triggered by a Switch Case block. See the Switch Case block and “Control Flow Blocks” for more information.



# Terminator

---

**Purpose** Terminate an unconnected output port

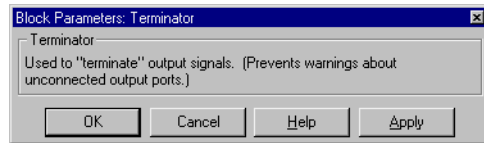
**Library** Sinks

**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** A Terminator block accepts real or complex signals of any data type, including fixed-point data types, except int64 and uint64.

**Parameters and Dialog Box**



**Characteristics** Sample Time                      Inherited from driving block  
Dimensionalized                      Yes



**Purpose** Generate linear models in the base workspace at specific times

**Library** Model-Wide Utilities

**Description** This block calls `linmod` or `dlinmod` to create a linear model for the system when the simulation clock reaches the time specified by the **Linearization time** parameter. No trimming is performed. The linear model is stored in the base workspace as a structure, along with information about the operating point at which the snapshot was taken. Multiple snapshots are appended to form an array of structures.

T=1

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 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> ).
Ts	The sample time of the linearization for a discrete linearization

# Time-Based Linearization

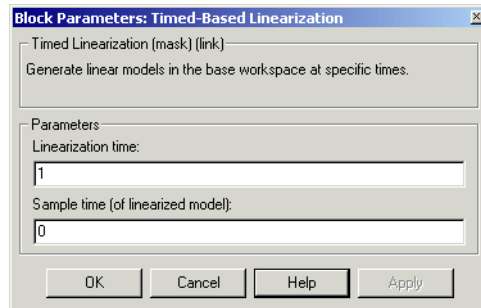
---

Use the Trigger-Based Linearization block if you need to generate linear models conditionally.

## Data Type Support

Not applicable.

## Parameters and Dialog Box



### Linearization time

Time at which you want the block to generate a linear model. Enter a vector of times if you want the block to generate linear models at more than one time step.

### Sample time (of linearized model)

Specify a sample time to create discrete-time linearizations of the model (see “Discrete-Time System Linearization” on page 3-3).

## Characteristics

Sample Time	Inherited from driving block
Dimensionalized	No

**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.

$$\begin{bmatrix} t_1 & t_2 & \dots & t_{final} \\ u_{1_1} & u_{1_2} & \dots & u_{1_{final}} \\ \dots & & & \\ u_{n_1} & u_{n_2} & \dots & u_{n_{final}} \end{bmatrix}$$

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. The block 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.

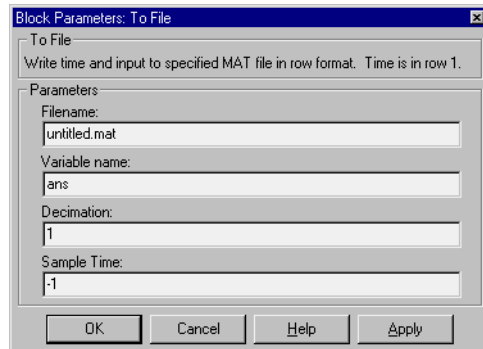
If the file exists at the time the simulation starts, the block overwrites its contents.

**Data Type Support**

A To File block accepts real signals of type `double`.

# To File

## 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 MATLAB's 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.

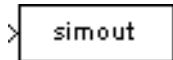
## Characteristics

Sample Time	Inherited from driving block
Dimensionalized	Yes

**Purpose** Write data to the workspace

**Library** Sinks

## Description



The To Workspace block writes its input to the workspace. The block writes its output to an array or structure that has the name specified by the block's **Variable name** parameter. The **Save format** parameter determines the output format.

## Array

Selecting this option causes the To Workspace block to save the input as an N-dimensional array where N is one more than the number of dimensions of the input signal. For example, if the input signal is a 1-D array (i.e., a vector), the resulting workspace array is two-dimensional. If the input signal is a 2-D array (i.e., a matrix), the array is three-dimensional.

The way samples are stored in the array depends on whether the input signal is a scalar or vector or a matrix. If the input is a scalar or a vector, each input sample is output as a row of the array. For example, suppose that the name of the output array is `simout`. Then, `simout(1, :)` corresponds to the first sample, `simout(2, :)` corresponds to the second sample, etc. If the input signal is a matrix, the third dimension of the workspace array corresponds to the values of the input signal at specified sampling point. For example, suppose again that `simout` is the name of the resulting workspace array. Then, `simout(:, :, 1)` is the value of the input signal at the first sample point; `simout(:, :, 2)` is the value of the input signal at the second sample point; etc.

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

- The **Limit data points to last** parameter indicates how many sample points to save. If the simulation generates more data points than the specified maximum, the simulation saves only the most recently generated samples. To capture all the data, set this value to `inf`.
- The **Decimation** parameter allows you to write data at every *n*th sample, where *n* is the decimation factor. The default decimation, 1, writes data at every time step.

# To Workspace

---

- 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. The block icon shows the name of the array to which the data is written.

## Structure

This format consists of a structure with three fields: `time`, `signals`, and `blockName`. The `time` field is empty. The `blockName` field contains the name of the To Workspace block. The `signals` field contains a structure with three fields: `values`, `dimensions`, and `label`. The `values` field contains the array of signal values. The `dimensions` field specifies the dimensions of the values array. The `label` field contains the label of the input line.

## Structure with Time

This format is the same as Structure except that the `time` field contains a vector of simulation time steps.

## Using Saved Data with a From Workspace Block

If the data written using a To Workspace block is intended to be played back in another simulation using a From Workspace block, use the Structure with Time format to save the data.

## Examples

In a simulation where the start time is 0, the **Maximum number of sample points** is 100, the **Decimation** is 1, and the **Sample time** is 0.5. The To Workspace block collects a maximum of 100 points, at time values of 0, 0.5, 1.0, 1.5, ..., seconds. Specifying a **Decimation** value of 1 directs the block to write data at each step.

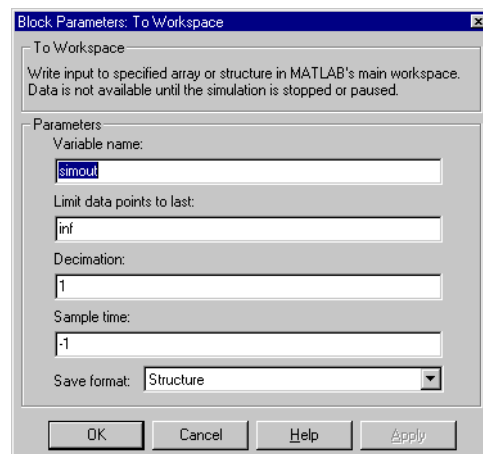
In a similar example, the **Maximum number of sample points** is 100 and the **Sample time** is 0.5, but the **Decimation** is 5. In this example, the block collects up to 100 points, at time values of 0, 2.5, 5.0, 7.5, ..., seconds. Specifying a **Decimation** value of 5 directs the block to write data at every fifth sample. The sample time ensures that data is written at these points.

In another example, all parameters are as defined in the first example except that the **Limit data points to last** is 3. In this case, only the last three sample points collected are written to the workspace. If the simulation stop time is 100, data corresponds to times 99.0, 99.5, and 100.0 seconds (three points).

## Data Type Support

A To Workspace block can save real or complex inputs of any data type to the MATLAB workspace except int64 and uint64. This includes fixed-point data types.

## 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.

# To Workspace

---

## **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.

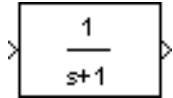
<b>Characteristics</b>	Sample Time	Inherited
	Dimensionalized	Yes



**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 Block Icon

The numerator and denominator are displayed on the Transfer Fcn block icon 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

# Transfer Fcn

**Denominator** as (den) where den is [7,5,3,1], the block icon looks like this:

$$\frac{3s^2+2s+1}{7s^3+5s^2+3s+1}$$

- If each is specified as a variable, the icon shows the variable name followed by (s). For example, if you specify **Numerator** as num and **Denominator** as den, the block icon looks like this:

$$\frac{\text{num}(s)}{\text{den}(s)}$$

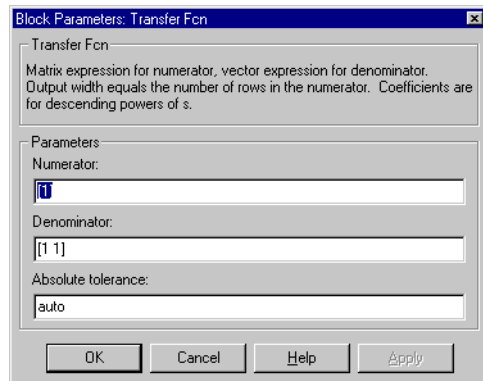
## Specifying the Absolute Tolerance for the Block's States

By default Simulink uses the absolute tolerance value specified in the **Simulation 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

A Transfer Fcn block accepts and outputs signals of type double.

## Parameters and Dialog Box



## Numerator

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

## Denominator

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

## Absolute tolerance

Absolute tolerance used to solve the block's states. You can enter auto or a numeric value. If you enter auto, Simulink determines the absolute tolerance (see "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 **Simulation 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

# 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  $t - t_{delay}$ .

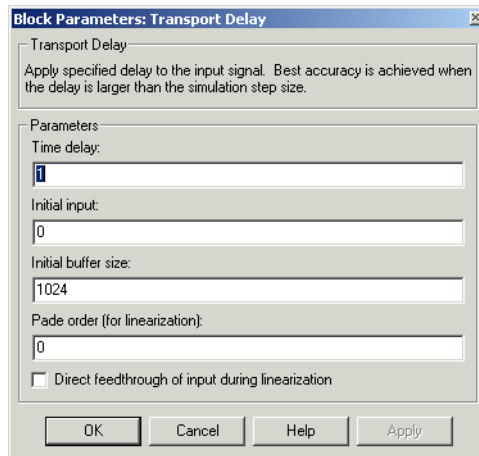
This block differs from the Unit Delay block, which delays and holds the output on sample hits only.

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 *Using Simulink*.

## Data Type Support

A 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

---

<b>Characteristics</b>	Direct Feedthrough	No
	Sample Time	Continuous
	Scalar Expansion	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 *Using Simulink*.

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

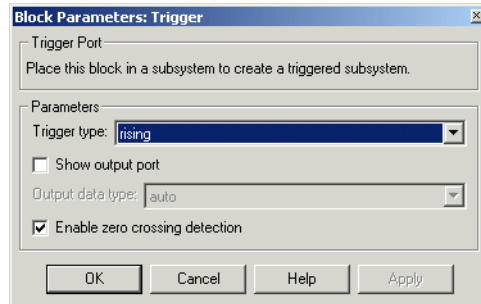
- 1 for a signal that causes a rising trigger
- -1 for a signal that causes a falling trigger
- 0 otherwise

### Data Type Support

A Trigger block accepts signals of any data type except int64 and uint64.

# Trigger

## Parameters and Dialog Box



### Trigger type

The type of event that triggers execution of the subsystem.

### 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.

### Enable zero crossing detection

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

## Characteristics

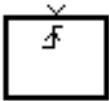
Sample Time	Determined by the signal at the trigger port
Dimensionalized	Yes



**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.

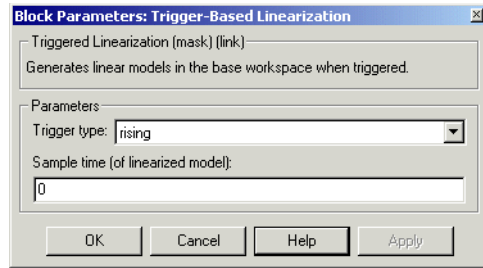
# Trigger-Based Linearization

---

## Data Type Support

The trigger port accepts signals of any data type except int64 and uint64.

## Parameters and Dialog Box



### Trigger type

Type of event on the trigger input signal that triggers generation of a linear model. See the **Trigger type** parameter of the Trigger block for an explanation of the various trigger types that you can select.

### Sample time (of linearized model)

Specify a sample time to create a discrete-time linearization of the model (see “Discrete-Time System Linearization” on page 3-3).

## Characteristics

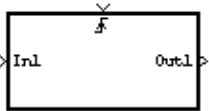
Sample Time	Inherited from driving block
Dimensionalized	No

# Triggered Subsystem

**Purpose** Represent a subsystem whose execution is triggered by external input

**Library** Ports & Subsystems

**Description** This block is a Subsystem block that is preconfigured to serve as the starting point for creating a triggered subsystem (see “Triggered Subsystems”).



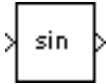
# Trigonometric Function

---

**Purpose** Perform a trigonometric function

**Library** Math Operations

**Description** The Trigonometric Function block performs numerous common trigonometric functions.



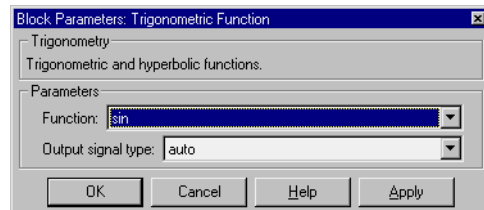
You can select one of these functions from the **Function** list: sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, and tanh. The block output is the result of the operation of the function on the input or inputs.

The name of the function appears on the block icon. 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** A 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.

<b>Characteristics</b>	Direct Feedthrough	Yes
	Sample Time	Inherited from driving block
	Scalar Expansion	Of the input when the function requires two inputs
	Dimensionalized	Yes
	Zero Crossing	No

# Uniform Random Number

---

**Purpose** Generate uniformly distributed random numbers

**Library** Sources

## Description



The Uniform Random Number block generates uniformly distributed random numbers over a specifiable interval with a specifiable starting seed. The seed is reset each time a simulation starts. The generated sequence is repeatable and can be produced by any Uniform Random Number block with the same seed and parameters. To generate normally distributed random numbers, use the Random Number block.

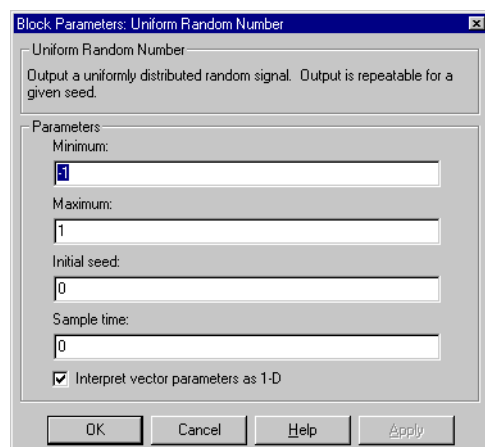
Avoid integrating a random signal, because solvers are meant to integrate relatively smooth signals. Instead, use the Band-Limited White Noise block.

The block's numeric parameters must be of the same dimensions after scalar expansion. If the **Interpret vector parameters as 1-D** option is off, the block outputs a signal of the same dimensions and dimensionality as the parameters. If the **Interpret vector parameters as 1-D** option is on and the numeric parameters are row or column vectors (i.e., single row or column 2-D arrays), the block outputs a vector (1-D array) signal; otherwise, the block outputs a signal of the same dimensions as the parameters.

## Data Type Support

A Uniform Random Number block outputs a real signal of type double.

## Parameters and Dialog Box



**Minimum**

The minimum of the interval. The default is -1.

**Maximum**

The maximum of the interval. The default is 1.

**Initial seed**

The starting seed for the random number generator. The default is 0.

**Sample time**

The sample period. The default is 0. See “Specifying Sample Time” in the online documentation for more information.

**Interpret vector parameters as 1-D**

If selected, column or row matrix values for the Step block’s numeric parameters result in a vector output signal; otherwise, the block outputs a signal of the same dimensionality as the parameters. If this option is not selected, the block always outputs a signal of the same dimensionality as the block’s numeric parameters.

**Characteristics**

Sample Time	Continuous, discrete, or inherited
Scalar Expansion	No
Dimensionalized	Yes
Zero Crossing	No

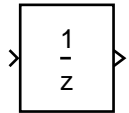
# Unit Delay

---

**Purpose** Delay a signal one sample period

**Library** Simulink Discrete and Fixed-Point Blockset Delays & Holds

## Description



Unit Delay

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.

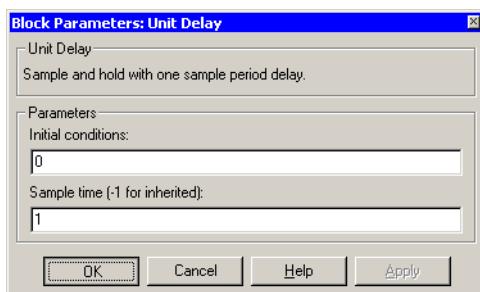
---

## Data Type Support

The Unit Delay block accepts real or complex signals of any data type except `int64` and `uint64`, including fixed-point data types. If the data type of the input signal is user-defined, the initial condition must be zero.



## 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.

### Sample time

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.

## Conversions and Operations

The **Initial conditions** parameter is converted from a double to the input data type offline using round-to-nearest and saturation.

## Characteristics

Dimensionalized	Yes
Direct Feedthrough	No
Sample Time	Discrete or continuous. When inheriting a continuous signal, this block acts as a Simulink Memory block.
Scalar Expansion	Of input or initial conditions
States	Yes—inherited from driving block for nonfixed-point data types.
Zero Crossing	No

# Variable Transport Delay

---

**Purpose** Delay the input by a variable amount of time

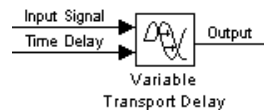
**Library** Continuous

## Description



The Variable Transport Delay block can be used to simulate a variable time delay. The block might be used to model a system with a pipe where the speed of a motor pumping fluid in the pipe is variable.

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.

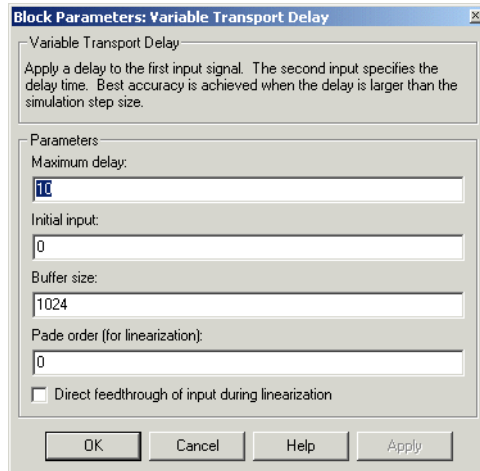
When output is required at a time that does not correspond to the times of the stored input values, the block interpolates linearly between points. If the time delay is smaller than the step size, the block extrapolates an output point. This can result in less accurate results. The block cannot use the 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

A 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.

# Variable Transport Delay

---

<b>Characteristics</b>	Direct Feedthrough	Yes, of the time delay (second) input
	Sample Time	Continuous
	Scalar Expansion	Of input and all parameters except <b>Buffer size</b>
	Dimensionalized	Yes
	Zero Crossing	No

## Purpose

Implement a C-like while or do-while control flow statement as a While subsystem

## Library

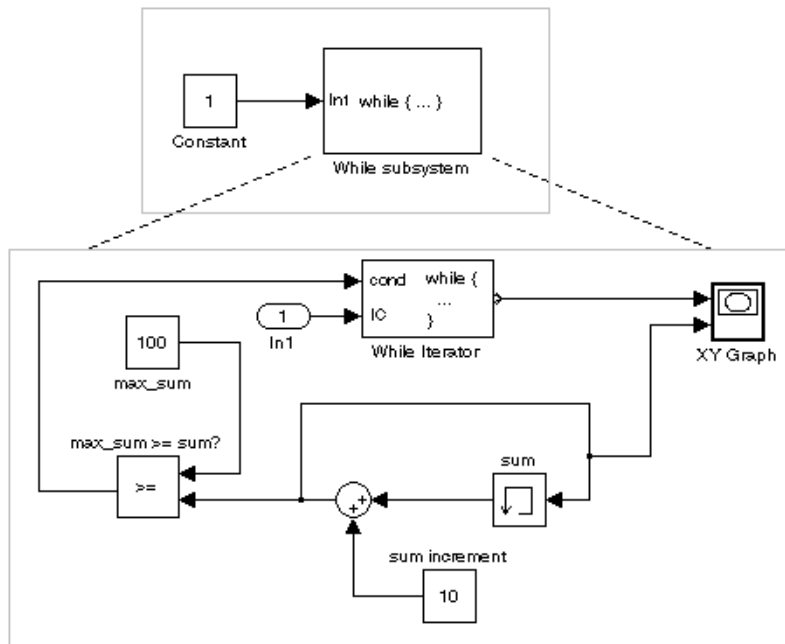
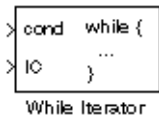
Ports & Subsystems/While Subsystem

## Description

The While Iterator block, when placed in a subsystem, implements a C-like while or do-while control flow statement in Simulink as a While subsystem. It has iterative control over any accompanying Simulink block programming placed in the same subsystem with it.

For each iteration of the While Iterator block, the accompanying blocks perform one execution. Iteration takes place as long as the input conditions are true. This applies to an initial condition for the first execution (input port labeled IC) and a condition for succeeding executions (input port labeled cond).

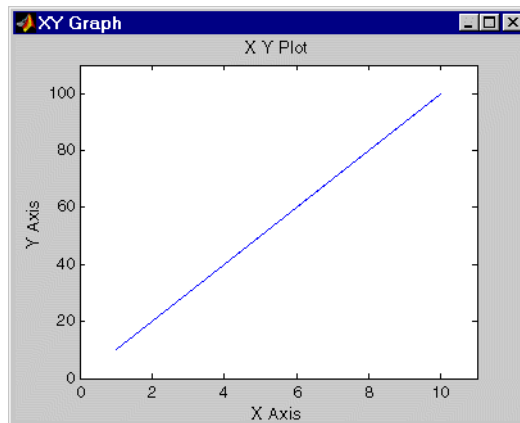
The following While subsystem example increments an initial value of 0 by 10 for every execution.



# While Iterator

In the preceding example, a subsystem with a While block receives an input, which it passes to the IC (initial condition) port of the While block inside. If this value is true, the While block executes the blocks of the subsystem it is in. Since this value is 1 (true), the blocks execute and a value of 10 is added to a sum, which is initially 0. The sum is then compared to a value of 100. If the sum is less than or equal to 100, a value of true is passed to the While block through the cond (condition) port. This causes the blocks to execute again and again until the value passed to the While Iterator block is false and execution ceases.

In addition, for each time the blocks of the subsystem execute, the While block outputs a value equal to the number of times that the blocks have executed, including the current execution. This value, along with the sum value, is sent to the  $x$  and  $y$  coordinate inputs, respectively, of an XY Graph block with the following result.



Points:  
(1,10)  
(2,20)  
etc.

The preceding while control flow statement example can be represented by the following pseudocode.

```
sum = 0;  
IC = 1;  
iteration_number = 0;  
cond = IC;  
while (cond != 0) {  
    iteration_number = iteration_number + 1;  
    sum = sum + sum_increment;  
}
```

```
    if (sum >= 100 OR iterations > max_iterations) cond = 0;
}
```

You construct a Simulink While subsystem like the preceding example as follows:

- 1 Place a While Iterator block in a subsystem.  
This changes the subsystem icon to `while{...}`.  
You can use an ordinary subsystem or an atomic subsystem. In either case, the resulting While subsystem is atomic.
- 2 The source for the initial condition (labeled IC) data input that controls the first execution must be outside the While subsystem.
- 3 Input to the cond port controls any succeeding executions and must originate from within the Simulink programming residing in the While subsystem.
- 4 Open the **Block Parameters** dialog of the While block and enter as follows:

- You can limit the number of iterations of the While Iterator block by setting the **Maximum number of iterations** field.

You can set this field to a positive value to limit iterations and avoid infinite loops, or you can set this property to -1 if you want the While Iterator block to iterate without limit until an iteration condition (IC or cond port) is false (=0). This is done in the preceding example.

- If you want an optional iterator output port, select the **Show iterations number port** check box.

This port outputs 1 for the first execution, which is incremented by 1 for each succeeding execution. If you select the **Show iterations number port** check box the **Output data type** field is enabled (it is grayed out otherwise). This allows you to select the type for the iteration number data output. The default type is `int32`.

- You can specify a do-while iteration instead of a while iteration by selecting **do-while** for the **While loop type** field.

The equivalent pseudocode for a do-while iteration in the preceding example (minus the IC port) is as follows:

```
sum = 0;
iteration_number = 0;
cond = 1;
```

# While Iterator

---

```
do {
    iteration_number = iteration_number + 1;
    sum = sum + sum_increment;
    if (sum >= 100 AND iterations > max_iterations) cond = 0;
} while (cond);
```

When you change the While Iterator block to do-while operation, the IC (initial condition) input disappears from the While Iterator block. The important distinction between the while and do-while modes of the While Iterator block is that the do-while mode runs the While subsystem at least once. In while mode, the While subsystem might not run its blocks at all, depending on the value of the initial condition (IC).

- The setting of the **States when starting** field applies only to cases in which the While subsystem is called repeatedly. If you set this field to reset (the default value), every time the While subsystem is called, its states are reset to their initial values. If you set the **States when starting** field to held, the states of the While subsystem are retained between calls. In the preceding example, the While subsystem is called only once.

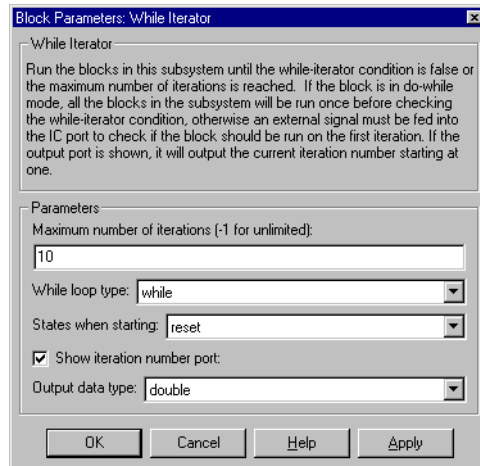
## Data Type Support

Acceptable data inputs for the condition ports are any type except int64 and uint64 that includes a 0 value (includes fixed-point data).

You can select the data output type for the iterator output port in the parameter dialog box of the While block as double, int32, int16, or int8.



## Parameters and Dialog Box



### Maximum number of iterations

The maximum number of iterations allowed. In the pseudocode examples, this value is represented as `max_iterations`. If you set this value to -1, the resulting While subsystem iterates as long as the input conditions (IC and cond ports) allow.

### While loop type

Specifies while or a do-while operation for the While subsystem.

### States when starting

Set this field to `reset` if you want the variables of the While subsystem to be reinitialized for each iteration. Otherwise, set this field to `held` (the default) to make sure that the While subsystem states retain their values from one call to another.

### Show iteration number port

If this check box is selected (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 the **Show iteration number port** check box is selected (the default), this field is enabled. It sets the data type of the iteration number output to `int32`, `int16`, `int8`, or `double`.

# While Iterator

---

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

# While Iterator Subsystem

---

**Purpose** Represent a subsystem that executes repeatedly while a condition is satisfied during a simulation time step

**Library** Ports & Subsystems

**Description** This 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
```

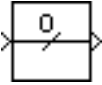
# Width

---

**Purpose** Output the width of the input vector

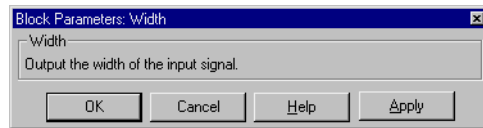
**Library** Signal Attributes

**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, including fixed-point data types, except int64 and uint64. The Width block supports mixed-type signal vectors. This block outputs real signals of type double.

## Parameters and Dialog Box



<b>Characteristics</b>	Sample Time	Constant
	Dimensionalized	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.



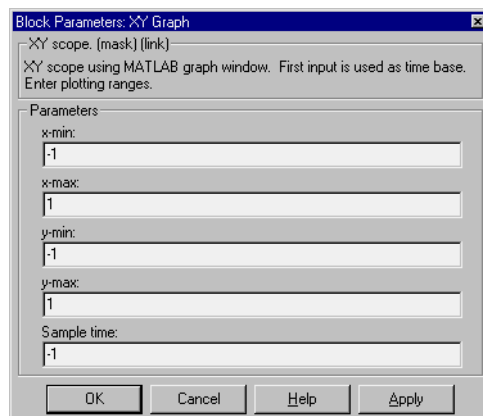
The block has two scalar inputs. The block plots data in the first input (the  $x$  direction) against data in the second input (the  $y$  direction). This block is useful for examining limit cycles and other two-state data. Data outside the specified range is not displayed.

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** An XY Graph block accepts real signals of type `double`.

## Parameters and Dialog Box



### **x-min**

The minimum  $x$ -axis value. The default is `-1`.

### **x-max**

The maximum  $x$ -axis value. The default is `1`.

# XY Graph

---

## **y-min**

The minimum  $y$ -axis value. The default is -1.

## **y-max**

The maximum  $y$ -axis value. The default is 1.

## **Sample time**

The time interval between samples. The default is -1, which means that the sample time is determined by the driving block. See “Specifying Sample Time” in the online documentation for more information.

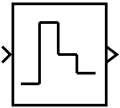
## **Characteristics**

Sample Time	Inherited from driving block
States	0

**Purpose** Implement a zero-order hold of one sample period

**Library** Simulink Discrete and Fixed-Point Blockset Delays & Holds

## Description



Zero-Order Hold

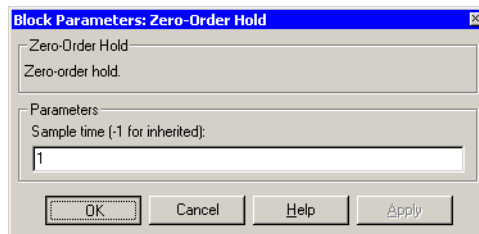
The Zero-Order Hold block samples and holds its input for the specified sample period. The block accepts one input and generates one output, both of which can be scalar or vector. If the input is a vector, all elements of the vector are held for the same sample period.

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 except int64 and uint64, including fixed-point data types.

## Parameters and Dialog Box



### Sample time

Specify the time between samples. A value of -1 means the sample time is inherited. See “Specifying Sample Time” in the online documentation for more information.

# Zero-Order Hold

---

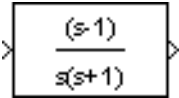
<b>Characteristics</b>	Dimensionalized	Yes
	Direct Feedthrough	Yes
	Sample Time	Discrete
	Scalar Expansion	No
	Zero Crossing	No



**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.

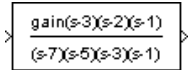
Block input and output widths are equal to the number of rows in the zeros matrix.

### The Zero-Pole Block Icon

The Zero-Pole block displays the transfer function in its icon depending on how the parameters are specified:

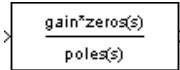
- If each is specified as an expression or a vector, the icon shows the transfer function with the specified zeros, poles, and gain. If you specify a variable in parentheses, the variable is evaluated.

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.


$$\frac{\text{gain} * \text{zeros}(s)}{\text{poles}(s)}$$

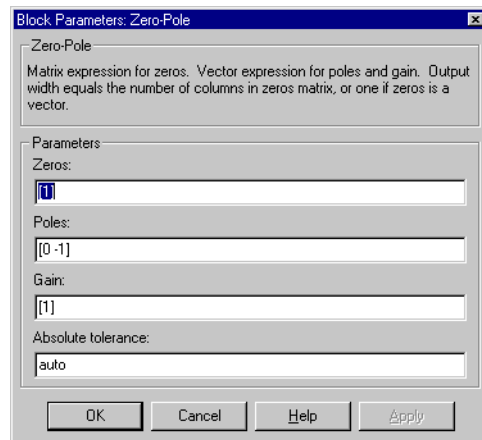
## Specifying the Absolute Tolerance for the Block's States

By default, Simulink uses the absolute tolerance value specified in the **Simulation 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

A 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 **Simulation Parameters** dialog box.

<b>Characteristics</b>	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

# Zero-Pole

---

# 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.

# linmod, dlinmod, linmod2

---

**Purpose** Extract the linear state-space model of a system around an operating point

**Syntax**

```
[A,B,C,D] = linfun('sys', x, u)
[num,den] = linfun('sys', x, u)
sys_struct = linfun('sys', x, u)
```

**Arguments**

<i>linfun</i>	linmod, dlinmod, or linmod2.
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.

**Description** linmod obtains linear models from systems of ordinary differential equations described as Simulink models. linmod returns the linear model in state-space form,  $A, B, C, D$ , which describes the linearized input-output relationship.

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx + Du\end{aligned}$$

Inputs and outputs are denoted in Simulink block diagrams using Inport and Outport blocks.

$[A,B,C,D] = \text{linmod}('sys', x, u)$  obtains the linearized model of sys around an operating point with the specified state variables x and the input u. If you omit x and u, the default values are zero.

$[\text{num},\text{den}] = \text{linfun}('sys', x, u)$  returns the linearized model in transfer function form.

$\text{sys\_struc} = \text{linfun}('sys', x, u)$  returns a structure that contains the linearized model, including state names, input and output names, and information about the operating point.

## 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.

# linmod, dlinmod, linmod2

---

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 pert to a two-element vector, where the second element is used to set the value of t at which to obtain the linear model.

The ordering of the states from the nonlinear model to the linear model is maintained. For Simulink systems, a string variable that contains the block name associated with each state can be obtained using

```
[sizes,x0,xstring] = sys
```

where xstring is a vector of strings whose ith row is the block name associated with the ith state. Inputs and outputs are numbered sequentially on the diagram.

For single-input multioutput 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.

Linearizing a model that contains Derivative or Transport Delay blocks can be troublesome (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  $x_0$ ,  $u_0$ ,  $y_0$  that satisfies a specified set of state, input, and/or output conditions. The integer vectors  $ix$ ,  $iu$ , and  $iy$  select the values in  $x_0$ ,  $u_0$ , and  $y_0$  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,  $dx_0$  specifies the state derivative values at the search's starting point and  $idx$  selects the values in  $dx_0$  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

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

The  $A$ ,  $B$ ,  $C$ , and  $D$  matrices are as follows in a system called `sys`.

$$A = [-0.09 \quad -0.01; \quad 1 \quad 0];$$

$$B = [ \quad 0 \quad -7; \quad 0 \quad -2];$$

$$C = [ \quad 0 \quad 2; \quad 1 \quad -5];$$

$$D = [-3 \quad 0; \quad 1 \quad 0];$$

### Example 1

To find an equilibrium point, use

```
[x,u,y,dx,options] = trim('sys')
```

```
x =
    0
    0
u =
    0
y =
    0
    0
dx =
    0
    0
```

The number of iterations taken is

```
options(10)
ans =
     7
```

## Example 2

To find an equilibrium point near  $x = [1;1]$ ,  $u = [1;1]$ , enter

```
x0 = [1;1];
u0 = [1;1];
[x,u,y,dx,options] = trim('sys', x0, u0);

x =
    1.0e-11 *
   -0.1167
   -0.1167
u =
    0.3333
    0.0000
y =
   -1.0000
    0.3333
dx =
    1.0e-11 *
    0.4214
    0.0003
```

The number of iterations taken is

```
options(10)
ans =
    25
```

## Example 3

To find an equilibrium point with the outputs fixed to 1, use

```
y = [1;1];
iy = [1;2];
[x,u,y,dx] = trim('sys', [], [], y, [], [], iy)

x =
    0.0009
   -0.3075
```

```
u =
    -0.5383
     0.0004
y =
     1.0000
     1.0000
dx =
     1.0e-16 *
    -0.0173
     0.2396
```

**Example 4**

To find an equilibrium point with the outputs fixed to 1 and the derivatives set to 0 and 1, use

```
y = [1;1];
iy = [1;2];
dx = [0;1];
idx = [1;2];
[x,u,y,dx,options] = trim('sys',[],[],y,[],[],iy,dx,idx)

x =
     0.9752
    -0.0827
u =
    -0.3884
    -0.0124
y =
     1.0000
     1.0000
dx =
     0.0000
     1.0000
```

The number of iterations taken is

```
options(10)
ans =
     13
```

# 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 describes commands that you can use in programs that create or modify models.

Task-Oriented List of Commands (p. 4-2)	List of commands arranged by tasks to be performed
Specifying Parameters and Object Paths (p. 4-4)	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.

<b>Task</b>	<b>Command</b>
Create a new Simulink system.	<code>new_system</code>
Open an existing system.	<code>open_system</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>
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>
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>
Get a parameter value.	<code>get_param</code>
Set parameter values.	<code>set_param</code>
Delete a system parameter.	<code>delete_param</code>
Get the pathname of the current block.	<code>gcb</code>
Get the pathname of the current system.	<code>gcs</code>
Get the handle of the current block.	<code>gcbh</code>
Get the name of the root-level system.	<code>bdroot</code>
Open the Simulink block library.	<code>simulink</code>



---

<b>Task</b>	<b>Command</b>
Discretize a model.	sldiscmdl
Open the Model Discretizer GUI	slmdliscui
Compare two models.	compare_model

# 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. Appendix , “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')
```

# 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`

**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, and State ports are identified by name, such as `'subsystem_name/Enable'`, `'subsystem_name/Trigger'`, or `'Integrator/State'`.

`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.

# add\_line

---

## Examples

This command adds a line to the mymodel system connecting the output of the Sine Wave block to the first input of the Mux block.

```
add_line('mymodel','Sine Wave/1','Mux/1')
```

This command adds a line to the mymodel system extending from (20,55) to (40,10) to (60,60).

```
add_line('mymodel',[20 55; 40 10; 60 60])
```

## See Also

`delete_line`

- Purpose** Add a parameter to a Simulink system
- Syntax** `add_param('sys','parameter1',value1,'parameter2',value2,...)`
- Description** The `add_param` command adds the specified parameters to the specified system and initializes the parameters to the specified values. Case is ignored for parameter names. Value strings are case sensitive. The value of the parameter must be a string. Once the parameter is added to a system, `set_param` and `get_param` can be used on the new parameters as if they were standard Simulink parameters.
- Examples** This command
- ```
add_param('vdp','Param1','Value1','Param2','Value2')
```
- adds the parameters `Param1` and `Param2` with values `'Value1'` and `'Value2'` to the `vdp` system.
- See Also** `delete_param`, `get_param`, `set_param`

# bdclose

---

**Purpose** Close any or all Simulink system windows unconditionally

**Syntax**  
`bdclose`  
`bdclose('sys')`  
`bdclose('all')`

**Description** `bdclose` with no arguments closes the current system window unconditionally and without confirmation. Any changes made to the system since it was last saved are lost.

`bdclose('sys')` closes the specified system window.

`bdclose('all')` closes all system windows.

**Examples** This command closes the vdp system.

```
bdclose('vdp')
```

**See Also** `close_system`, `new_system`, `open_system`, `save_system`



**Purpose** Return the name of the top-level Simulink system

**Syntax** bdroot  
bdroot('obj')

**Description** bdroot with no arguments returns the top-level system name.  
bdroot('obj', where 'obj' is a system or block pathname, returns the name of the top-level system containing the specified object name.

**Examples** This command returns the name of the top-level system that contains the current block.

```
bdroot(gcb)
```

**See Also** find\_system, gcb

# 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

# compare\_model

---

**Purpose** Compare two models

**Syntax**  
`compare_model('model1', 'model2', 0)`  
`compare_model('model1', 'model2', 1)`

**Description**  
`compare_model('model1', 'model2', 0)` compares model1 and model2 and returns a cell array that details the differences between the two models.  
`compare_model('model1', 'model2', 1)` returns only the nongraphical differences between the models.

The cell array returned by this function contains a cell array for each item compared between the two models. Each item cell array contains the following cells:

| Cell | Contents                                                                                     |
|------|----------------------------------------------------------------------------------------------|
| 1    | Name of compared item in first model                                                         |
| 2    | Name of compared item in second model                                                        |
| 3    | Type of the compared item (e.g., block, subsystem, state, etc.)                              |
| 4    | Comparison result: d if the two items differ, s if they're the same, u if the item is unique |
| 5    | Handle of the parent of the first item                                                       |
| 6    | Handle of the first item                                                                     |
| 7    | Handle of the parent of the second item                                                      |
| 8    | Handle of the second item                                                                    |

|                    |                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------|
| <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.<br><pre>delete_block('vdp/Out1')</pre>                     |
| <b>See Also</b>    | <code>add_block</code>                                                                                              |

# delete\_line

---

**Purpose** Delete a line from a Simulink system

**Syntax** `delete_line('sys', 'oport', 'iport')`

**Description** `delete_line('sys', 'oport', 'iport')` deletes the line extending from the specified block output port 'oport' to the specified block input port 'iport'. 'oport' and 'iport' are strings consisting of a block name and a port identifier in the form 'block/port'. Most block ports are identified by numbering the ports from top to bottom or from left to right, such as 'Gain/1' or 'Sum/2'. Enable, Trigger, and State ports are identified by name, such as 'subsystem\_name/Enable', 'subsystem\_name/Trigger', or 'Integrator/State'.

`delete_line('sys', [x y])` deletes one of the lines in the system that contains the specified point (x,y), if any such line exists.

**Example** This command removes the line from the mymodel system connecting the Sum block to the second input of the Mux block.

```
delete_line('mymodel', 'Sum/1', 'Mux/2')
```

**See Also** `add_line`

- Purpose** Delete a system parameter added via the `add_param` command
- Syntax** `delete_param('sys', 'parameter1', 'parameter2', ...)`
- Description** This command deletes parameters that were added to the system using the `add_param` command. The command displays an error message if a specified parameter was not added with the `add_param` command.
- Examples** The following example
- ```
delete_param('vdp', 'Param1')
add_param('vdp', 'Param1', 'Value1', 'Param2', 'Value2')
```
- adds the parameters `Param1` and `Param2` to the `vdp` system, then deletes `Param1` from the system.
- See Also** `add_param`

# find\_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 Appendix , “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`

<b>Purpose</b>	Get the pathname of the current block
<b>Syntax</b>	<pre>gcb gcb('sys')</pre>
<b>Description</b>	<p><code>gcb</code> returns the full block path name of the current block in the current system.</p> <p><code>gcb('sys')</code> returns the full block path name of the current block in the specified system.</p> <p>The current block is one of these:</p> <ul style="list-style-type: none"><li>• During editing, the current block is the block most recently clicked on.</li><li>• During simulation of a system that contains S-Function blocks, the current block is the S-Function block currently executing its corresponding MATLAB function.</li><li>• During callbacks, the current block is the block whose callback routine is being executed.</li><li>• During evaluation of the <code>MaskInitialization</code> string, the current block is the block whose mask is being evaluated.</li></ul>
<b>Examples</b>	<p>This command returns the path of the most recently selected block.</p> <pre>gcb ans =     clutch/Locked/Inertia</pre> <p>This command gets the value of the <b>Gain</b> parameter of the current block.</p> <pre>get_param(gcb, 'Gain') ans =     1/(Iv+Ie)</pre>
<b>See Also</b>	<code>gcbh</code> , <code>gcs</code>

# 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.

Appendix , "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'
    'Output'
```

This command returns the name of the currently selected block.

```
get_param(gcb, 'Name')
```

The following commands get the attributes of the currently selected block’s Name parameter.

```
p = get_param(gcb, 'ObjectParameters');
a = p.Name.Attributes

ans =
    'read-write'    'always-save'
```

The following command gets the dialog parameters of a Sine Wave block.

```
p = get_param('untitled/Sine Wave', 'DialogParameters')
p =
    'Amplitude'
    'Frequency'
    'Phase'
    'SampleTime'
```

## See Also

find\_system, set\_param

# new\_system

---

**Purpose** Create an empty Simulink system

**Syntax** `new_system('sys')`

**Description** `new_system('sys')` creates a new empty system with the specified name. If 'sys' specifies a path, the new system is a subsystem of the system specified in the path. `new_system` does not open the system window.

See Appendix , “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')
```

This command creates a new subsystem named 'mysys' in the vdp system.

```
new_system('vdp/mysys')
```

**See Also** `close_system`, `open_system`, `save_system`

**Purpose** Open a Simulink system window or a block dialog box

**Syntax**

```
open_system('sys')
open_system('blk')
open_system('blk', 'force')
```

**Description**

`open_system('sys')` opens the specified system or subsystem window, where 'sys' is the name of a model on the MATLAB path, the fully qualified pathname of a model, or the relative pathname of a subsystem of an already open system (for example, engine/Combustion). On UNIX, the fully qualified pathname of a model can start with a tilde (~), signifying your home directory.

`open_system('blk')`, where 'blk' is a full block pathname, opens the dialog box associated with the specified block. If the block's `OpenFcn` callback parameter is defined, the routine is evaluated.

`open_system('blk', 'force')`, where 'blk' is a full pathname or a masked system, looks under the mask of the specified system. This command is equivalent to using the **Look Under Mask** menu item.

**Example** This command opens the controller system in its default screen location.

```
open_system('controller')
```

This command opens the block dialog box for the Gain block in the controller system.

```
open_system('controller/Gain')
```

**See Also** `close_system`, `new_system`, `save_system`

# 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')
```

**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.

**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 Appendix, “Model and Block Parameters.”</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** Open the Simulink block library

**Syntax** `simulink`

**Description** On Microsoft Windows, the `simulink` command opens (or activates) the Simulink block library browser. On UNIX, the command opens the Simulink library window.

# sldiscmdl

---

**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:

<b>Value</b>	<b>Description</b>
'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:

<b>Value</b>	<b>Description</b>
'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:

<b>Value</b>	<b>Description</b>
'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
```

# slmdliscui

---

**Purpose** Open the Model Discretizer GUI

**Syntax** `slmdliscui('name')`

**Description** `slmdliscui('name')` Opens the Model Discretizer with the library or model specified by 'name'.

**Examples** This command opens the Model Discretizer with the f14 model.

```
slmdliscui('f14')
```

This command opens the Model Discretizer with the library named Test.

```
slmdliscui('Test')
```

# 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.

<b>Task</b>	<b>Command</b>
Create a new Simulink system.	<code>new_system</code>
Open an existing system.	<code>open_system</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>
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>
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>
Get a parameter value.	<code>get_param</code>
Set parameter values.	<code>set_param</code>
Delete a system parameter.	<code>delete_param</code>
Get the pathname of the current block.	<code>gcb</code>
Get the pathname of the current system.	<code>gcs</code>
Get the handle of the current block.	<code>gcbh</code>
Get the name of the root-level system.	<code>bdroot</code>
Open the Simulink block library.	<code>simulink</code>



---

## 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. Appendix , “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')
```

**Purpose** Execute a particular phase of the simulation of a model

**Syntax** `[sys,x0,str,ts] = model(t,x,u,flag);`

**Description** The `model` command executes a specific phase of the simulation of a Simulink model whose name is `model`. Use the command's `flag` argument to indicate 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 for interactive use, for example, to debug a model.

Use this command if you want to write an M-file program that needs to examine intermediate results of a simulation. Use the `sim` command if the program does not need to examine intermediate results. 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 inputs</li> <li>• <code>sys(4)</code> = number of outputs</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 that returns the initial condition of each of the system's states
<code>str</code>	State-ordering strings
<code>ts</code>	An $m$ -by-2 matrix containing the sample time (period, offset) information
<code>t</code>	Time step
<code>x</code>	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 states of the model's blocks at time t.</li><li>• 'outputs' computes the outputs of the model's blocks at time t.</li><li>• 'deriv' computes the state derivatives of the model's block at time step t.</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')
```

---

**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>	<table border="0"> <tr> <td style="padding-right: 20px;"><code>t</code></td> <td>Returns the simulation's time vector.</td> </tr> <tr> <td><code>x</code></td> <td>Returns the simulation's state matrix consisting of continuous states followed by discrete states.</td> </tr> <tr> <td><code>y</code></td> <td>Returns the simulation's output matrix. Each column contains the output of a root-level Outputport block, in port number order. If any Outputport block has a vector input, its output takes the appropriate number of columns.</td> </tr> <tr> <td><code>y1, ..., yn</code></td> <td>Each <math>y_i</math> returns the output of the corresponding root-level Outputport block for a model that has <math>n</math> such blocks.</td> </tr> <tr> <td><code>model</code></td> <td>Name of a block diagram.</td> </tr> </table>	<code>t</code>	Returns the simulation's time vector.	<code>x</code>	Returns the simulation's state matrix consisting of continuous states followed by discrete states.	<code>y</code>	Returns the simulation's output matrix. Each column contains the output of a root-level Outputport block, in port number order. If any Outputport block has a vector input, its output takes the appropriate number of columns.	<code>y1, ..., yn</code>	Each $y_i$ returns the output of the corresponding root-level Outputport block for a model that has $n$ such blocks.	<code>model</code>	Name of a block diagram.
<code>t</code>	Returns the simulation's time vector.										
<code>x</code>	Returns the simulation's state matrix consisting of continuous states followed by discrete states.										
<code>y</code>	Returns the simulation's output matrix. Each column contains the output of a root-level Outputport block, in port number order. If any Outputport block has a vector input, its output takes the appropriate number of columns.										
<code>y1, ..., yn</code>	Each $y_i$ returns the output of the corresponding root-level Outputport block for a model that has $n$ such blocks.										
<code>model</code>	Name of a block diagram.										

**timespan** Simulation start and stop time. Specify as one of these:  
tFinal to specify the stop time. The start time is 0.  
[tStart tFinal] to specify the start and stop times.  
[tStart OutputTimes tFinal] to specify the start and stop times and time points to be returned in t. Generally, t will include more time points. OutputTimes is equivalent to choosing **Produce additional output** on the dialog box. 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

$$T_s * [\text{vector of integers}]$$

where  $T_s$  is the fundamental time step to specify the additional output times. Do not use an expression of the form  $0:T_s:N*T_s$ .

**options** Optional simulation parameters specified as a structure created by the simset command (see simset on page 5-12).

**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

---

**Purpose** Plot simulation data in a figure window

**Syntax** `simplot(data);`  
`simplot(time, data);`

**Description** The `simplot` command plots output from a simulation in a Handle Graphics figure window. The plot looks like the display on the screen of a Scope block. Plotting the output on a figure window allows you to annotate and print the output.

**Arguments**

<code>data</code>	Data produced by one of Simulink's 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> .

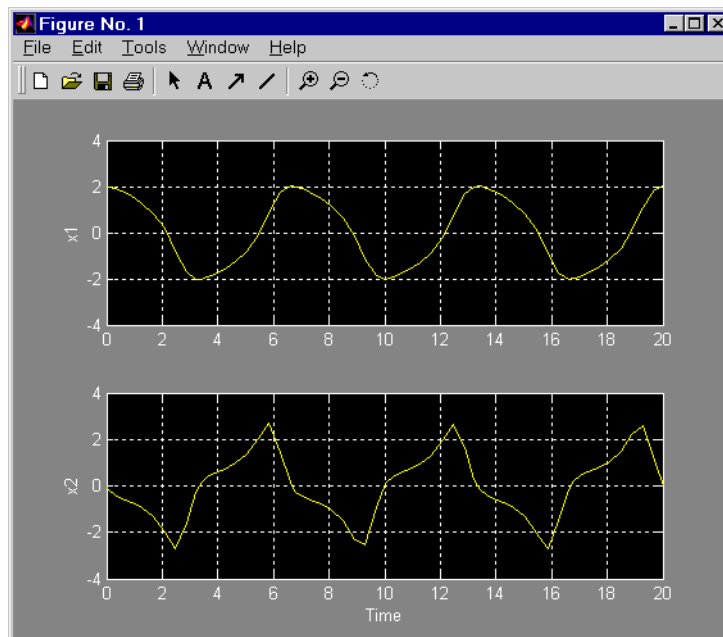


**Examples**

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

---

**Purpose** Create or edit simulation parameters and solver properties for the `sim` command

**Syntax**

```
options = simset(property, value, ...);
options = simset(old_opstruct, property, value, ...);
options = simset(old_opstruct, new_opstruct);
simset
```

**Description** The `simset` command creates a structure called `options`, 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.

`options = simset(property, value, ...)` sets the values of the named properties and stores the structure in `options`.

`options = simset(old_opstruct, property, value, ...)` modifies the named properties in `old_opstruct`, an existing structure.

`options = simset(old_opstruct, new_opstruct)` combines two existing options structures, `old_opstruct` and `new_opstruct`, into `options`. Any properties defined in `new_opstruct` overwrite the same properties defined in `old_opstruct`.

`simset` with no input arguments displays all property names and their possible values.

You cannot obtain or set values of these properties and parameters using the `get_param` and `set_param` commands.

**Parameters**

`AbsTol`                      positive scalar {1e-6}  
*Absolute error tolerance.* This scalar applies to all elements of the state vector. `AbsTol` applies only to the variable-step solvers.

`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

$$e(i) \leq \max(\text{RelTol} * \text{abs}(x(i)), \text{AbsTol}(i))$$

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`

# Mask Icon Drawing 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

This table summarizes the commands that you can use to create icons for masked subsystems.

<b>Command</b>	<b>Usage</b>
<code>disp</code>	Display text centered on a mask icon.
<code>dpoly</code>	Display a transfer function on a mask icon.
<code>fprintf</code>	Display variable text on a mask icon.
<code>image</code>	Display an image on a mask icon.
<code>patch</code>	Draws a color patch of a specified shape on a mask icon.
<code>plot</code>	Display graphics on a mask icon.
<code>port_label</code>	Display a port label on a mask icon.
<code>text</code>	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. Appendix , “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')
```

# disp

---

**Purpose** Display text on the icon of a masked subsystem

**Syntax** `disp(text)`  
`disp(text, 'texmode', 'on')`

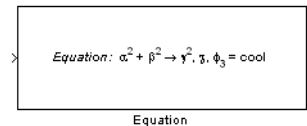
**Description** `disp(text)` displays *text* centered on the icon where *text* is any MATLAB expression that evaluates to a string.

`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 = {\bfcool}', 'texmode', 'on')
```

draws the equation that appears on this masked block icon.



**See Also** `fprintf`, `port_label`, `text`

# dpoly

---

**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-')`

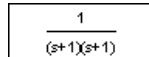
For example, for `num` and `den` as defined previously, the icon looks like this:

$$\frac{z^2}{1+2z^{-1}+z^2}$$

- 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`

# fprintf

---

**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`

**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`

# patch

---

**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,

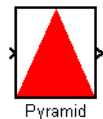
```
patch([0 .5 1], [0 1 0], [1 0 0])
```

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`



**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`

# port\_label

---

**Purpose** Draw a port label on the icon of a masked subsystem

**Syntax**  
`port_label(port_type, port_number, label)`  
`port_label(port_type, port_number, label, 'texmode', 'on')`

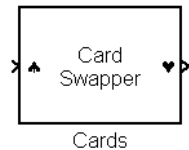
**Description** `port_label(port_type, port_number, label)` draws a label on a port where `port_type` is either 'input' or 'output', `port_number` is an integer, and `label` is a string specifying the port's label.

`port_label(port_type, port_number, label, 'texmode', 'on')` lets you use TeX formatting commands in `label`. The TeX formatting commands allow you to include symbols and Greek letters in the port label. See “Mathematical Symbols, Greek Letters, and Tex Characters” in the MATLAB documentation for information on the TeX formatting commands supported by Simulink.

**Examples** The command  
`port_label('input', 1, 'a')`  
defines `a` as the label of input port 1.

The commands  
`disp('Card\nSwapper');`  
`port_label('input',1,'\spadesuit','texmode','on');`  
`port_label('output',1,'\heartsuit','texmode','on');`

draw playing card symbols as the labels of the ports on a masked subsystem.



**See Also** `disp`, `fprintf`, `text`

**Purpose** Display text at a specific location on the icon of a masked subsystem

**Syntax**

```
text(x, y, text)
text(x, y, text, 'horizontalAlignment', halign,
      'verticalAlignment', valign)
text(x, y, text, 'texmode', 'on')
```

**Description** The text command places a character string (*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

# text

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

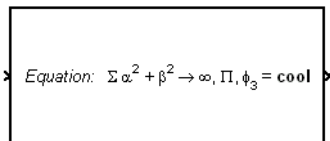
```
text(0.5, 0.5, 'foobar', 'horizontalAlignment', 'center')
```

centers foobar in the icon.

The command

```
text(.05,.5,'\itEquation:} \Sigma \alpha^2 +  
\beta^2 \rightarrow \infty, \Pi, \phi_3 = {\bfcool}',  
'hor','left','texmode','on')
```

draws a left-aligned equation on the icon.



Equation

## See Also

disp, fprintf, port\_label

# Simulink Debugger Commands

---

## Command Summary

The following table lists the debugger commands. The table's Repeat column specifies whether pressing the **Return** key at the command line repeats the command. Detailed descriptions of the commands follow the table.

Command	Short Form	Repeat	Description
ashow	as	No	Show an algebraic loop.
atrace	at	No	Set algebraic loop trace level.
bafter	ba	No	Insert a breakpoint after execution of a block.
break	b	No	Insert a breakpoint before execution of a block.
bshow	bs	No	Show a specified block.
clear	cl	No	Clear a breakpoint from a block.
continue	c	Yes	Continue the simulation.
disp	d	Yes	Display a block's I/O when the simulation stops.
help	? or h	No	Display help for debugger commands.
ishow	i	No	Enable or disable display of integration information.

<b>Command</b>	<b>Short Form</b>	<b>Repeat</b>	<b>Description</b>
minor	m	No	Enable or disable minor step mode.
nanbreak	na	No	Set or clear break on nonfinite value.
next	n	Yes	Go to start of the next time step.
probe	p	No	Display a block's I/O.
quit	q	No	Abort simulation.
run	r	No	Run the simulation to completion.
slist	sli	No	List a model's nonvirtual blocks.
states	state	No	Display current state values.
status	stat	No	Display debugging options in effect.
step	s	Yes	Step to next block.
stop	sto	No	Stop the simulation.
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 it 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.
xbreak	x	No	Break when the debugger encounters a step-size-limiting state.

---

<b>Command</b>	<b>Short Form</b>	<b>Repeat</b>	<b>Description</b>
zcbreak	zcb	No	Break at nonsampled zero-crossing events.
zclist	zcl	No	List blocks containing nonsampled zero crossings.

**Purpose** Show an algebraic loop.

**Syntax** `ashow <gcb | s:b | s#n | clear>`

**Arguments**

<code>s:b</code>	The block whose system index is <code>s</code> and block index is <code>b</code> .
<code>gcb</code>	Current block.
<code>s#n</code>	The algebraic loop numbered <code>n</code> in system <code>s</code> .
<code>clear</code>	Switch that clears loop coloring.

**Description** `ashow` without any arguments lists all of a model's algebraic loops in the MATLAB command window. `ashow gcb` or `ashow s:b` highlights the algebraic loop that contains the specified block. `ashow s#n` highlights the `n`th algebraic loop in system `s`. `ashow clear` removes algebraic loop highlights from the model diagram.

**See Also** `atrace`, `slist`

**Purpose** Show an algebraic loop.



**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.

<b>Command</b>	<b>Displays for Each Algebraic Loop</b>
<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 block is executed
<b>Syntax</b>	<code>bafter gcb</code> <code>bafter s:b</code>
<b>Arguments</b>	<code>s:b</code> The block whose system index is <code>s</code> and block index is <code>b</code> . <code>gcb</code> Current block.
<b>Description</b>	The <code>bafter</code> command inserts a breakpoint after execution of the specified block.
<b>See Also</b>	<code>break</code> , <code>xbreak</code> , <code>tbreak</code> , <code>nanbreak</code> , <code>zcbreak</code> , <code>slist</code>

<b>Purpose</b>	Insert a breakpoint before a block is executed				
<b>Syntax</b>	<pre>break gcb break s:b</pre>				
<b>Arguments</b>	<table><tr><td>s:b</td><td>The block whose system index is s and block index is b.</td></tr><tr><td>gcb</td><td>Current block.</td></tr></table>	s:b	The block whose system index is s and block index is b.	gcb	Current block.
s:b	The block whose system index is s and block index is b.				
gcb	Current block.				
<b>Description</b>	The break command inserts a breakpoint before execution of the specified block.				
<b>See Also</b>	bafter, tbreak, xbreak, nanbreak, zcbreak, slist				

# 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

<b>Purpose</b>	Clear a breakpoint from a block
<b>Syntax</b>	<code>clear gcb</code> <code>clear 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>clear</code> command clears a breakpoint from the specified block.
<b>See Also</b>	<code>bafter</code> , <code>slist</code>

# continue

---

**Purpose** Continue the simulation

**Syntax** `continue`

**Description** The `continue` command continues the simulation from the current breakpoint. The simulation continues until it reaches another breakpoint or its final time step.

**See Also** `run`, `stop`, `quit`

<b>Purpose</b>	Display a block's I/O when the simulation stops
<b>Syntax</b>	<code>disp gcb</code> <code>disp s:b</code> <code>disp</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>

# help

---

**Purpose** Display help for debugger commands

**Syntax** help

**Description** The help command displays a list of debugger commands in the command window. The list includes the syntax and a brief description of each command.



<b>Purpose</b>	Toggle model execution between accelerated and normal mode
<b>Syntax</b>	emode
<b>Description</b>	See “Using the Simulink Accelerator with the Simulink Debugger” for more information.

# ishow

---

<b>Purpose</b>	Enable or disable display of integration information
<b>Syntax</b>	ishow
<b>Description</b>	The ishow command toggles display of integration information during a simulation.
<b>See Also</b>	atrace

**Purpose** Enable or disable minor step mode

**Syntax** `minor`

**Description** The `minor` command causes the debugger to enter or exit minor step mode. In minor step mode, the `step` command advances the simulation by blocks within a minor step. In minor step mode, after executing the last block in the model's sorted block list, the `step` command advances the simulation to the next minor time step, if any minor time steps remain in the current major time step; otherwise, the `step` command advances the simulation to the first minor time step in the next major time step.

**See Also** `step`

# nanbreak

---

**Purpose** Set or clear nonfinite value break mode

**Syntax** nanbreak

**Description** The nanbreak command causes the debugger to break whenever the simulation encounters a nonfinite (NaN or Inf) value. If nonfinite break mode is set, nanbreak clears it.

**See Also** break, bafter, xbreak, tbreak, zcbreak

<b>Purpose</b>	Go to start of the next time step
<b>Syntax</b>	next
<b>Description</b>	The next command evaluates all the blocks remaining to be evaluated in the current time step, stopping at the start of the next time step. After executing the next command, the debugger highlights the first block to be evaluated on the next time step and displays the time of the next step.
<b>See Also</b>	step

# probe

---

**Purpose** Display a block's state

**Syntax** `probe [<s:b | gcb>] [level io | (all)]`

**Arguments**

<code>s:b</code>	The block whose system index is <code>s</code> and block index is <code>b</code> .
<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.

**Description** `probe` causes the debugger to enter or exit probe mode. In probe mode, the debugger displays the I/O of any block you select. To exit probe mode, enter any command. `probe gcb` displays the I/O of the currently selected block. `probe s:b` displays the I/O of the block whose index is `s:b`.

**See Also** `disp`, `trace`

<b>Purpose</b>	Abort simulation
<b>Syntax</b>	quit
<b>Description</b>	The quit command terminates the current simulation.
<b>See Also</b>	stop

# run

---

**Purpose** Run the simulation to completion

**Syntax** run

**Description** The run command runs the simulation from the current breakpoint to its final time step. It ignores breakpoints and display points.

**See Also** continue, stop, quit



- Purpose** List a model's nonvirtual blocks
- Syntax** `slist`
- Description** The `slist` command lists the nonvirtual blocks in the model being debugged. The list shows the block index and name of each listed block.
- See Also** `systems`

# states

---

**Purpose**            Display current state values

**Syntax**            states

**Description**        The states command displays a list of the current states of the model. The display lists the value, index, and name of each state.

**See Also**            ishow

<b>Purpose</b>	Display debugging options in effect
<b>Syntax</b>	status
<b>Description</b>	The status command displays a list of the debugging options in effect.

# step

---

**Purpose** Step to next block

**Syntax** step

**Description** The step command evaluates the next block to be evaluated in the current time step. After executing the step command, the debugger highlights the next block to be evaluated and its output signal lines. It also displays the name of the next block as part of the debugger command-line prompt.

**See Also** next

<b>Purpose</b>	Stop the simulation
<b>Syntax</b>	<code>stop</code>
<b>Description</b>	The <code>stop</code> command stops the simulation.
<b>See Also</b>	<code>continue</code> , <code>run</code> , <code>quit</code>

# systems

---

<b>Purpose</b>	List a model's nonvirtual systems
<b>Syntax</b>	systems
<b>Description</b>	The systems command lists a model's nonvirtual systems in the MATLAB command window.
<b>See Also</b>	slist

**Purpose** Set or clear a time breakpoint

**Syntax** tbreak t  
tbreak

**Description** The tbreak command sets a breakpoint at the specified time step. If a breakpoint already exists at the specified time, tbreak clears the breakpoint. If you do not specify a time, tbreak toggles a breakpoint at the current time step.

**See Also** break, bafter, xbreak, nanbreak, zcbreak

# 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>



<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 <code>undisp</code> 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>

<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>zcbreak</code> , <code>tbreak</code> , <code>nanbreak</code>

# zcbreak

---

<b>Purpose</b>	Toggle breaking at nonsampled zero-crossing events
<b>Syntax</b>	zcbreak
<b>Description</b>	The zcbreak command causes the debugger to break when a nonsampled zero-crossing event occurs. If zero-crossing break mode is already on, zcbreak turns the mode off.
<b>See Also</b>	break, bafter, xbreak, tbreak, nanbreak, zclist

<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>



# Model and Block Parameters

---

The following sections lists parameters that you can set, using the `set_param` command.

“Model Parameters” on page 8-2	Parameters specific to models
“Common Block Parameters” on page 8-7	Parameters that all blocks have
“Block-Specific Parameters” on page 8-10	Parameters that a specific block has
“Mask Parameters” on page 8-26	Parameters of 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 9, “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.

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.

**Table 8-1: Model Parameters**

Parameter	Description	Values
AbsTol	Absolute error tolerance	scalar {1e-6}
AlgebraicLoopMsg	Algebraic loop diagnostic	none   {warning}   error
ArrayBoundsChecking	Enable array bounds checking	'none'   'warning'   'error'
BooleanDataType	Enable Boolean mode	on   {off}
BufferReuse	Enable reuse of block I/O buffers	{on}   off
CloseFcn	Close callback	command or variable
ConfigurationManager	Configuration manager for this model	text
ConsistencyChecking	Consistency checking	on   {off}
Created	Date and time model was created	text
Creator	Name of model creator	text
Decimation	Decimation factor	scalar {1}
DefaultBlockFontSize	Default font size for blocks contained by this model	{10}



**Table 8-1: Model Parameters**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
Description	Description of this model	text
ExternalInput	Time and input variable names	scalar or vector [t, u]
FinalStateName	Final state name	variable {xFinal}
FixedStep	Fixed step size	scalar {auto}
InitialState	Initial state name or values	variable or vector {xInitial}
InitialStep	Initial step size	scalar {auto}
InvariantConstants	Invariant constant setting	on   {off}
LimitDataPoints	Limit output	on   {off}
LoadExternalInput	Load input from workspace	on   {off}
LoadInitialState	Load initial state	on   {off}
MaxDataPoints	Maximum number of output data points to save	scalar {1000}
MaxOrder	Maximum order for ode15s	1   2   3   4   {5}
MaxStep	Maximum step size	scalar {auto}
MinStepSizeMsg	Minimum step size diagnostic	{warning}   error
ModelVersionFormat	Format of model's version number	text
ModifiedBy	Last modifier of this model	text
ModifiedDateFormat	Format of modified date	text
Name	Model name	text
ObjectParameters	Names/attributes of model parameters	structure
OutputOption	Output option	AdditionalOutputTimes   {RefineOutputTimes}   SpecifiedOutputTimes
OutputSaveName	Simulation output name	variable {yout}
OutputTimes	Values for chosen OutputOption	vector {[]}

**Table 8-1: Model Parameters**

Parameter	Description	Values
PaperOrientation	Printing paper orientation	portrait   {landscape}
PaperPosition	Position of diagram on paper	[left, bottom, width, height]
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
PostLoadFcn	Post-load callback	command or variable
PreLoadFcn	Preload callback	command or variable
Refine	Refine factor	scalar {1}
RelTol	Relative error tolerance	scalar {1e-3}
SampleTimeColors	<b>Sample Time Colors</b> menu option	on   {off}
SaveFcn	Save callback	command or variable
SaveFinalState	Save final state	on   {off}
SaveFormat	Format used to save data to the MATLAB workspace	Array   Structure   StructureWithTime
SaveOutput	Save simulation output	{on}   off
SaveState	Save states	on   {off}
SaveTime	Save simulation time	{on}   off

**Table 8-1: Model Parameters**

<b>Parameter</b>	<b>Description</b>	<b>Values</b>
ScreenColor	Background color of the model window	black   {white}   red   green   blue   cyan   magenta   yellow   gray   lightBlue   orange   darkGreen
ShowLineWidths	<b>Show Line Widths</b> menu option	on   {off}
SimulationCommand	Executes a simulation command.	start   stop   pause   continue   update
SimParamPage	<b>Simulation Parameters</b> dialog box page to display (page last displayed)	{Solver}   WorkspaceI/O   Diagnostics
Solver	Solver	{ode45}   ode23   ode113   ode15s   ode23s   ode5   ode4   ode3   ode2   ode1   FixedStepDiscrete   VariableStepDiscrete
StartFcn	Start simulation callback	command or variable
StartTime	Simulation start time	scalar {0.0}
StateSaveName	State output name	variable {xout}
StopFcn	Stop simulation callback	command or variable
StopTime	Simulation stop time	scalar {10.0}
TimeSaveName	Simulation time name	variable {tout}
UnconnectedInputMsg	Unconnected input ports diagnostic	none   {warning}   error
UnconnectedLineMsg	Unconnected lines diagnostic	none   {warning}   error
UnconnectedOutputMsg	Unconnected output ports diagnostic	none   {warning}   error
Version	Simulink version used to modify the model (read-only)	(release)
WideVectorLines	<b>Wide Vector Lines</b> menu option	on   {off}
ZeroCross	Intrinsic zero-crossing detection (see “Zero-Crossing Detection”)	{on}   off

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.

**Table 8-2: Common Block Parameters**

Parameter	Description	Values
AttributesFormatString	Specifies parameters to be displayed below block in a block diagram	string
BackgroundColor	Block icon background	black   {white}   red   green   blue   cyan   magenta   yellow   gray   lightBlue   orange   darkGreen
BlockDescription	Block description	text
BlockType	Block type	text
CloseFcn	Close callback	MATLAB expression
CompiledPortWidths	Structure of port widths	scalar and vector
CopyFcn	Copy callback	MATLAB expression
DeleteFcn	Delete callback	MATLAB expression
Description	User-specifiable description	text
DialogParameters	Names/attributes of parameters in blocks parameter dialog	structure
DropShadow	Display drop shadow	{off}   on
FontAngle	Font angle	(system-dependent) {normal}   italic   oblique
FontName	Font	{Helvetica}

**Table 8-2: 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.	{-1}
FontWeight	Font weight	(system-dependent) light   {normal}   demi   bold
ForegroundColor	Block name, icon, outline, output signals, and signal label	{black}   white   red   green   blue   cyan   magenta   yellow   gray   lightBlue   orange   darkGreen
InitFcn	Initialization callback	MATLAB expression
InputPorts	Array of input port locations	[h1,v1; h2,v2; ...]
LinkStatus	Link status of block	none   resolved   unresolved   implicit
LoadFcn	Load callback	MATLAB expression
ModelCloseFcn	Model close callback	MATLAB expression
Name	Block's name	string
NameChangeFcn	Block name change callback	MATLAB expression
NamePlacement	Position of block name	{normal}   alternate
ObjectParameters	Names/attributes of block's parameters	structure
OpenFcn	Open callback	MATLAB expression
Orientation	Where block faces	{right}   left   down   up
OutputPorts	Array of output port locations	[h1,v1; h2,v2; ...]
Parent	Name of the system that owns the block	string
ParentCloseFcn	Parent subsystem close callback	MATLAB expression

**Table 8-2: Common Block Parameters (Continued)**

Parameter	Description	Values
Position	Position of block in model window	vector [left top right bottom] <i>not</i> enclosed in quotation marks
PostSaveFcn	Postsave callback	MATLAB expression
PreSaveFcn	Presave callback	MATLAB expression
Selected	Block selected state	on   {off}
ShowName	Display block name	{on}   off
StartFcn	Start simulation callback	MATLAB expression
StopFcn	Termination of simulation callback	MATLAB expression
Tag	User-defined string	' '
Type	Simulink object type (read-only)	'block'
UndoDeleteFcn	Undo block delete callback	MATLAB expression
UserData	Any MATLAB data type	[ ]
UserDataPersistent	Save UserData in the model file.	on   {off}

These examples illustrate how to change these 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 8-26.

---

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).

**Table 8-3: Continuous Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Derivative (Derivative) (no block-specific parameters)		
Integrator (Integrator)		
ExternalReset	External reset	{none}   rising   falling   either
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}
ShowSaturationPort	Show saturation port	{off}   on
ShowStatePort	Show state port	{off}   on



**Table 8-3: Continuous Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
AbsoluteTolerance	Absolute tolerance	scalar {auto}
State-Space (StateSpace)		
A	A	matrix {1}
B	B	matrix {1}
C	C	matrix {1}
D	D	matrix {1}
X0	Initial conditions	vector {0}
Transfer Fcn (TransferFcn)		
Numerator	Numerator	vector or matrix {[1]}
Denominator	Denominator	vector {[1 1]}
Transport Delay (TransportDelay)		
DelayTime	Time delay	scalar or vector {1}
InitialInput	Initial input	scalar or vector {0}
BufferSize	Initial buffer size	scalar {1024}
Variable Transport Delay (VariableTransportDelay)		
MaximumDelay	Maximum delay	scalar or vector {10}
InitialInput	Initial input	scalar or vector {0}
MaximumPoints	Buffer size	scalar {1024}
Zero-Pole (ZeroPole)		
Zeros	Zeros	vector {[1]}
Poles	Poles	vector {[0 -1]}
Gain	Gain	vector {[1]}

**Table 8-4: Discontinuities Library Block Parameters**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Backlash (Backlash)		
BacklashWidth	Deadband width	scalar or vector {1}
InitialOutput	Initial output	scalar or vector {0}
Coulomb & Viscous Friction (Coulombic and Viscous Friction) (masked)		
Dead Zone (DeadZone)		
LowerValue	Start of dead zone	scalar or vector {-0.5}
UpperValue	End of dead zone	scalar or vector {0.5}
Hit Crossing (HitCross)		
HitCrossingOffset	Hit crossing offset	scalar or vector {0}
HitCrossingDirection	Hit crossing direction	rising   falling   {either}
ShowOutputPort	Show output port	{on}   off
Quantizer (Quantizer)		
QuantizationInterval	Quantization interval	scalar or vector {0.5}
Rate Limiter (RateLimiter)		
RisingSlewLimit	Rising slew rate	scalar or vector {1.}
FallingSlewLimit	Falling slew rate	scalar or vector {-1.}
Relay (Relay)		
OnSwitchValue	Switch on point	scalar or vector {eps}
OffSwitchValue	Switch off point	scalar or vector {eps}
OnOutputValue	Output when on	scalar or vector {1}
OffOutputValue	Output when off	scalar or vector {0}
Saturation (Saturate)		

**Table 8-4: Discontinuities Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
UpperLimit	Upper limit	scalar or vector {0.5}
LowerLimit	Lower limit	scalar or vector {-0.5}

**Table 8-5: Discrete Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Discrete Filter (DiscreteFilter)		
Numerator	Numerator	vector {[1]}
Denominator	Denominator	vector {[1 2]}
SampleTime	Sample time	scalar (sample period) {1} or vector [period offset]
Discrete State-Space (DiscreteStateSpace)		
A	A	matrix {1}
B	B	matrix {1}
C	C	matrix {1}
D	D	matrix {1}
X0	Initial conditions	vector {0}
SampleTime	Sample time	scalar (sample period) {1} or vector [period offset]
Discrete-Time Integrator (DiscreteIntegrator)		
IntegratorMethod	Integrator method	{ForwardEuler}   BackwardEuler   Trapezoidal
ExternalReset	External reset	{none}   rising   falling   either
InitialConditionSource	Initial condition source	{internal}   external
InitialCondition	Initial condition	scalar or vector {0}

**Table 8-5: Discrete Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
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
SampleTime	Sample time	scalar (sample period) {1} or vector [period offset]
Discrete Transfer Fcn (DiscreteTransferFcn)		
Numerator	Numerator	vector {[1]}
Denominator	Denominator	vector {[1 0.5]}
SampleTime	Sample time	scalar (sample period) {1} or vector [period offset]
Discrete Zero-Pole (DiscreteZeroPole)		
Zeros	Zeros	vector {[1]}
Poles	Poles	vector [0 0.5]
Gain	Gain	scalar {1}
SampleTime	Sample time	scalar (sample period) {1} or vector [period offset]
First-Order Hold (First Order Hold) (masked)		
Memory (Memory)		
X0	Initial condition	scalar or vector {0}
InheritSampleTime	Inherit sample time	{off}   on
Unit Delay (UnitDelay)		
X0	Initial condition	scalar or vector {0}

**Table 8-5: Discrete Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time	scalar (sample period) {1} or vector [period offset]
Zero-Order Hold (ZeroOrderHold)		
SampleTime	Sample time	scalar (sample period) {1} or vector [period offset]

**Table 8-6: Look-Up Tables Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Look-Up Table (Lookup)		
InputValues	Vector of input values	vector {[-5:5]}
OutputValues	Vector of output values	vector {tanh([-5:5])}
Look-Up Table (2-D) (Lookup Table (2-D)) (masked)		
RowIndex	Row	vector
ColumnIndex	Column	vector
OutputValues	Table	2-D matrix

**Table 8-7: Math Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Abs (Abs) (no block-specific parameters)		
Algebraic Constraint (Algebraic Constraint) (masked)		
Combinatorial Logic (CombinatorialLogic)		
TruthTable	Truth table	matrix {[0 0;0 1;0 1;1 0; 0 1;1 0;1 0;1 1]}

**Table 8-7: Math Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Complex to Magnitude-Angle		
Complex to Real-Imag		
Dot Product (Dot Product) (masked)		
Gain (Gain)		
Gain	Gain	scalar or vector {1}
Logical Operator (Logic)		
Operator	Operator	{AND}   OR   NAND   NOR   XOR   NOT
Inputs	Number of input ports	scalar {2}
Magnitude-Angle to Complex		
Math Function (Math)		
Operator	Function	{exp}   log   log10   square   sqrt   pow   reciprocal   hypot   rem   mod
Matrix Gain (Matrix Gain) (masked)		
MinMax (MinMax)		
Function	Function	{min}   max
Inputs	Number of input ports	scalar {1}
Product (Product)		
Inputs	Number of inputs	scalar {2}
Relational Operator (RelationalOperator)		
Operator	Operator	==   !=   <   {<=}   >=   >
Relational Operator (RelationalOperator)		
Operator	Operator	==   !=   <   {<=}   >=   >
Rounding Function (Rounding)		
Operator	Function	{floor}   ceil   round   fix

**Table 8-7: Math Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
Sign (Signum) (no block-specific parameters)		
Slider Gain (SliderGain) (masked)		
Sum (Sum)		
Inputs	List of signs	scalar or list of signs {++}
Trigonometric Function (Trigonometry)		
Operator	Function	{sin}   cos   tan   asin   acos   atan   atan2   sinh   cosh   tanh

**Table 8-8: Model-Wide Utilities**

Block (Type)/Parameter	Dialog Box Prompt	Values
Model Info (CMBlock) (mask)		

**Table 8-9: Ports & Subsystems**

Block (Type)/Parameter	Dialog Box Prompt	Values
Configurable Subsystem (mask)		
Choice	Block choice	string
LibraryName	Library name	string
Enable (EnablePort)		
StatesWhenEnabling	States when enabling	{held}   reset
ShowOutputPort	Show output port	{off}   on
In (Inport)		
Port	Port number	scalar {1}
PortWidth	Port width	scalar {-1}

**Table 8-9: Ports & Subsystems (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
SampleTime	Sample time	scalar (sample period) {-1} or vector [period offset]
Out (Outport)		
Port	Port number	scalar {1}
OutputWhenDisabled	Output when disabled	{held}   reset
InitialOutput	Initial output	scalar or vector {0}
Subsystem (SubSystem)		
ShowPortLabels	Show/Hide Port Labels <b>Format</b> menu item	{on}   off
Terminator (Terminator) (no block-specific parameters)		
Trigger (TriggerPort)		
TriggerType	Trigger type	{rising}   falling   either   function-call
ShowOutputPort	Show output port	{off}   on

**Table 8-10: Signal Attributes**

Block (Type)/Parameter	Dialog Box Prompt	Values
Data Type Conversion		
IC (InitialCondition)		
Value	Initial value	scalar or vector {1}
Width (Width) (no block-specific parameters)		



**Table 8-11: Signal Routing**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Bus Selector (BusSelector)		
InputSignals		Cell array of the input signals nested to reflect the signal hierarchy
Data Store Memory (DataStoreMemory)		
DataStoreName	Data store name	tag {A}
InitialValue	Initial value	vector {0}
Data Store Read (DataStoreRead)		
DataStoreName	Data store name	tag {A}
SampleTime	Sample time	scalar (sample period) {-1} or vector [period offset]
Data Store Write (DataStoreWrite)		
DataStoreName	Data store name	tag {A}
SampleTime	Sample time	scalar (sample period) {-1} or vector [period offset]
Demux (Demux)		
Outputs	Number of outputs	scalar or vector {3}
From (From)		
GotoTag	Goto tag	tag {A}
Goto (Goto)		
GotoTag	Tag	tag {A}
TagVisibility	Tag visibility	{local}   scoped   global
Goto Tag Visibility (GotoTagVisibility)		
GotoTag	Goto tag	tag {A}

**Table 8-11: Signal Routing (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Manual Switch (Manual Switch) (masked)		
Merge		
Multiport Switch (MultiPortSwitch)		
Inputs	Number of inputs	scalar or vector {3}
Mux (Mux)		
Inputs	Number of inputs	scalar or vector {3}
Switch (Switch)		
Threshold	Threshold	scalar or vector {0}

**Table 8-12: Sinks Library Block Parameters**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Display (Display)		
Format	Format	{short}   long   short_e   long_e   bank
Decimation	Decimation	scalar {1}
Floating	Floating display	{off} on
SampleTime	Sample time	scalar (sample period) {-1} or vector [period offset]
Out (Outputport)		
Port	Port number	scalar {1}
OutputWhenDisabled	Output when disabled	{held}   reset
InitialOutput	Initial output	scalar or vector {0}
Scope (Scope)		

**Table 8-12: Sinks Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
Location	Position of Scope window on screen	vector {[left top right bottom]}
Open	(If Scope is open when the model is opened; cannot be set from dialog box)	{off}   on
NumInputPorts	Number of axes	positive integer > 0
TickLabels	Hide tick labels	{on}   off
ZoomMode	(Zoom button initially pressed)	{on}   xonly   yonly
AxesTitles	Title (on right-click axes)	scalar {auto}
Grid	(For future use)	{on}   off
TimeRange	Time range	scalar {auto}
YMin	Y min	scalar {-5}
YMax	Y max	scalar {5}
SaveToWorkspace	Save data to workspace	{off}   on
SaveName	Variable name	variable {ScopeData}
DataFormat	Format	{matrix   structure}
LimitMaxRows	Limit rows to last	{on}   off
MaxRows	(no label)	scalar {5000}
Decimation	(Value if Decimation is selected)	scalar {1}
SampleInput	(Toggles with Decimation)	{off}   on
SampleTime	(SampleInput value)	scalar (sample period) {0} or vector [period offset]
Stop Simulation (StopSimulation) (no block-specific parameters)		
Terminator (Terminator) (no block-specific parameters)		

**Table 8-12: Sinks Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
To File (ToFile)		
Filename	Filename	filename {untitled.mat}
MatrixName	Variable name	variable {ans}
Decimation	Decimation	scalar {1}
SampleTime	Sample time	scalar (sample period) {-1} or vector [period offset]
To Workspace (ToWorkspace)		
VariableName	Variable name	variable {simout}
Buffer	Maximum number of rows	scalar {inf}
Decimation	Decimation	scalar {1}
SampleTime	Sample time	scalar (sample period) {-1} or vector [period offset]
XY Graph (XY scope.) (masked)		

**Table 8-13: Sources Library Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Band-Limited White Noise (Continuous White Noise) (masked)		
Chirp Signal (chirp) (masked)		
VectorParams1D	Interpret vector parameters as 1-D	off {on}
Clock (Clock) (no block-specific parameters)		
Constant (Constant)		
Value	Constant value	scalar or vector {1}

**Table 8-13: Sources Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
VectorParams1D	Interpret vector parameters as 1-D	off {on}
Digital Clock (DigitalClock)		
SampleTime	Sample time	scalar (sample period) {1} or vector [period offset]
From File (FromFile)		
FileName	Filename	filename {untitled.mat}
From Workspace (FromWorkspace)		
VariableName	Matrix table	matrix {[T,U]}
Ground (Ground) (no block-specific parameters)		
In (Inport)		
Port	Port number	scalar {1}
PortWidth	Port width	scalar {-1}
SampleTime	Sample time	scalar (sample period) {-1} or vector [period offset]
Pulse Generator (Pulse Generator)		
Amplitude	Amplitude	scalar {1}, vector, or matrix
PhaseDelay	Phase delay	scalar {0}, vector, or matrix
PulseType	Pulse type	{'Time based'}   'Sample based'
PulseWidth	Pulse width	scalar {50}, vector, or matrix
SampleTime	Sample time	scalar (sample period) {1} or vector [period offset]
VectorParams1D	Interpret vector parameters as 1-D	off {on}
Ramp (Ramp) (masked)		

**Table 8-13: Sources Library Block Parameters (Continued)**

<b>Block (Type)/Parameter</b>	<b>Dialog Box Prompt</b>	<b>Values</b>
VectorParams1D	Interpret vector parameters as 1-D	off {on}
Random Number (RandomNumber)		
Seed	Initial seed	scalar or vector {0}
VectorParams1D	Interpret vector parameters as 1-D	off {on}
Repeating Sequence (Repeating table) (masked)		
Signal Generator (SignalGenerator)		
WaveForm	Wave form	{sine}   square   sawtooth   random
Amplitude	Amplitude	scalar or vector {1}
Frequency	Frequency	scalar or vector {1}
Units	Units	{Hertz}   rad/sec
VectorParams1D	Interpret vector parameters as 1-D	off {on}
Sine Wave (Sin)		
Amplitude	Amplitude	scalar or vector {1}
Frequency	Frequency	scalar or vector {1}
Phase	Phase	scalar or vector {0}
SampleTime	Sample time	scalar (sample period) {-1} or vector [period offset]
VectorParams1D	Interpret vector parameters as 1-D	off {on}
Step (Step)		
Time	Step time	scalar or vector {1}
Before	Initial value	scalar or vector {0}

**Table 8-13: Sources Library Block Parameters (Continued)**

Block (Type)/Parameter	Dialog Box Prompt	Values
After	Final value	scalar or vector {1}
VectorParams1D	Interpret vector parameters as 1-D	off {on}
Uniform Random Number (Uniform RandomNumber)		
Minimum	Minimum	scalar or vector {-1}
Maximum	Maximum	scalar or vector {1}
Seed	Initial Seed	scalar or vector {0}
SampleTime	Sample Time	scalar or vector {0}
VectorParams1D	Interpret vector parameters as 1-D	off {on}

**Table 8-14: User-Defined Functions Block Parameters**

Block (Type)/Parameter	Dialog Box Prompt	Values
Fcn (Fcn)		
Expr	Expression	expression {sin(u(1)*exp(2.3*(-u(2))))}
MATLAB Fcn (MATLABFcn)		
MATLABFcn	MATLAB function	MATLAB function {sin}
OutputWidth	Output width	scalar or vector {-1}
S-Function (S-Function)		
FunctionName	S-function name	name {system}
Parameters	S-function parameters	Additional parameters if needed

## Mask Parameters

This section lists parameters that describe masked blocks. This table lists masking parameters, which correspond to **Mask Editor** dialog box parameters.

**Table 8-15: Mask Parameters**

Parameter	Description/Prompt	Values
Mask	Turns mask on or off.	{on}   off
MaskCallbackString	Mask parameter callbacks	delimited string
MaskCallbacks	Mask parameter callbacks	cell array
MaskDescription	Block description	string
MaskDisplay	Drawing commands	display commands
MaskEditorHandle	Mask editor figure handle (for internal use)	handle
MaskEnableString	Mask parameter enable status	delimited string
MaskEnables	Mask parameter enable status	cell array of strings, each either 'on' or 'off'
MaskHelp	Block help	string
MaskIconFrame	Icon frame (Visible is on, Invisible is off)	{on}   off
MaskIconOpaque	Icon transparency (Opaque is on, Transparent is off)	{on}   off
MaskIconRotate	Icon rotation (Rotates is ON, Fixed is off)	on   {off}
MaskIconUnits	Drawing coordinates	Pixel   {Autoscale}   Normalized
MaskInitialization	Initialization commands	MATLAB command
MaskNames		
MaskPrompts	Prompt (see below)	cell array of strings
MaskPromptString	Prompt (see below)	delimited string



**Table 8-15: Mask Parameters (Continued)**

Parameter	Description/Prompt	Values
MaskPropertyNameString		
MaskSelfModifiable	Indicates that the block can modify itself.	on   {off}
MaskStyles	Control type (see below)	cell array {Edit}   Checkbox   Popup
MaskStyleString	Control type (see below)	{Edit}   Checkbox   Popup
MaskTunableValues	Tunable parameter attributes	cell array of strings
MaskTunableValueString	Tunable parameter attributes	delimited string
MaskType	Mask type	string
MaskValues	Block parameter values (see below)	cell array of strings
MaskValueString	Block parameter values (see below)	delimited string
MaskVariables	Variable (see below)	string
MaskVisibilities	Specifies visibility of 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**

The mask parameters, listed in the table on the previous page, 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 8, “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.



## A

- Abs block 2-3
- absolute tolerance
  - simset parameter 5-12
  - specifying for a block state 2-186
- absolute value
  - generating 2-3, 2-123
- Action Port block 2-5
- Action subsystems
  - creating 2-5
  - with If block 2-168
  - with SwitchCase block 2-341
- add\_block command 4-6
- add\_line command 4-7
- add\_param command 4-9
- Algebraic Constraint block 2-8
- algebraic equations
  - modeling 2-8
- algebraic loops
  - integrator block reset or IC port 2-112
- analysis functions
  - perturbing model 2-175
- AND operator 2-23
- AnnotationDefaults section of mdl file 9-3
- annotations
  - annotation block
    - See Model Info block*
- ashow debug command 7-1
- Assert block 2-10
- Assignment block 2-12
- Atomic Subsystem block 2-330
- atrace debug command 7-2
- automatic scaling
  - and Look-Up Table (2D) block 2-203
- autoscaling Scope axes 2-291

## B

- Backlash block 2-17
- Backward Euler method 2-110
- Backward Rectangular method 2-110
- bafter debug command 7-3
- Band-Limited White Noise block 2-21
- bdclose command 4-10
- bdroot command 4-11
- Bitwise Logical Operator block 2-23
- block dialog boxes
  - closing 4-12
  - opening 4-29
- block libraries
  - Blocksets and Toolboxes 1-19
  - Demos 1-20
  - Extras 1-19
- block names
  - newline character in 4-4, 5-3, 6-3
  - slash character in 4-5, 5-4, 6-4
- block parameters
  - changing during simulation 4-33
  - common 8-7
  - Continuous library 8-10
  - Discontinuities library 8-12
  - Discrete library 8-13
  - Look-Up Tables library 8-15, 8-25
  - Math library 8-15
  - Model-Wide Utilities library 8-17
  - Ports & Subsystems library 8-17
  - Signal Attributes library 8-18
  - Signal Routing library 8-19
  - Sinks library 8-20
  - Sources library 8-22
- BlockDefaults section of mdl file 9-3

## blocks

adding to model 4-6

current 4-23

deleting

`delete_block` command 4-15

handle of current 4-24

specifying path 4-4, 5-3, 6-3

## blocks

*See also* block parameters 8-10

Blocksets and Toolboxes library 1-19

bode function 3-4

Boolean expressions

modeling 2-64

break debug command 7-4

bshow debug command 7-5

Bus Creator block 2-27

Bus Selector block 2-31

**C**

capping unconnected blocks 2-346, 2-363

Check Discrete Gradient block 2-33

Check Dynamic Gap block 2-36

Check Dynamic Lower Bound block 2-39

Check Dynamic Range block 2-42

Check Dynamic Upper Bound block 2-44

Check Input Resolution block 2-46

Check Static Gap block 2-48

Check Static Lower Bound block 2-51

Check Static Range block 2-54

Check Static Upper Bound block 2-57

Chirp Signal block 2-60

`clear` debug command 7-6

Clock block 2-62

`close_system` command 4-12`clutch_demo` 2-164

Combinatorial Logic block 2-64

combining input lines into vector line 2-236

`compare_model` command 4-14

Complex to Magnitude-Angle block 2-68

Complex to Real-Imag block 2-69

concatenating matrices 2-220

Configurable Subsystem block 2-70

Constant block 2-74

constant value

generating 2-74

`continue` debug command 7-7

Continuous block library

block parameters 8-10

control flow diagrams

Action subsystem 2-5

do-while

While Iterator block 2-375

for

For Iterator block 2-135

if-else

If block 2-168

switch

Switch Case block 2-341

while

While Iterator block 2-375

Coulomb and Viscous Friction block 2-77

Coulomb friction 2-77

Create Subsystem menu item 2-330

current block

getting pathname 4-23

handle 4-24

current system

getting pathname 4-25



**D**

Data Store Memory block 2-79

Data Store Read block 2-82

Data Store Write block 2-84

Data Type Conversion block 2-86

Dead Zone block 2-88

deadband 2-17

debug commands

ashow 7-1

atrace 7-2

bafter 7-3

break 7-4

bshow 7-5

clear 7-6

continue 7-7

disp 7-8

emode 7-10

help 7-9

ishow 7-11

minor 7-12

nanbreak 7-13

next 7-14

probe 7-15

quit 7-16

run 7-17

slist 7-18

states 7-19

status 7-20

step 7-21

stop 7-22

systems 7-23

tbreak 7-24

trace 7-25

undisp 7-26

untrace 7-27

xbreak 7-28

zcbreak 7-29

zclist 7-30

decimation factor 5-12

decision tables

modeling 2-64

delaying input by variable amount 2-372

delete\_block command 4-15

delete\_line command 4-16

delete\_param command 4-17

demos

clutch 2-164

fohdemo 2-133

hardstop 2-164

lorenzs 2-383

Demos library 1-20

Demux block 2-90

Derivative block 2-96

accuracy of 2-96

derivatives

calculating 2-96

limiting 2-264

differential/algebraic systems

modeling 2-8

Digital Clock block 2-98

Discontinuities block library

block parameters 8-12

Discrete block library

block parameters 8-13

Discrete Filter block 2-105

Discrete State-Space block 2-107

discrete state-space model 3-3

Discrete Transfer Fcn block 2-116

Discrete Zero-Pole block 2-118

Discrete-Time Integrator block 2-109

discrete-time systems

linearization 3-3

disp command 6-5

disp debug command 7-8

Display block 2-120

as floating display 2-120

displaying

signals graphically 2-288

dlinmod function 3-2, 3-3

DocBlock block 2-123

Dot Product block 2-124

dpoly command 6-6

## E

eigenvalues of linearized matrix 3-3

emode debug command 7-10

Enable block 2-126

Enabled and Triggered Subsystem block 2-128

Enabled Subsystem block 2-129

enabled subsystems

Enable block 2-126

expressions

applying to block inputs 2-130

MATLAB Fcn block 2-218

external inputs

flag 5-6

from workspace 2-175

ut 5-8

Extras block library 1-19

## F

Fcn block 2-130

compared to Math Function block 2-216

compared to Rounding Function block 2-284

compared to Trigonometric Function block  
2-366

files

reading from 2-143

writing to

To File block 2-349

find\_system command 4-18

finding objects 4-18

Finite Impulse Response filter 2-105

finite-state machines

implementing 2-64

First-Order Hold block 2-133

compared to Zero-Order Hold block 2-133

fixed step size 5-13

flip-flops

implementing 2-64

floating scope

definition 2-295

Floating Scope block 2-288

fohdemo demo 2-133

for control flow diagram

creating 2-135

For Iterator block 2-135

For Iterator Subsystem block 2-140

For subsystems

creating 2-135

Forward Euler method 2-109

Forward Rectangular method 2-109

fprintf command 6-8

From block 2-141  
From File block 2-143  
From Workspace block 2-146  
Function-Call Generator block 2-150  
Function-Call Subsystem block 2-152

## G

gain  
    varying during simulation 2-322  
gcb command 4-23  
gcbh command 4-24  
gcs command 4-25  
get\_param command 4-26  
global Goto tag visibility 2-159  
Goto block 2-159  
Goto Tag Visibility block 2-162  
graphics  
    displaying on mask icon 6-11  
Greek letters  
    displaying on mask icons 6-5, 6-13  
Ground block 2-163

## H

handle of current block 4-24  
hardstop demo 2-164  
help debug command 7-9  
Hide Name menu item  
    suppressing display of port label 2-239  
Hit Crossing block 2-164  
hybrid systems  
    linearization 3-3

## I

IC block 2-166  
If Action Subsystem block 2-173  
If block 2-168  
if-else control flow diagram  
    creating 2-168  
image  
    displaying on mask icon 6-9, 6-10  
image command 6-9  
inf values  
    in mask plotting commands 6-11  
Infinite Impulse Response filter 2-105  
initial conditions  
    setting 2-166  
initial states 5-13  
initial step size 5-13  
Inport block 2-174  
    linmod function 3-2  
Inport blocks  
    in subsystem 2-330  
input ports  
    unconnected 2-163  
inputs  
    applying expressions to 2-130  
    applying MATLAB function to  
        Fcn block 2-130  
        MATLAB Fcn block 2-218  
    combining into vector line 2-236  
    delaying by variable amount 2-372  
    external 5-8  
    from outside system 2-174  
    from previous time step 2-222  
    from workspace 2-175  
    generating step between two levels 2-327  
    interpolated mapping 2-207  
    logical operations on 2-192

inputs (*continued*)  
    multiplying block inputs during simulation  
        2-322  
    outputting minimum or maximum 2-228  
    passing through stair-step function 2-258  
    piecewise linear mapping of two 2-202  
    plotting 2-383  
    reading from file 2-143  
    width of 2-382  
integration  
    block input 2-179  
    discrete-time 2-109  
Integrator block 2-179  
interpolated mapping 2-207  
inverting signal bits 2-23  
ishow debug command 7-11

## J

Jacobians 3-3

## L

left-hand approximation 2-109  
limiting  
    signals 2-286  
limiting derivative of signal 2-264  
limiting integral 2-181  
linear models  
    extracting  
        linmod function 3-2  
linearization  
    discrete-time systems 3-3  
    linmod function 3-2  
linearized matrix  
    eigenvalues 3-3

## lines

    adding 4-7  
    deleting 4-16  
linmod function 3-2  
    Transport Delay block 2-358  
linmod2 function 3-2  
local Goto tag visibility 2-159  
logic circuits  
    modeling 2-64  
Logical Operator block 2-192  
Look-Up Table (2-D) block 2-202  
Look-Up Table (n-D) block 2-207  
Look-Up Tables block library  
    block parameters 8-15, 8-25  
lorenzs demo 2-383

## M

Magnitude-Angle to Complex block 2-213  
Manual Switch block 2-215  
mask icon  
    displaying graphics on 6-11  
    displaying image on 6-9, 6-10  
    displaying port label on 6-12  
    displaying symbols and Greek letters on 6-13  
    displaying text on 6-5, 6-8, 6-13  
    displaying transfer function on 6-6  
mask icons  
    displaying symbols and Greek letters on 6-5  
    question marks in 6-11  
mask parameters  
    undefined 6-7  
masked blocks  
    parameters 8-26  
masked subsystems  
    question marks in icon 6-11

- masking signal bits 2-23
  - Math block library
    - block parameters 8-15
  - Math Function block 2-216
  - mathematical functions
    - performing
      - Math Function block 2-216
      - Rounding Function block 2-284
      - Trigonometric Function block 2-366
  - mathematical symbols
    - displaying on mask icons 6-5, 6-13
  - MATLAB Fcn block 2-218
  - MATLAB functions
    - applying to block input
      - Fcn block 2-130
      - MATLAB Fcn block 2-218
  - matrices
    - concatenation 2-220
    - writing to 2-351
  - Matrix Concatenation block 2-220
  - maximum number of output rows 5-13
  - maximum order of ode15s solver 5-13
  - maximum step size
    - simset command 5-13
  - mdl file 9-2
  - Memory block 2-222
  - memory region
    - shared
      - Data Store Memory block 2-79
      - Data Store Read block 2-82
      - Data Store Write block 2-84
  - Merge block 2-224
  - MinMax block 2-228
  - minor debug command 7-12
  - model command 5-5
  - model files 9-2
  - Model Info block 2-230
  - model parameters
    - table 8-2
  - models
    - closing 4-10
    - comparing
      - compare\_model command 4-14
    - creating
      - new\_system command 4-28
    - getting name 4-11
    - parameters 8-2
    - replacing blocks 4-30
    - simulating 5-7
  - Model-Wide Utilities block library
    - block parameters 8-17
  - multiplying block inputs
    - during simulation 2-322
  - multirate systems
    - linearization 3-3
  - Mux block 2-236
- ## N
- Nan values
    - in mask plotting commands 6-11
  - nanbreak debug command 7-13
  - new\_system command 4-28
  - newline in block name 4-4, 5-3, 6-3
  - next debug command 7-14
  - nonlinear systems
    - spectral analysis of 2-60
  - normally distributed random numbers 2-262
  - NOT operator 2-23

**O**

- objects
  - finding 4-18
  - specifying path 4-4, 5-3, 6-3
- ode113 solver
  - Memory block 2-222
- ode15s solver
  - maximum order property 5-13
  - Memory block 2-222
- open\_system command 4-29
- opening
  - block dialog boxes 4-29
  - Simulink Library Browser 4-35
  - system windows 4-29
- operating point 3-2
- options structure
  - getting values 5-16
  - setting values 5-12
- OR operator 2-23
- Outport block 2-238
  - linmod function 3-2
- Outport blocks
  - in subsystem 2-330
- output
  - maximum rows 5-13
  - outside system 2-238
  - refine factor 5-14
  - selected elements of input vector 2-302
  - selected information about the signal on input 2-252
  - specifying points 5-14
  - switching between two inputs 2-215
  - values
    - displaying 2-120
  - variables 5-14
  - writing to file
    - To File block 2-349

- writing to workspace
  - To Workspace block 2-351
- zero within range 2-88

- output ports
  - capping unconnected 2-346, 2-363

**P**

- parameters
  - adding 4-9
  - block
    - list 8-7
  - deleting 4-17
  - getting values of 4-26
  - masked blocks 8-26
  - model 8-2
  - setting values of
    - set\_param command 4-33
- patch command 6-10
- path
  - specifying 4-4, 5-3, 6-3
- phase-shifted wave 2-310, 2-312
- piecewise linear mapping
  - two inputs 2-202
- piecewise linear signal
  - generating
    - Signal Builder block 2-310
- plot command 6-11
- plotting input signals
  - Scope block 2-288
  - XY Graph block 2-383
- plotting simulation data 5-10
- port label
  - displaying on mask icon 6-12
- port labels
  - suppressing display 2-239
- port\_label command 6-12

**Ports & Subsystems block library**

- block parameters 8-17

- probe debug command 7-15

- programmable logic arrays

- modeling 2-64

- properties of Scope block 2-295

- Pulse Generator block 2-254

**Q**

- Quantizer block 2-258

- question marks in mask icon 6-11

- quit debug command 7-16

**R**

- random noise

- generating 2-262

- Random Number block 2-262

- and Band-Limited White Noise block 2-21

- compared to Band-Limited White Noise block 2-262

- random numbers

- generating normally distributed 2-21

- normally distributed 2-262

- uniformly distributed 2-368

- Rate Limiter block 2-264

- Rate Transition block 2-266

- reading data

- from data store 2-82

- from file 2-143

- from workspace 2-146

- Real-Imag to Complex block 2-269

- refine factor

- simset command 5-14

- region of zero output 2-88

- regular expressions 4-20

- relative tolerance 5-14

- Repeating Sequence block 2-279

- repeating signals 2-279

- replace\_block command 4-30

- replacing blocks in model 4-30

- Reshape block 2-281

- right-hand approximation 2-110

- Rounding Function block 2-284

- run debug command 7-17

**S**

- sample-and-hold

- applying to block input 2-222

- sampling interval

- generating simulation time 2-98

- Saturation block 2-286

- save\_system command 4-32

- sawtooth wave

- generating 2-311

- Scope axes

- autoscaling 2-291

- Scope block 2-288

- properties 2-295

- saving axes settings 2-294

- scoped Goto tag visibility 2-159

- Selector block 2-302

- separating vector signal 2-90

- sequence of signals 2-254

- sequential circuits

- implementing 2-66

- set\_param command 4-33

- setting parameter values 4-33

- S-Function block 2-306

- S-Function Builder block 2-308

- shared data store
  - Data Store Memory block 2-79
  - Data Store Read block 2-82
  - Data Store Write block 2-84
- SHIFT\_LEFT operator 2-23
- SHIFT\_RIGHT operator 2-23
- shifting signal bits 2-23
- Sign block 2-309
- Signal Attributes block library
  - block parameters 8-18
- Signal Generator block 2-311
- Signal Inspection block 2-252
- Signal Routing block library
  - block parameters 8-19
- Signal Specification block 2-314
- signals
  - displaying graphically 2-288
  - displaying vector 2-289
  - displaying X-Y plot of 2-383
  - generating pulses 2-254
  - limiting 2-286
  - limiting derivative of 2-264
  - passed from Goto block 2-141
  - passing to From block 2-159
  - plotting
    - Scope block 2-288
    - XY Graph block 2-383
  - repeating 2-279
- sim command 5-7
- simget command 5-16
- simplot command
  - plotting simulation data 5-10
- simset command 5-12
- simulating models 5-7
- simulation
  - parameters
    - specifying using simset command 5-12
  - stopping
    - Stop Simulation block 2-329
- simulation time
  - generating at sampling interval 2-98
  - outputting 2-62
- simulink command 4-35
- Simulink Library Browser
  - opening 4-35
- sine wave
  - generating
    - Signal Generator block 2-311
    - Sine Wave block 2-317
  - generating with increasing frequency
    - Chirp Signal block 2-60
- Sine Wave block 2-317
- Sinks block library
  - block parameters 8-20
- slash in block name 4-5, 5-4, 6-4
- Slider Gain block 2-322
- slist debug command 7-18
- solvers
  - properties
    - specifying 5-12
  - specifying using simset command 5-14
- Sources block library
  - block parameters 8-22
- spectral analysis of nonlinear systems 2-60
- square wave
  - generating 2-311
- ss2tf function 3-4
- ss2zp function 3-4
- stair-step function
  - passing signal through 2-258



- state derivatives
  - setting to zero 3-5
- state space in discrete system 2-107
- states
  - initial 5-13
  - outputting 5-14
  - resetting 2-181
  - saving at end of simulation 5-13
  - specifying absolute tolerance for 2-186
- states debug command 7-19
- State-Space block 2-324
- status debug command 7-20
- Step block 2-327
- step debug command 7-21
- stop debug command 7-22
- Stop Simulation block 2-329
- stopping simulation 2-329
- Subsystem block 2-330
- subsystems
  - and Inport blocks 2-174
  - enabled 2-126
  - specifying path 4-4, 5-3, 6-3
- Sum block 2-334
- Switch Case Action Subsystem block 2-345
- switch control flow diagram
  - creating 2-341
- switching output between inputs
  - Manual Switch block 2-215
- switching output between two inputs 2-215
- System section of mdl file 9-3
- system windows
  - closing 4-12
- systems
  - current 4-25
  - saving 4-32
  - specifying path 4-4, 5-3, 6-3
- systems debug command 7-23

**T**

- tbreak debug command 7-24
- Terminator block 2-346
- TeX formatting commands
  - using in mask icon text 6-5, 6-13
- text command 6-13
- tf2ss utility
  - converting Transfer Fcn to state-space form 2-355
- time delay
  - simulating 2-358
- Time-Based Linearization block 2-347
- To File block 2-349
- To Workspace block 2-351
- trace debug command 7-25
- tracing facilities 5-14
- Transfer Fcn block 2-355
- transfer function
  - displaying on mask icon 6-6
- transfer function form
  - converting to 3-4
- transfer functions
  - discrete 2-116
  - linear 2-355
  - poles and zeros 2-387
    - discrete 2-118
- Transport Delay block 2-358
- Trapezoidal method 2-110
- Trigger block 2-361
- Trigger-Based Linearization block 2-363
- Triggered Subsystem block 2-365
- triggered subsystems
  - Trigger block 2-361
- Trigonometric Function block 2-366
- trim function 3-5
- truth tables
  - implementing 2-64

**U**

- unconnected input ports 2-163
- unconnected output ports
  - capping 2-346, 2-363
- undisp debug command 7-26
- Uniform Random Number block 2-368
  - compared to Band-Limited White Noise block 2-368
- uniformly distributed random numbers 2-368
- Unit Delay block
  - compared to Transport Delay block 2-358
- untraced debug command 7-27
- Update Diagram menu item
  - changing block parameters during simulation 4-33

**V**

- variable time delay 2-372
- Variable Transport Delay block 2-372
- vdp model
  - Scope block 2-290
- vector signals
  - displaying 2-289
  - generating from inputs 2-236
  - separating 2-90
- viscous friction 2-77
- visibility of Goto tag 2-162

**W**

- while control flow diagram
  - creating 2-375
- While Iterator block 2-375
- While Iterator Subsystem block 2-381
- While subsystems
  - creating 2-375

- white noise
  - generating 2-21
- Width block 2-382
- workspace
  - destination 5-13
  - reading data from 2-146
  - source 5-14
  - writing output to 2-351
- writing data to data store 2-84
- writing output to file 2-349
- writing output to workspace 2-351

**X**

- xbreak debug command 7-28
- XOR operator 2-23
- XY Graph block 2-383

**Z**

- zcbreak debug command 7-29
- zclist debug command 7-30
- zero crossings
  - detecting
    - Hit Crossing block 2-164
    - simset command 5-15
- zero output in region
  - generating 2-88
- Zero-Order Hold block
  - compared to First-Order Hold block 2-133
- Zero-Pole block 2-387
- zero-pole form
  - converting to 3-4
- zooming in on displayed data 2-292