

RELEASE  
**14**

***Learning***

# SIMULINK® 6

MATLAB®  
& SIMULINK®  
STUDENT VERSION

 The MathWorks

## **How to Contact The MathWorks:**



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

Web  
Newsgroup



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

Product enhancement suggestions  
Bug reports  
Documentation error reports

ISBN 0-9755787-7-4

### *Learning Simulink*

© COPYRIGHT 1999 - 2005 by The MathWorks, Inc.

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

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

### **Trademarks**

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks of The MathWorks, Inc. Other product or brand names are trademarks or registered trademarks of their respective holders.

### **Patents**

The MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## **Revision History**

August 1999	First printing	New manual
January 2001	Second printing	Revised for Simulink 4.0 (Release 12)
November 2002	Third printing	Revised for Simulink 5.0 (Release 13)
July 2004	Fourth printing	Revised for Simulink 6.0 (Release 14)
December 2005	Fifth printing	Revised for Simulink 6.3 (Release 14SP3)



## Introducing Simulink

1

<b>About the Student Version</b> .....	<b>1-2</b>
Student Use Policy .....	1-2
Student Version Activation .....	1-2
<b>Obtaining Additional MathWorks Products</b> .....	<b>1-4</b>
<b>Getting Started with Simulink</b> .....	<b>1-5</b>
<b>Finding Reference Information</b> .....	<b>1-6</b>
<b>Troubleshooting</b> .....	<b>1-7</b>
<b>Other Resources</b> .....	<b>1-8</b>
Documentation .....	1-8
MathWorks Web Site .....	1-8
MathWorks Academia Web Site .....	1-8
MATLAB & Simulink Based Books .....	1-8
MathWorks Store .....	1-8
MATLAB Central – File Exchange/Newsgroup Access .....	1-9
Technical Support .....	1-9
<b>Differences Between the Student and Professional Versions</b> .....	<b>1-10</b>

## How Simulink Works

2

<b>Introduction</b> .....	<b>2-2</b>
---------------------------	------------

<b>Modeling Dynamic Systems</b> . . . . .	<b>2-3</b>
Block Diagram Semantics . . . . .	<b>2-3</b>
Creating Models . . . . .	<b>2-4</b>
Time . . . . .	<b>2-4</b>
States . . . . .	<b>2-4</b>
Block Parameters . . . . .	<b>2-8</b>
Tunable Parameters . . . . .	<b>2-8</b>
Block Sample Times . . . . .	<b>2-9</b>
Custom Blocks . . . . .	<b>2-10</b>
Systems and Subsystems . . . . .	<b>2-10</b>
Signals . . . . .	<b>2-11</b>
Block Methods . . . . .	<b>2-12</b>
Model Methods . . . . .	<b>2-13</b>
<b>Simulating Dynamic Systems</b> . . . . .	<b>2-14</b>
Model Compilation . . . . .	<b>2-14</b>
Link Phase . . . . .	<b>2-14</b>
Simulation Loop Phase . . . . .	<b>2-15</b>
Solvers . . . . .	<b>2-17</b>
Zero-Crossing Detection . . . . .	<b>2-19</b>
Algebraic Loops . . . . .	<b>2-24</b>
<b>Modeling and Simulating Discrete Systems</b> . . . . .	<b>2-31</b>
Specifying Sample Time . . . . .	<b>2-31</b>
Purely Discrete Systems . . . . .	<b>2-34</b>
Multirate Systems . . . . .	<b>2-34</b>
Determining Step Size for Discrete Systems . . . . .	<b>2-36</b>
Sample Time Propagation . . . . .	<b>2-37</b>
Constant Sample Time . . . . .	<b>2-39</b>
Mixed Continuous and Discrete Systems . . . . .	<b>2-41</b>

## Simulink Basics

# 3

<b>Starting Simulink</b> . . . . .	<b>3-2</b>
<b>Opening Models</b> . . . . .	<b>3-3</b>
Opening Models with Different Character Encodings . . . . .	<b>3-3</b>
Avoiding Initial Model Open Delay . . . . .	<b>3-4</b>

<b>Model Editor</b> .....	<b>3-5</b>
Editor Components .....	<b>3-5</b>
Undoing a Command .....	<b>3-6</b>
Zooming Block Diagrams .....	<b>3-7</b>
Panning Block Diagrams .....	<b>3-7</b>
View Command History .....	<b>3-7</b>
<b>Updating a Block Diagram</b> .....	<b>3-9</b>
<b>Saving a Model</b> .....	<b>3-11</b>
Saving Models with Different Character Encodings .....	<b>3-12</b>
Saving a Model in Earlier Formats .....	<b>3-12</b>
<b>Printing a Block Diagram</b> .....	<b>3-15</b>
Print Dialog Box .....	<b>3-15</b>
Print Command .....	<b>3-17</b>
Specifying Paper Size and Orientation .....	<b>3-18</b>
Positioning and Sizing a Diagram .....	<b>3-18</b>
<b>Generating a Model Report</b> .....	<b>3-20</b>
Model Report Options .....	<b>3-21</b>
<b>Summary of Mouse and Keyboard Actions</b> .....	<b>3-23</b>
Model Viewing Shortcuts .....	<b>3-23</b>
Block Editing Shortcuts .....	<b>3-24</b>
Line Editing Shortcuts .....	<b>3-25</b>
Signal Label Editing Shortcuts .....	<b>3-25</b>
Annotation Editing Shortcuts .....	<b>3-26</b>
<b>Ending a Simulink Session</b> .....	<b>3-27</b>

## Creating a Model

**4**

---

<b>Creating an Empty Model</b> .....	<b>4-2</b>
Creating a Model Template .....	<b>4-2</b>

<b>Selecting Objects</b> .....	<b>4-4</b>
Selecting an Object .....	<b>4-4</b>
Selecting Multiple Objects .....	<b>4-4</b>
<b>Specifying Block Diagram Colors</b> .....	<b>4-6</b>
Choosing a Custom Color .....	<b>4-6</b>
Defining a Custom Color .....	<b>4-7</b>
Specifying Colors Programmatically .....	<b>4-7</b>
Displaying Sample Time Colors .....	<b>4-8</b>
<b>Connecting Blocks</b> .....	<b>4-10</b>
Automatically Connecting Blocks .....	<b>4-10</b>
Manually Connecting Blocks .....	<b>4-13</b>
Disconnecting Blocks .....	<b>4-18</b>
<b>Annotating Diagrams</b> .....	<b>4-19</b>
Annotations Properties Dialog Box .....	<b>4-21</b>
Annotations API .....	<b>4-23</b>
<b>Creating Subsystems</b> .....	<b>4-24</b>
Creating a Subsystem by Adding the Subsystem Block .....	<b>4-24</b>
Creating a Subsystem by Grouping Existing Blocks .....	<b>4-25</b>
Model Navigation Commands .....	<b>4-26</b>
Window Reuse .....	<b>4-27</b>
Labeling Subsystem Ports .....	<b>4-27</b>
<b>Creating Conditionally Executed Subsystems</b> .....	<b>4-29</b>
Enabled Subsystems .....	<b>4-30</b>
Triggered Subsystems .....	<b>4-34</b>
Triggered and Enabled Subsystems .....	<b>4-37</b>
<b>Using Callback Functions</b> .....	<b>4-41</b>
Tracing Callbacks .....	<b>4-41</b>
Creating Model Callback Functions .....	<b>4-41</b>
Creating Block Callback Functions .....	<b>4-42</b>
Port Callback Parameters .....	<b>4-45</b>

<b>About Blocks</b> .....	<b>5-2</b>
Block Data Tips .....	<b>5-2</b>
Virtual Blocks .....	<b>5-2</b>
<b>Editing Blocks</b> .....	<b>5-4</b>
Copying and Moving Blocks from One Window to Another .....	<b>5-4</b>
Moving Blocks in a Model .....	<b>5-6</b>
Copying Blocks in a Model .....	<b>5-6</b>
Deleting Blocks .....	<b>5-6</b>
<b>Working with Block Parameters</b> .....	<b>5-7</b>
Displaying a Block's Parameter Dialog Box .....	<b>5-8</b>
Specifying Parameter Values .....	<b>5-8</b>
Working with Tunable Parameters .....	<b>5-8</b>
Block Properties Dialog Box .....	<b>5-10</b>
<b>Changing a Block's Appearance</b> .....	<b>5-15</b>
Changing the Orientation of a Block .....	<b>5-15</b>
Resizing a Block .....	<b>5-15</b>
Displaying Parameters Beneath a Block .....	<b>5-16</b>
Using Drop Shadows .....	<b>5-16</b>
Manipulating Block Names .....	<b>5-17</b>
Specifying a Block's Color .....	<b>5-18</b>
<b>Displaying Block Outputs</b> .....	<b>5-19</b>
Enabling Port Values Display .....	<b>5-19</b>
Port Values Display Options .....	<b>5-20</b>
<b>Working with Block Libraries</b> .....	<b>5-21</b>
Terminology .....	<b>5-21</b>
Simulink Block Library .....	<b>5-21</b>
Creating a Library Link .....	<b>5-22</b>
Disabling Library Links .....	<b>5-23</b>
Modifying a Linked Subsystem .....	<b>5-23</b>
Propagating Link Modifications .....	<b>5-24</b>
Updating a Linked Block .....	<b>5-25</b>
Breaking a Link to a Library Block .....	<b>5-25</b>

Finding the Library Block for a Reference Block .....	<b>5-26</b>
Library Link Status .....	<b>5-26</b>
Displaying Library Links .....	<b>5-27</b>
Getting Information About Library Blocks .....	<b>5-28</b>
Browsing Block Libraries .....	<b>5-28</b>

## Working with Signals

# 6

---

<b>Signal Basics</b> .....	<b>6-2</b>
About Signals .....	<b>6-2</b>
Creating Signals .....	<b>6-2</b>
Signal Line Styles .....	<b>6-2</b>
Signal Labels .....	<b>6-3</b>
Displaying Signal Values .....	<b>6-4</b>
Signal Dimensions .....	<b>6-4</b>
Complex Signals .....	<b>6-4</b>
Virtual Signals .....	<b>6-5</b>
Control Signals .....	<b>6-8</b>
Signal Buses .....	<b>6-9</b>
Checking Signal Connections .....	<b>6-12</b>
Signal Glossary .....	<b>6-13</b>
<b>Determining Output Signal Dimensions</b> .....	<b>6-14</b>
Determining the Output Dimensions of Source Blocks ...	<b>6-14</b>
Determining the Output Dimensions of Nonsource	
Blocks .....	<b>6-15</b>
Signal and Parameter Dimension Rules .....	<b>6-15</b>
Scalar Expansion of Inputs and Parameters .....	<b>6-16</b>
<b>Displaying Signal Properties</b> .....	<b>6-19</b>
Display Options .....	<b>6-19</b>
Signal Names .....	<b>6-21</b>
Signal Labels .....	<b>6-21</b>
Displaying Signals Represented by Virtual Signals .....	<b>6-22</b>

<b>Simulation Basics</b> .....	<b>7-2</b>
Controlling Execution of a Simulation .....	<b>7-2</b>
Interacting with a Running Simulation .....	<b>7-5</b>
<b>Specifying a Simulation Start and Stop Time</b> .....	<b>7-6</b>
<b>Choosing a Solver</b> .....	<b>7-7</b>
Choosing a Solver Type .....	<b>7-7</b>
Choosing a Fixed-Step Solver .....	<b>7-8</b>
Choosing a Variable-Step Solver .....	<b>7-13</b>
<b>Importing and Exporting Simulation Data</b> .....	<b>7-17</b>
Importing Data from the MATLAB Workspace .....	<b>7-17</b>
Exporting Data to the MATLAB Workspace .....	<b>7-22</b>
Importing and Exporting States .....	<b>7-24</b>
Limiting Output .....	<b>7-26</b>
Specifying Output Options .....	<b>7-26</b>
<b>Configuration Sets</b> .....	<b>7-29</b>
Configuration Set Components .....	<b>7-29</b>
The Active Set .....	<b>7-29</b>
Displaying Configuration Sets .....	<b>7-30</b>
Activating a Configuration Set .....	<b>7-30</b>
Copying, Deleting, and Moving Configuration Sets .....	<b>7-31</b>
Copying Configuration Set Components .....	<b>7-31</b>
Creating Configuration Sets .....	<b>7-32</b>
Setting Values in Configuration Sets .....	<b>7-32</b>
Model Configuration Dialog Box .....	<b>7-32</b>
Model Configuration Preferences Dialog Box .....	<b>7-34</b>
<b>Configuration Parameters Dialog Box</b> .....	<b>7-36</b>
<b>Diagnosing Simulation Errors</b> .....	<b>7-38</b>
Simulation Diagnostics Viewer .....	<b>7-38</b>
<b>Improving Simulation Performance and Accuracy</b> .....	<b>7-40</b>
Speeding Up the Simulation .....	<b>7-40</b>

## Exploring, Searching, and Browsing Models

8

<b>The Model Explorer</b> .....	<b>8-2</b>
Setting the Model Explorer's Font Size .....	8-3
Model Hierarchy Pane .....	8-3
Contents Pane .....	8-5
Dialog Pane .....	8-9
Main Toolbar .....	8-9
Search Bar .....	8-12
<b>The Finder</b> .....	<b>8-16</b>
Filter Options .....	8-18
Search Criteria .....	8-18
<b>The Model Browser</b> .....	<b>8-22</b>
Navigating with the Mouse .....	8-23
Navigating with the Keyboard .....	8-23
Showing Library Links .....	8-23
Showing Masked Subsystems .....	8-23

## Creating Masked Subsystems

9

<b>About Masks</b> .....	<b>9-2</b>
Mask Features .....	9-2
Creating Masks .....	9-5
<b>Masked Subsystem Example</b> .....	<b>9-6</b>
Creating Mask Dialog Box Prompts .....	9-7
Creating the Block Description and Help Text .....	9-10
Creating the Block Icon .....	9-10

<b>Masking a Subsystem</b> .....	<b>9-12</b>
<b>Mask Editor</b> .....	<b>9-14</b>
Icon Pane .....	9-16
Parameters Pane .....	9-19
Control Types .....	9-23
Initialization Pane .....	9-25
Documentation Pane .....	9-28
Changing Default Values for Mask Parameters in a Library .....	9-30
<b>Linking Mask Parameters to Block Parameters</b> .....	<b>9-32</b>

## Simulink Debugger

# 10

---

<b>Introduction</b> .....	<b>10-2</b>
<b>Using the Debugger's Graphical User Interface</b> .....	<b>10-3</b>
Toolbar .....	10-4
Breakpoints Pane .....	10-5
Simulation Loop Pane .....	10-6
Outputs Pane .....	10-7
Sorted List Pane .....	10-8
Status Pane .....	10-9
<b>Using the Debugger's Command-Line Interface</b> .....	<b>10-10</b>
Method ID .....	10-10
Block ID .....	10-10
Accessing the MATLAB Workspace .....	10-10
<b>Getting Online Help</b> .....	<b>10-12</b>
<b>Starting the Debugger</b> .....	<b>10-13</b>
<b>Starting a Simulation</b> .....	<b>10-14</b>

<b>Running a Simulation Step by Step</b> .....	<b>10-15</b>
Block Data Output .....	<b>10-16</b>
Stepping Commands .....	<b>10-16</b>
Continuing a Simulation .....	<b>10-17</b>
Running a Simulation Nonstop .....	<b>10-19</b>
Debug Pointer .....	<b>10-20</b>
<b>Setting Breakpoints</b> .....	<b>10-22</b>
Setting Unconditional Breakpoints .....	<b>10-22</b>
Setting Conditional Breakpoints .....	<b>10-24</b>
<b>Displaying Information About the Simulation</b> .....	<b>10-28</b>
Displaying Block I/O .....	<b>10-28</b>
Displaying Algebraic Loop Information .....	<b>10-30</b>
Displaying System States .....	<b>10-31</b>
Displaying Solver Information .....	<b>10-31</b>
<b>Displaying Information About the Model</b> .....	<b>10-32</b>
Displaying a Model's Sorted Lists .....	<b>10-32</b>
Displaying a Block .....	<b>10-33</b>

## Block Libraries

---

# A

<b>Commonly Used</b> .....	<b>A-3</b>
<b>Continuous</b> .....	<b>A-5</b>
<b>Discontinuities</b> .....	<b>A-6</b>
<b>Discrete</b> .....	<b>A-7</b>
<b>Logic and Bit Operations</b> .....	<b>A-8</b>
<b>Lookup Tables</b> .....	<b>A-10</b>
<b>Math Operations</b> .....	<b>A-11</b>

<b>Model Verification</b> .....	<b>A-13</b>
<b>Model-Wide Utilities</b> .....	<b>A-15</b>
<b>Ports &amp; Subsystems</b> .....	<b>A-16</b>
<b>Signal Attributes</b> .....	<b>A-18</b>
<b>Signal Routing</b> .....	<b>A-19</b>
<b>Sinks</b> .....	<b>A-20</b>
<b>Sources</b> .....	<b>A-21</b>
<b>User-Defined Functions</b> .....	<b>A-23</b>
<b>Additional Discrete</b> .....	<b>A-24</b>
<b>Additional Math</b> .....	<b>A-26</b>
<b>Simulink Extras</b> .....	<b>A-27</b>

## **Index**



# Introducing Simulink

---

This chapter introduces the version of Simulink® included in MATLAB & Simulink Student Version.

About the Student Version (p. 1-2)	Description of the version of Simulink included in MATLAB & Simulink Student Version.
Obtaining Additional MathWorks Products (p. 1-4)	How to acquire other products that enhance the version of Simulink included in MATLAB & Simulink Student Version
Getting Started with Simulink (p. 1-5)	Basic steps for using Simulink
Finding Reference Information (p. 1-6)	How to learn more about Simulink
Troubleshooting (p. 1-7)	Getting help with and reporting problems
Other Resources (p. 1-8)	Additional sources of information for the version of Simulink included in MATLAB & Simulink Student Version
Differences Between the Student and Professional Versions (p. 1-10)	How the version of Simulink included in MATLAB & Simulink Student Version differs from the professional version of Simulink.

## About the Student Version

Simulink is an interactive tool for modeling, simulating, and analyzing dynamic systems, including controls, signal processing, communications, and other complex systems. The version of Simulink included in MATLAB & Simulink Student Version provides all of the features of professional Simulink, with model sizes up to 1000 blocks. It gives you immediate access to the high-performance simulation power you need.

Simulink is a key member of the MATLAB® family of products used in a broad range of industries, including automotive, aerospace, electronics, environmental, telecommunications, computer peripherals, finance, and medical. More than one million technical professionals at the world's most innovative technology companies, government research labs, financial institutions, and at more than 3500 universities, rely on MATLAB and Simulink as the fundamental tools for their engineering and scientific work.

### **Student Use Policy**

This MATLAB & Simulink Student Version License is for use in conjunction with courses offered at degree-granting institutions. The MathWorks offers this license as a special service to the student community and asks your help in seeing that its terms are not abused.

To use this Student License, you must be a student either enrolled in a degree-granting institution or participating in a continuing education program at a degree-granting educational university.

You may not use this Student License at a company or government lab. Also, you may not use it if you are an instructor at a university, or for research, commercial, or industrial purposes. In these cases, you can acquire the appropriate professional or academic license by contacting The MathWorks.

### **Student Version Activation**

Activation is a secure process that verifies licensed student users. This process validates the serial number and ensures that it is not used on more systems than allowed by the MathWorks End User License Agreement.

The activation technology is designed to provide an easier and more effective way for students to authenticate and use their product than prior releases of MATLAB & Simulink Student Version.

The quickest way to activate your software is to use the activation program that starts following product installation. The activation program will guide you through the activation process. Alternatively, you can activate your software on the [mathworks.com](http://mathworks.com) Web site.

Activation requires completion of three activities:

- Provide registration information by creating a MathWorks Account.
- Provide your serial number and the Machine ID for the computer you will be installing the software on.
- For those students who did not provide proof of student status at the time of purchase, submission and verification of proof of student status is needed.

For more information on activation, see  
[www.mathworks.com/academia/student\\_version/activation.html](http://www.mathworks.com/academia/student_version/activation.html).

## Obtaining Additional MathWorks Products

Simulink comes with a standard blockset that addresses a wide range of applications. Additional blocksets are available that simplify modeling in specific applications. Other products that address specific simulation applications are available as well. You may purchase and download some of these additional products at special student prices from the MathWorks Store at [www.mathworks.com/store](http://www.mathworks.com/store).

Some of the products you can purchase include

- Communications Blockset
- Stateflow® (A demo version of Stateflow is included with your Student Version.)

For an up-to-date list of available toolboxes and blocksets, visit the MathWorks Store.

---

**Note** The blocksets that are available for use with MATLAB & Simulink Student Version have the same functionality as the full, professional versions. However, the student versions of the toolboxes and blocksets will work only with the Student Version. Likewise, the professional versions of the toolboxes and blocksets will not work with the Student Version.

---

# Getting Started with Simulink

What I Want	What I Should Do
I need to install Simulink.	See Chapter 2, “Installing MATLAB & Simulink Student Version” in the Learning MATLAB book.
I want to start Simulink.	<b>(Microsoft Windows)</b> Double-click the MATLAB icon on your desktop. Click the Simulink icon on the toolbar to start Simulink. <b>(Macintosh OS X)</b> Double-click the MATLAB icon on your desktop. Click the Simulink icon on the toolbar to start Simulink. <b>(Linux)</b> Enter the <code>matlab</code> command at the command prompt. Click the Simulink icon on the toolbar to start Simulink.
I'm new to Simulink and want to learn it quickly.	Start by reading “Getting Started with Simulink” in the online documentation. You'll get a quick overview of how to model, simulate, and analyze dynamic systems. Then read this book for a more in-depth introduction to Simulink. Since Simulink is graphical and interactive, this book encourages you to use it quickly. You can access the rest of the Simulink documentation through the online help facility (Help).
I want to look at some samples of what you can do with Simulink.	There are numerous demonstrations included with Simulink. You can see the demos by clicking <b>Demos</b> in the Help Navigator or selecting <b>Demos</b> from the <b>Help</b> menu. There are Simulink demos for simple models, complex models, and other related products. You also will find a large selection of demos at <a href="http://www.mathworks.com/demos">www.mathworks.com/demos</a> .

## Finding Reference Information

What I Want	What I Should Do
I want to know how to use a specific Simulink block.	Use the online help facility (Help). The Simulink blocks are described under Simulink ( <b>Simulink &gt; Blocks—Categorical List</b> and <b>Simulink &gt; Blocks—Alphabetical List</b> ).
I want to find a block for a specific purpose but I don't know its name.	There are several choices: <ul style="list-style-type: none"><li>From Help, peruse <b>Simulink &gt; Blocks—Categorical List</b> or <b>Simulink &gt; Blocks—Alphabetical List</b>.</li><li>Use <b>Index</b> or <b>Search</b> from Help.</li></ul>
I want to know what blocks are available in a general area.	Use Help to view <b>Simulink &gt; Blocks—Categorical List</b> under Simulink. Help provides access to the reference pages for the blocks included with Simulink.

## Troubleshooting

What I Want	What I Should Do
I have a Simulink specific problem I want help with.	From Help, select <b>Support and Web Services</b> and then choose <b>Technical Support</b> .
I want to report a bug or make a suggestion.	Use Help or send e-mail to <a href="mailto:bugs@mathworks.com">bugs@mathworks.com</a> or <a href="mailto:suggest@mathworks.com">suggest@mathworks.com</a> .

## Other Resources

### Documentation

When you install MATLAB & Simulink Student Version on your computer, you automatically install the complete online documentation for Simulink. Access this documentation from Help.

---

**Note** References to UNIX in the documentation include both Linux and Mac OS X.

---

### Web-Based Documentation

Documentation for all MathWorks products, including Simulink, is online and available from the Support area of the MathWorks Web site. In addition to tutorials and function reference pages, you can find PDF versions of all the manuals.

### MathWorks Web Site

At [www.mathworks.com](http://www.mathworks.com), you'll find information about MathWorks products and how they are used in education and industry, product demos, and MATLAB and Simulink based books.

### MathWorks Academia Web Site

At [www.mathworks.com/academia](http://www.mathworks.com/academia), you'll find resources for students and instructors for courses in engineering, mathematics, and science.

### MATLAB & Simulink Based Books

At [www.mathworks.com/support/books](http://www.mathworks.com/support/books), you'll find books in many disciplines that use MATLAB and Simulink for examples and assignments.

### MathWorks Store

At [www.mathworks.com/store](http://www.mathworks.com/store), you can purchase add-on products and documentation.

## MATLAB Central – File Exchange/Newsgroup Access

At [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral), you can access the MATLAB Usenet newsgroup (`comp.soft-sys.matlab`) as well as an extensive library of user-contributed files called the MATLAB Central File Exchange. MATLAB Central is also home to the Link Exchange where you can share your favorite links to various educational, personal, and commercial MATLAB Web sites.

The `comp.soft-sys.matlab` newsgroup is for professionals and students who use MATLAB and have questions or comments about it and its associated products. This is an important resource for posing questions and answering queries from other MATLAB users. MathWorks staff also participates actively in this newsgroup.

## Technical Support

At [www.mathworks.com/support](http://www.mathworks.com/support), you can get technical support.

Telephone and e-mail access to our technical support staff is not available for students running the version of Simulink included in MATLAB & Simulink Student Version unless you are experiencing difficulty installing or downloading MATLAB or related products. There are numerous other vehicles of technical support that you can use. The “Resources and Support” section in the CD holder identifies the ways to obtain additional help.

After checking the available MathWorks sources for help, if you still cannot resolve your problem, please contact your instructor. Your instructor should be able to help you. If your instructor needs assistance doing so, telephone and e-mail technical support is available to registered instructors who have adopted MATLAB & Simulink Student Version in their courses.

## Differences Between the Student and Professional Versions

This Student Version differs from the Professional Version in the following respects.

- Models are limited to 1000 blocks.

---

**Note** You may encounter some demos that use more than 1000 blocks. In these cases, a dialog will display stating that the block limit has been exceeded and the demo will not run.

---

- The window title bars include the words  
<Student Version>
- Printouts contain the footer  
Student Version of MATLAB

---

**Note** The Using Simulink documentation, which is accessible from the Help browser, contains all of the information in this book plus additional advanced information.

---

# How Simulink Works

---

The following sections explain how Simulink® models and simulates dynamic systems. This information can be helpful in creating models and interpreting simulation results.

Introduction (p. 2-2)

Brief overview of Simulink.

Modeling Dynamic Systems (p. 2-3)

How Simulink models a dynamic system.

Simulating Dynamic Systems  
(p. 2-14)

How Simulink simulates a dynamic system.

Modeling and Simulating Discrete Systems (p. 2-31)

How Simulink models and simulates discrete systems.

## Introduction

Simulink is a software package that enables you to model, simulate, and analyze systems whose outputs change over time. Such systems are often referred to as dynamic systems. Simulink can be used to explore the behavior of a wide range of real-world dynamic systems, including electrical circuits, shock absorbers, braking systems, and many other electrical, mechanical, and thermodynamic systems. This section explains how Simulink works.

Simulating a dynamic system is a two-step process with Simulink. First, a user creates a block diagram, using the Simulink model editor, that graphically depicts time-dependent mathematical relationships among the system's inputs, states, and outputs. The user then commands Simulink to simulate the system represented by the model from a specified start time to a specified stop time.

# Modeling Dynamic Systems

A Simulink block diagram model is a graphical representation of a mathematical model of a dynamic system. A mathematical model of a dynamic system is described by a set of equations. The mathematical equations described by a block diagram model are known as algebraic, differential, and/or difference equations.

## Block Diagram Semantics

A classic block diagram model of a dynamic system graphically consists of blocks and lines (signals). The history of these block diagram model is derived from engineering areas such as Feedback Control Theory and Signal Processing. A block within a block diagram defines a dynamic system in itself. The relationships between each elementary dynamic system in a block diagram are illustrated by the use of signals connecting the blocks. Collectively the blocks and lines in a block diagram describe an overall dynamic system.

Simulink extends these classic block diagram models by introducing the notion of two classes of blocks, nonvirtual block and virtual blocks. Nonvirtual blocks represent elementary systems. A virtual block is provided for graphical organizational convenience and plays no role in the definition of the system of equations described by the block diagram model. Examples of virtual blocks are the Bus Creator and Bus Selector which are used to reduce block diagram clutter by managing groups of signals as a "bundle." You can use virtual blocks to improve the readability of your models.

In general, block and lines can be used to describe many "models of computations." One example would be a flow chart. A flow chart consists of blocks and lines, but one cannot describe general dynamic systems using flow chart semantics.

The term "time-based block diagram" is used to distinguish block diagrams that describe dynamic systems from that of other forms of block diagrams. In Simulink, we use the term block diagram (or model) to refer to a time-based block diagram unless the context requires explicit distinction.

To summarize the meaning of time-based block diagrams:

- Simulink block diagrams define time-based relationships between signals and state variables. The solution of a block diagram is obtained by evaluating these relationships over time, where time starts at a user specified "start time" and ends at a user specified "stop time." Each evaluation of these relationships is referred to as a time step.
- Signals represent quantities that change over time and are defined for all points in time between the block diagram's start and stop time.
- The relationships between signals and state variables are defined by a set of equations represented by blocks. Each block consists of a set of equations (block methods). These equations define a relationship between the input signals, output signals and the state variables. Inherent in the definition of a equation is the notion of parameters, which are the coefficients found within the equation.

## **Creating Models**

Simulink provides a graphical editor that allows you to create and connect instances of block types (see Chapter 4, “Creating a Model”) selected from libraries of block types (see “Blocks—Alphabetical List” in the online Simulink documentation) via a library browser. Simulink provides libraries of blocks representing elementary systems that can be used as building blocks. The blocks supplied with Simulink are called built-in blocks. Simulink users can also create their own block types and use the Simulink editor to create instances of them in a diagram. User-defined blocks are called custom blocks.

## **Time**

Time is an inherent component of block diagrams in that the results of a block diagram simulation change with time. Put another way, a block diagram represents the instantaneous behavior of a dynamic system. Determining a system’s behavior over time thus entails repeatedly solving the model at intervals, called time steps, from the start of the time span to the end of the time span. Simulink refers to the process of solving a model at successive time steps as simulating the system that the model represents.

## **States**

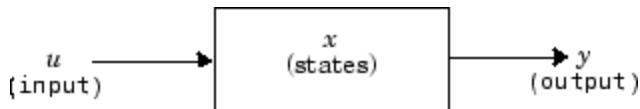
Typically the current values of some system, and hence model, outputs are functions of the previous values of temporal variables. Such variables are

called states. Computing a model's outputs from a block diagram hence entails saving the value of states at the current time step for use in computing the outputs at a subsequent time step. Simulink performs this task during simulation for models that define states.

Two types of states can occur in a Simulink model: discrete and continuous states. A continuous state changes continuously. Examples of continuous states are the position and speed of a car. A discrete state is an approximation of a continuous state where the state is updated (recomputed) using finite (periodic or aperiodic) intervals. An example of a discrete state would be the position of a car shown on a digital odometer where it is updated every second as opposed to continuously. In the limit, as the discrete state time interval approaches zero, a discrete state becomes equivalent to a continuous state.

Blocks implicitly define a model's states. In particular, a block that needs some or all of its previous outputs to compute its current outputs implicitly defines a set of states that need to be saved between time steps. Such a block is said to have states.

The following is a graphical representation of a block that has states:



Blocks that define continuous states include the following standard Simulink blocks:

- Integrator
- State-Space
- Transfer Fcn
- Zero-Pole

The total number of a model's states is the sum of all the states defined by all its blocks. Determining the number of states in a diagram requires parsing the diagram to determine the types of blocks that it contains and then aggregating the number of states defined by each instance of a block

type that defines states. Simulink performs this task during the Compilation phase of a simulation.

## Working with States

Simulink provides the following facilities for determining, initializing, and logging a model's states during simulation:

- The `model` command displays information about the states defined by a model, including the total number of states defined by the model, the block that defines each state, and the initial value of each state.
- The Simulink debugger displays the value of a state at each time step during a simulation, and the Simulink debugger's `states` command displays information about the model's current states (see Chapter 10, "Simulink Debugger").
- The **Data Import/Export** pane of a model's **Configuration Parameters** dialog box (see "Importing and Exporting States" on page 7-24) allows you to specify initial values for a model's states and instruct Simulink to record the values of the states at each time step during simulation as an array or structure variable in the MATLAB® workspace.

## Continuous States

Computing a continuous state entails knowing its rate of change, or derivative. Since the rate of change of a continuous state typically itself changes continuously (i.e., is itself a state), computing the value of a continuous state at the current time step entails integration of its derivative from the start of a simulation. Thus modeling a continuous state entails representing the operation of integration and the process of computing the state's derivative at each point in time. Simulink block diagrams use Integrator blocks to indicate integration and a chain of blocks connected to an integrator block's input to represent the method for computing the state's derivative. The chain of blocks connected to the integrator block's input is the graphical counterpart to an ordinary differential equation (ODE).

In general, excluding simple dynamic systems, analytical methods do not exist for integrating the states of real-world dynamic systems represented by ordinary differential equations. Integrating the states requires the use of numerical methods called ODE solvers. These various methods trade

computational accuracy for computational workload. Simulink comes with computerized implementations of the most common ODE integration methods and allows a user to determine which it uses to integrate states represented by Integrator blocks when simulating a system.

Computing the value of a continuous state at the current time step entails integrating its values from the start of the simulation. The accuracy of numerical integration in turn depends on the size of the intervals between time steps. In general, the smaller the time step, the more accurate the simulation. Some ODE solvers, called variable time step solvers, can automatically vary the size of the time step, based on the rate of change of the state, to achieve a specified level of accuracy over the course of a simulation. Simulink allows the user to specify the size of the time step in the case of fixed-step solvers or allow the solver to determine the step size in the case of variable-step solvers. To minimize the computation workload, the variable-step solver chooses the largest step size consistent with achieving an overall level of precision specified by the user for the most rapidly changing model state. This ensures that all model states are computed to the accuracy specified by the user.

## Discrete States

Computing a discrete state requires knowing the relationship between its value at the current time step and its value at the previous time step. Simulink refers to this relationship as the state's update function. A discrete state depends not only on its value at the previous time step but also on the values of a model's inputs. Modeling a discrete state thus entails modeling the state's dependency on the systems' inputs at the previous time step. Simulink block diagrams use specific types of blocks, called discrete blocks, to specify update functions and chains of blocks connected to the inputs of discrete blocks to model the dependency of a system's discrete states on its inputs.

As with continuous states, discrete states set a constraint on the simulation time step size. Specifically, the step size must ensure that all the sample times of the model's states are hit. Simulink assigns this task to a component of the Simulink system called a discrete solver. Simulink provides two discrete solvers: a fixed-step discrete solver and a variable-step discrete solver. The fixed-step discrete solver determines a fixed step size that hits all the sample times of all the model's discrete states, regardless of whether the states actually change value at the sample time hits. By contrast, the variable-step

discrete solver varies the step size to ensure that sample time hits occur only at times when the states change value.

## Modeling Hybrid Systems

A hybrid system is a system that has both discrete and continuous states. Strictly speaking, Simulink treats any model that has both continuous and discrete sample times as a hybrid model, presuming that the model has both continuous and discrete states. Solving such a model entails choosing a step size that satisfies both the precision constraint on the continuous state integration and the sample time hit constraint on the discrete states. Simulink meets this requirement by passing the next sample time hit, as determined by the discrete solver, as an additional constraint on the continuous solver. The continuous solver must choose a step size that advances the simulation up to but not beyond the time of the next sample time hit. The continuous solver can take a time step short of the next sample time hit to meet its accuracy constraint but it cannot take a step beyond the next sample time hit even if its accuracy constraint allows it to.

## Block Parameters

Key properties of many standard blocks are parameterized. For example, the Constant value of the Simulink Constant block is a parameter. Each parameterized block has a block dialog that lets you set the values of the parameters. You can use MATLAB expressions to specify parameter values. Simulink evaluates the expressions before running a simulation. You can change the values of parameters during a simulation. This allows you to determine interactively the most suitable value for a parameter.

A parameterized block effectively represents a family of similar blocks. For example, when creating a model, you can set the Constant value parameter of each instance of the Constant block separately so that each instance behaves differently. Because it allows each standard block to represent a family of blocks, block parameterization greatly increases the modeling power of the standard Simulink libraries.

## Tunable Parameters

Many block parameters are tunable. A *tunable parameter* is a parameter whose value can be changed without recompiling the model (see “Model

Compilation” on page 2-14 for more information on compiling a Simulink model). For example, the gain parameter of the Gain block is tunable. You can alter the block’s gain while a simulation is running. If a parameter is not tunable and the simulation is running, Simulink disables the dialog box control that sets the parameter.

---

**Note** Simulink does not allow you to change the values of source block parameters through either a dialog box or the Model Explorer while a simulation is running. Opening the dialog box of a source block with tunable parameters causes a running simulation to pause. While the simulation is paused, you can edit the parameter values displayed on the dialog box. However, you must close the dialog box to have the changes take effect and allow the simulation to continue.

---

It should be pointed out that parameter changes do not immediately occur, but are queued up and then applied at the start of the next time step during model execution. Returning to our example of the constant block, the function it defines is `signal(t) = ConstantValue` for all time. If we were to allow the constant value to be changed immediately, then the solution at the point in time at which the change occurred would be invalid. Thus we must queue the change for processing at the next time step.

You can use the **Inline parameters** option on the **Optimization** pane of the **Configuration Parameters** dialog box to specify that all parameters in your model are nontunable except for those that you specify. This can speed up execution of large models and enable generation of faster code from your model. See “Configuration Parameters Dialog Box” on page 7-36 for more information.

## Block Sample Times

Every Simulink block is considered to have a sample time, even continuous blocks (e.g., blocks that define continuous states, such as the Integrator block) and blocks that do not define states, such as the Gain block. Most blocks allow you to specify their sample times via a Sample Time parameter. Continuous blocks are considered to have an infinitesimal sample time called a continuous sample time. A block that does not specify its sample time is said to have an implicit sample time that it inherits from its inputs. The implicit sample

time is continuous if any of the block's inputs are continuous. Otherwise, the implicit sample time is discrete. An implicit discrete sample time is equal to the shortest input sample time if all the input sample times are integer multiples of the shortest time. Otherwise, the implicit sample time is equal to the *fundamental sample time* of the inputs, where the fundamental sample time of a set of sample times is defined as the greatest integer divisor of the set of sample times. See also “Sample Time Propagation” on page 2-37 for a description of how Simulink uses a process called sample time propagation to determine the sample times of blocks that inherit their sample times.

Simulink can optionally color code a block diagram to indicate the sample times of the blocks it contains, e.g., black (continuous), magenta (constant), yellow (hybrid), red (fastest discrete), and so on. See “Displaying Sample Time Colors” on page 4-8 for more information.

## Custom Blocks

Simulink allows you to create libraries of custom blocks that you can then use in your models. You can create a custom block either graphically or programmatically. To create a custom block graphically, you draw a block diagram representing the block's behavior, wrap this diagram in an instance of the Simulink Subsystem block, and provide the block with a parameter dialog, using the Simulink block mask facility. To create a block programmatically, you create an M-file or a MEX-file that contains the block's system functions (see “Writing S-Functions” in the online Simulink documentation). The resulting file is called an S-function. You then associate the S-function with instances of the Simulink S-Function block in your model. You can add a parameter dialog to your S-Function block by wrapping it in a Subsystem block and adding the parameter dialog to the Subsystem block.

## Systems and Subsystems

A Simulink block diagram can consist of layers. Each layer is defined by a subsystem. A subsystem is part of the overall block diagram and ideally has no impact on the meaning of the block diagram. Subsystems are provided primarily to help in the organization aspects a block diagram. Subsystems do not define a separate block diagram.

Simulink differentiates between two different types of subsystems: virtual and nonvirtual. The main difference is that nonvirtual subsystems provide the ability to control when the contents of the subsystem are evaluated.

### Flattening the Model Hierarchy

While preparing a model for execution, Simulink generates internal "systems" that are collections of block methods (equations) that are evaluated together. The semantics of time-based block diagrams doesn't require creation of these systems. Simulink creates these internal systems as a means to manage the execution of the model. Roughly speaking, there will be one system for the top-level block diagram which is referred to as the root system, and several lower-level systems derived from nonvirtual subsystems and other elements in the block diagram. You will see these systems in the Simulink Debugger. The act of creating these internal systems is often referred to as flattening the model hierarchy.

### Conditionally Executed Subsystems

You can create conditionally executed subsystems that are executed only when a transition occurs on a triggering, function-call, action, or enabling input (see "Creating Conditionally Executed Subsystems" on page 4-29). Conditionally executed subsystems are atomic, i.e., the equations that they define are evaluated as a unit.

### Atomic Subsystems

Unconditionally executed subsystems are virtual by default. You can, however, designate an unconditionally executed subsystem as atomic (see the Atomic Subsystem block for more information). This is useful if you need to ensure that the equations defined by a subsystem are evaluated as a unit.

### Signals

Simulink uses the term *signal* to refer to a time varying quantity that has values at all points in time. Simulink allows you to specify a wide range of signal attributes, including signal name, data type (e.g., 8-bit, 16-bit, or 32-bit integer), numeric type (real or complex), and dimensionality (one-dimensional or two-dimensional array). Many blocks can accept or output signals of any data or numeric type and dimensionality. Others impose restrictions on the attributes of the signals they can handle.

On the block diagram, you will find that the signals are represented with lines that have an arrowhead. The source of the signal corresponds to the block that writes to the signal during evaluation of its block methods (equations). The destinations of the signal are blocks that read the signal during the evaluation of its block methods (equations). A good analogy of the meaning of a signal is to consider a classroom. The teacher is the one responsible for writing on the white board and the students read what is written on the white board when they choose to. This is also true of Simulink signals, a reader of the signal (a block method) can choose to read the signal as frequently or infrequently as so desired.

## Block Methods

Blocks represent multiple equations. These equations are represented as block methods within Simulink. These block methods are evaluated (executed) during the execution of a block diagram. The evaluation of these block methods is performed within a simulation loop, where each cycle through the simulation loop represent evaluation of the block diagram at a given point in time.

## Method Types

Simulink assigns names to the types of functions performed by block methods. Common method types include:

- Outputs

Computes the outputs of a block given its inputs at the current time step and its states at the previous time step.

- Update

Computes the value of the block's discrete states at the current time step, given its inputs at the current time step and its discrete states at the previous time step.

- Derivatives

Computes the derivatives of the block's continuous states at the current time step, given the block's inputs and the values of the states at the previous time step.

## Method Naming Convention

Block methods perform the same types of operations in different ways for different types of blocks. The Simulink user interface and documentation uses dot notation to indicate the specific function performed by a block method:

```
BlockType.MethodType
```

For example, Simulink refers to the method that computes the outputs of a Gain block as

```
Gain.Outputs
```

The Simulink debugger takes the naming convention one step further and uses the instance name of a block to specify both the method type and the block instance on which the method is being invoked during simulation, e.g.,

```
g1.Outputs
```

## Model Methods

In addition to block methods, Simulink also provides a set of methods that compute the model's properties and its outputs. Simulink similarly invokes these methods during simulation to determine a model's properties and its outputs. The model methods generally perform their tasks by invoking block methods of the same type. For example, the model Outputs method invokes the Outputs methods of the blocks that it contains in the order specified by the model to compute its outputs. The model Derivatives method similarly invokes the Derivatives methods of the blocks that it contains to determine the derivatives of its states.

## Simulating Dynamic Systems

Simulating a dynamic system refers to the process of computing a system's states and outputs over a span of time, using information provided by the system's model. Simulink simulates a system when you choose **Start** from the Model Editor's **Simulation** menu, with the system's model open.

A Simulink component called the Simulink Engine responds to a Start command, performing the following steps.

### Model Compilation

First, the Simulink engine invokes the model compiler. The model compiler converts the model to an executable form, a process called compilation. In particular, the compiler

- Evaluates the model's block parameter expressions to determine their values.
- Determines signal attributes, e.g., name, data type, numeric type, and dimensionality, not explicitly specified by the model and checks that each block can accept the signals connected to its inputs.
- Simulink uses a process called attribute propagation to determine unspecified attributes. This process entails propagating the attributes of a source signal to the inputs of the blocks that it drives.
- Performs block reduction optimizations.
- Flattens the model hierarchy by replacing virtual subsystems with the blocks that they contain (see “Solvers” on page 2-17).
- Determines the block sorted order (see “Controlling and Displaying the Sorted Order” in the online Simulink documentation for more information).
- Determines the sample times of all blocks in the model whose sample times you did not explicitly specify (see “Sample Time Propagation” on page 2-37).

### Link Phase

In this phase, the Simulink Engine allocates memory needed for working areas (signals, states, and run-time parameters) for execution of the block diagram. It also allocates and initializes memory for data structures that

store run-time information for each block. For built-in blocks, the principal run-time data structure for a block is called the SimBlock. It stores pointers to a block's input and output buffers and state and work vectors.

### Method Execution Lists

In the Link phase, the Simulink engine also creates method execution lists. These lists list the most efficient order in which to invoke a model's block methods to compute its outputs. Simulink uses the block sorted order lists generated during the model compilation phase to construct the method execution lists.

### Block Priorities

Simulink allows you to assign update priorities to blocks (see “Assigning Block Priorities” in the online Simulink documentation). Simulink executes the output methods of higher priority blocks before those of lower priority blocks. Simulink honors the priorities only if they are consistent with its block sorting rules.

## Simulation Loop Phase

The simulation now enters the simulation loop phase. In this phase, the Simulink engine successively computes the states and outputs of the system at intervals from the simulation start time to the finish time, using information provided by the model. The successive time points at which the states and outputs are computed are called time steps. The length of time between steps is called the step size. The step size depends on the type of solver (see “Solvers” on page 2-17) used to compute the system’s continuous states, the system’s fundamental sample time (see “Modeling and Simulating Discrete Systems” on page 2-31), and whether the system’s continuous states have discontinuities (see “Zero-Crossing Detection” on page 2-19).

The Simulation Loop phase has two subphases: the Loop Initialization phase and the Loop Iteration phase. The initialization phase occurs once, at the start of the loop. The iteration phase is repeated once per time step from the simulation start time to the simulation stop time.

At the start of the simulation, the model specifies the initial states and outputs of the system to be simulated. At each step, Simulink computes new

values for the system's inputs, states, and outputs and updates the model to reflect the computed values. At the end of the simulation, the model reflects the final values of the system's inputs, states, and outputs. Simulink provides data display and logging blocks. You can display and/or log intermediate results by including these blocks in your model.

## Loop Iteration

At each time step, the Simulink Engine

- 1 Computes the model's outputs.

The Simulink Engine initiates this step by invoking the Simulink model Outputs method. The model Outputs method in turn invokes the model system Outputs method, which invokes the Outputs methods of the blocks that the model contains in the order specified by the Outputs method execution lists generated in the Link phase of the simulation (see "Solvers" on page 2-17).

The system Outputs method passes the following arguments to each block Outputs method: a pointer to the block's data structure and to its SimBlock structure. The SimBlock data structures point to information that the Outputs method needs to compute the block's outputs, including the location of its input buffers and its output buffers.

- 2 Computes the model's states.

The Simulink Engine computes a model's states by invoking a solver. Which solver it invokes depends on whether the model has no states, only discrete states, only continuous states, or both continuous and discrete states.

If the model has only discrete states, the Simulink Engine invokes the discrete solver selected by the user. The solver computes the size of the time step needed to hit the model's sample times. It then invokes the Update method of the model. The model Update method invokes the Update method of its system, which invokes the Update methods of each of the blocks that the system contains in the order specified by the Update method lists generated in the Link phase.

If the model has only continuous states, the Simulink Engine invokes the continuous solver specified by the model. Depending on the solver, the

solver either in turn calls the Derivatives method of the model once or enters a subcycle of minor time steps where the solver repeatedly calls the model's Outputs methods and Derivatives methods to compute the model's outputs and derivatives at successive intervals within the major time step. This is done to increase the accuracy of the state computation. The model Outputs method and Derivatives methods in turn invoke their corresponding system methods, which invoke the block Outputs and Derivatives in the order specified by the Outputs and Derivatives methods execution lists generated in the Link phase.

- 3 Optionally checks for discontinuities in the continuous states of blocks.

Simulink uses a technique called zero-crossing detection to detect discontinuities in continuous states. See “Zero-Crossing Detection” on page 2-19 for more information.

- 4 Computes the time for the next time step.

Simulink repeats steps 1 through 4 until the simulation stop time is reached.

## Solvers

Simulink simulates a dynamic system by computing its states at successive time steps over a specified time span, using information provided by the model. The process of computing the successive states of a system from its model is known as solving the model. No single method of solving a model suffices for all systems. Accordingly, Simulink provides a set of programs, known as *solvers*, that each embody a particular approach to solving a model. The **Configuration Parameters** dialog box allows you to choose the solver most suitable for your model (see “Choosing a Solver Type” on page 7-7).

### Fixed-Step Solvers Versus Variable-Step Solvers

Simulink solvers fall into two basic categories: fixed-step and variable-step.

*Fixed-step solvers* solve the model at regular time intervals from the beginning to the end of the simulation. The size of the interval is known as the step size. You can specify the step size or let the solver choose the step size. Generally, decreasing the step size increases the accuracy of the results while increasing the time required to simulate the system.

*Variable-step solvers* vary the step size during the simulation, reducing the step size to increase accuracy when a model's states are changing rapidly and increasing the step size to avoid taking unnecessary steps when the model's states are changing slowly. Computing the step size adds to the computational overhead at each step but can reduce the total number of steps, and hence simulation time, required to maintain a specified level of accuracy for models with rapidly changing or piecewise continuous states.

### **Continuous Versus Discrete Solvers**

Simulink provides both continuous and discrete solvers.

*Continuous solvers* use numerical integration to compute a model's continuous states at the current time step from the states at previous time steps and the state derivatives. Continuous solvers rely on the model's blocks to compute the values of the model's discrete states at each time step.

Mathematicians have developed a wide variety of numerical integration techniques for solving the ordinary differential equations (ODEs) that represent the continuous states of dynamic systems. Simulink provides an extensive set of fixed-step and variable-step continuous solvers, each implementing a specific ODE solution method (see “Choosing a Solver Type” on page 7-7).

*Discrete solvers* exist primarily to solve purely discrete models. They compute the next simulation time step for a model and nothing else. They do not compute continuous states and they rely on the model's blocks to update the model's discrete states.

---

**Note** You can use a continuous solver, but not a discrete solver, to solve a model that contains both continuous and discrete states. This is because a discrete solver does not handle continuous states. If you select a discrete solver for a continuous model, Simulink disregards your selection and uses a continuous solver instead when solving the model.

---

Simulink provides two discrete solvers, a fixed-step discrete solver and a variable-step discrete solver. The fixed-step solver by default chooses a step size and hence simulation rate fast enough to track state changes in the

fastest block in your model. The variable-step solver adjusts the simulation step size to keep pace with the actual rate of discrete state changes in your model. This can avoid unnecessary steps and hence shorten simulation time for multirate models (see “Determining Step Size for Discrete Systems” on page 2-36 for more information).

### Minor Time Steps

Some continuous solvers subdivide the simulation time span into major and minor time steps, where a minor time step represents a subdivision of the major time step. The solver produces a result at each major time step. It uses results at the minor time steps to improve the accuracy of the result at the major time step.

### Zero-Crossing Detection

When simulating a dynamic system, Simulink checks for discontinuities in the system’s state variables at each time step, using a technique known as zero-crossing detection. If Simulink detects a discontinuity within the current time step, it determines the precise time at which the discontinuity occurs and takes additional time steps before and after the discontinuity. This section explains why zero-crossing detection is important and how it works.

Discontinuities in state variables often coincide with significant events in the evolution of a dynamic system. For example, the instant when a bouncing ball hits the floor coincides with a discontinuity in its velocity. Because discontinuities often indicate a significant change in a dynamic system, it is important to simulate points of discontinuity precisely. Otherwise, a simulation could lead to false conclusions about the behavior of the system under investigation. Consider, for example, a simulation of a bouncing ball. If the point at which the ball hits the floor occurs between simulation steps, the simulated ball appears to reverse position in midair. This might lead an investigator to false conclusions about the physics of the bouncing ball.

To avoid such misleading conclusions, it is important that simulation steps occur at points of discontinuity. A simulator that relies purely on solvers to determine simulation times cannot efficiently meet this requirement. Consider, for example, a fixed-step solver. A fixed-step solver computes the values of state variables at integral multiples of a fixed step size. However, there is no guarantee that a point of discontinuity will occur at an integral

multiple of the step size. You could reduce the step size to increase the probability of hitting a discontinuity, but this would greatly increase the execution time.

A variable-step solver appears to offer a solution. A variable-step solver adjusts the step size dynamically, increasing the step size when a variable is changing slowly and decreasing the step size when the variable changes rapidly. Around a discontinuity, a variable changes extremely rapidly. Thus, in theory, a variable-step solver should be able to hit a discontinuity precisely. The problem is that to locate a discontinuity accurately, a variable-step solver must again take many small steps, greatly slowing down the simulation.

### **How Zero-Crossing Detection Works**

Simulink uses a technique known as zero-crossing detection to address this problem. With this technique, a block can register a set of zero-crossing variables with Simulink, each of which is a function of a state variable that can have a discontinuity. The zero-crossing function passes through zero from a positive or negative value when the corresponding discontinuity occurs. At the end of each simulation step, Simulink asks each block that has registered zero-crossing variables to update the variables. Simulink then checks whether any variable has changed sign since the last step. Such a change indicates that a discontinuity occurred in the current time step.

If any zero crossings are detected, Simulink interpolates between the previous and current values of each variable that changed sign to estimate the times of the zero crossings (e.g., discontinuities). Simulink then steps up to and over each zero crossing in turn. In this way, Simulink avoids simulating exactly at the discontinuity, where the value of the state variable might be undefined.

Zero-crossing detection enables Simulink to simulate discontinuities accurately without resorting to excessively small step sizes. Many Simulink blocks support zero-crossing detection. The result is fast and accurate simulation of all systems, including systems with discontinuities.

## Implementation Details

An example of a Simulink block that uses zero crossings is the Saturation block. Zero crossings detect these state events in the Saturation block:

- The input signal reaches the upper limit.
- The input signal leaves the upper limit.
- The input signal reaches the lower limit.
- The input signal leaves the lower limit.

Simulink blocks that define their own state events are considered to have *intrinsic zero crossings*. If you need explicit notification of a zero-crossing event, use the Hit Crossing block. See “Blocks with Zero Crossings” on page 2-23 for a list of blocks that incorporate zero crossings.

The detection of a state event depends on the construction of an internal zero-crossing signal. This signal is not accessible by the block diagram. For the Saturation block, the signal that is used to detect zero crossings for the upper limit is  $zcSignal = \text{UpperLimit} - u$ , where  $u$  is the input signal.

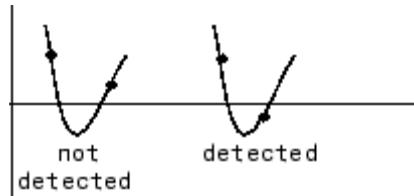
Zero-crossing signals have a direction attribute, which can have these values:

- *rising* - A zero crossing occurs when a signal rises to or through zero, or when a signal leaves zero and becomes positive.
- *falling* - A zero crossing occurs when a signal falls to or through zero, or when a signal leaves zero and becomes negative.
- *either* - A zero crossing occurs if either a rising or falling condition occurs.

For the Saturation block’s upper limit, the direction of the zero crossing is *either*. This enables the entering and leaving saturation events to be detected using the same zero-crossing signal.

If the error tolerances are too large, it is possible for Simulink to fail to detect a zero crossing. For example, if a zero crossing occurs within a time step, but the values at the beginning and end of the step do not indicate a sign change, the solver steps over the crossing without detecting it.

The following figure shows a signal that crosses zero. In the first instance, the integrator steps over the event. In the second, the solver detects the event.



If you suspect this is happening, tighten the error tolerances to ensure that the solver takes small enough steps. For more information, see “Maximum Order” in the online Simulink documentation.

---

**Note** Using the `Refine output` option (see “Output options” in the online Simulink documentation) does not help locate the missed zero crossings. You should alter the maximum step size or output times.

---

### Caveat

It is possible to create models that exhibit high-frequency fluctuations about a discontinuity (chattering). Such systems typically are not physically realizable; a massless spring, for example. Because chattering causes repeated detection of zero crossings, the step sizes of the simulation become very small, essentially halting the simulation.

If you suspect that this behavior applies to your model, you can use the **Zero crossing control** option on the **Solver** pane of the **Configuration Parameters** dialog box (see “Zero crossing control” in the online Simulink documentation) to disable zero-crossing detection. Although disabling zero-crossing detection can alleviate the symptoms of this problem, you no longer benefit from the increased accuracy that zero-crossing detection provides. A better solution is to try to identify the source of the underlying problem in the model.

## Blocks with Zero Crossings

The following table lists blocks that use zero crossings and explains how the blocks use the zero crossings:

<b>Block</b>	<b>Description of Zero Crossing</b>
Abs	One: to detect when the input signal crosses zero in either the rising or falling direction.
Backlash	Two: one to detect when the upper threshold is engaged, and one to detect when the lower threshold is engaged.
Dead Zone	Two: one to detect when the dead zone is entered (the input signal minus the lower limit), and one to detect when the dead zone is exited (the input signal minus the upper limit).
From Workspace	One: to detect when the input signal has a discontinuity in either the rising or falling direction
Hit Crossing	One: to detect when the input crosses the threshold.
If	One: to detect when the If condition is met.
Integrator	If the reset port is present, to detect when a reset occurs. If the output is limited, there are three zero crossings: one to detect when the upper saturation limit is reached, one to detect when the lower saturation limit is reached, and one to detect when saturation is left.
MinMax	One: for each element of the output vector, to detect when an input signal is the new minimum or maximum.
Relay	One: if the relay is off, to detect the switch on point. If the relay is on, to detect the switch off point.
Relational Operator	One: to detect when the output changes.
Saturation	Two: one to detect when the upper limit is reached or left, and one to detect when the lower limit is reached or left.
Sign	One: to detect when the input crosses through zero.

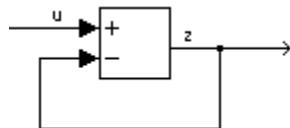
Block	Description of Zero Crossing
Signal Builder	One: to detect when the input signal has a discontinuity in either the rising or falling direction
Step	One: to detect the step time.
Subsystem	For conditionally executed subsystems: one for the enable port if present, and one for the trigger port, if present.
Switch	One: to detect when the switch condition occurs.
Switch Case	One: to detect when the case condition is met.

## Algebraic Loops

Some Simulink blocks have input ports with *direct feedthrough*. This means that the output of these blocks cannot be computed without knowing the values of the signals entering the blocks at these input ports. Some examples of blocks with direct feedthrough inputs are as follows:

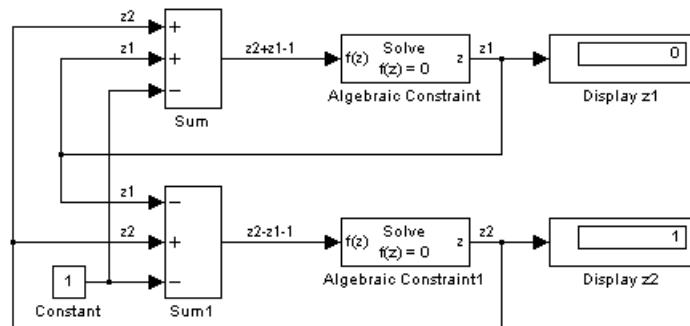
- The Math Function block
- The Gain block
- The Integrator block's initial condition ports
- The Product block
- The State-Space block when there is a nonzero D matrix
- The Sum block
- The Transfer Fcn block when the numerator and denominator are of the same order
- The Zero-Pole block when there are as many zeros as poles

An *algebraic loop* generally occurs when an input port with direct feedthrough is driven by the output of the same block, either directly, or by a feedback path through other blocks with direct feedthrough. An example of an algebraic loop is this simple scalar loop.



Mathematically, this loop implies that the output of the Sum block is an algebraic state  $z$  constrained to equal the first input  $u$  minus  $z$  (i.e.  $z = u - z$ ). The solution of this simple loop is  $z = u/2$ , but most algebraic loops cannot be solved by inspection.

It is easy to create vector algebraic loops with multiple algebraic state variables  $z1$ ,  $z2$ , etc., as shown in this model.



The Algebraic Constraint block is a convenient way to model algebraic equations and specify initial guesses. The Algebraic Constraint block constrains its input signal  $F(z)$  to zero and outputs an algebraic state  $z$ . This block outputs the value necessary to produce a zero at the input. The output must affect the input through some feedback path. You can provide an initial guess of the algebraic state value in the block's dialog box to improve algebraic loop solver efficiency.

A scalar algebraic loop represents a scalar algebraic equation or constraint of the form  $F(z) = 0$ , where  $z$  is the output of one of the blocks in the loop and the function  $F$  consists of the feedback path through the other blocks in the loop to the input of the block. In the simple one-block example shown on the previous page,  $F(z) = z - (u - z)$ . In the vector loop example shown above, the equations are

$$z2 + z1 - 1 = 0$$

$$z2 - z1 - 1 = 0$$

Algebraic loops arise when a model includes an algebraic constraint  $F(z) = 0$ . This constraint might arise as a consequence of the physical interconnectivity of the system you are modeling, or it might arise because you are specifically trying to model a differential/algebraic system (DAE).

When a model contains an algebraic loop, Simulink calls a loop solving routine at each time step. The loop solver performs iterations to determine the solution to the problem (if it can). As a result, models with algebraic loops run slower than models without them.

To solve  $F(z) = 0$ , the Simulink loop solver uses Newton's method with weak line search and rank-one updates to a Jacobian matrix of partial derivatives. Although the method is robust, it is possible to create loops for which the loop solver will not converge without a good initial guess for the algebraic states  $z$ . You can specify an initial guess for a line in an algebraic loop by placing an IC block (which is normally used to specify an initial condition for a signal) on that line. As shown above, another way to specify an initial guess for a line in an algebraic loop is to use an Algebraic Constraint block.

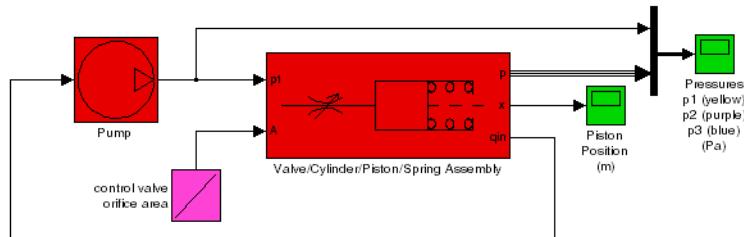
Whenever possible, use an IC block or an Algebraic Constraint block to specify an initial guess for the algebraic state variables in a loop.

### Highlighting Algebraic Loops

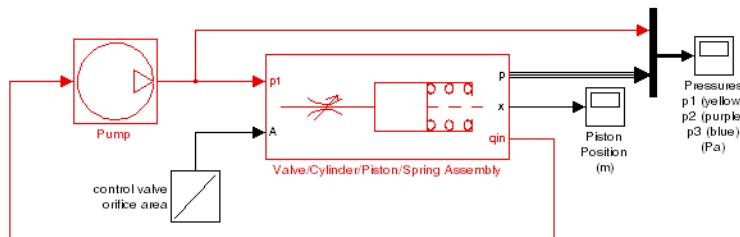
You can cause Simulink to highlight algebraic loops when you update, simulate, or debug a model. Use the `ashow` command to highlight algebraic loops when debugging a model.

To cause Simulink to highlight algebraic loops that it detects when updating or simulating a model, set the Algebraic loop diagnostic on the **Diagnostics** pane of the **Configuration Parameters** dialog box to Error (see “Configuration Parameters Dialog Box” on page 7-36 for more information). This causes Simulink to display an error dialog (the Diagnostics Viewer) and recolor portions of the diagram that represent the algebraic loops that it detects. Simulink uses red to color the blocks and lines that constitute the loops. Closing the error dialog restores the diagram to its original colors.

For example, the following figure shows the block diagram of the `hydcy1` demo model in its original colors.



The following figure shows the diagram after updating when the **Algebraic loop diagnostic** is set to **Error**.



In this example, Simulink has colored the algebraic loop red, making it stand out from the rest of the diagram.

## Eliminating Algebraic Loops

Simulink can eliminate some algebraic loops that include any of the following types of blocks:

- Atomic Subsystem
- Enabled Subsystem
- Model

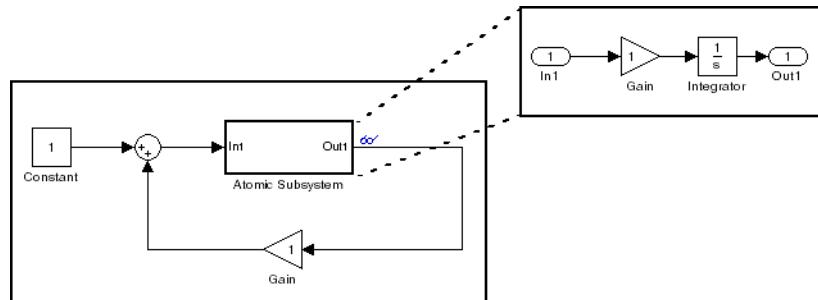
To enable automatic algebraic loop elimination for a loop involving a particular instance of an Atomic Subsystem or Enabled Subsystem block, select the **Minimize algebraic loop occurrences** parameter on the block's

parameters dialog box. To enable algebraic loop elimination for a loop involving a Model block, check the **Minimize algebraic loop occurrences** parameter on the **Model Referencing Pane** of the **Configuration Parameters** dialog box (see “The Model Referencing Pane” in the online Simulink documentation) of the model referenced by the Model block. If a loop includes more than one instance of these blocks, you should enable algebraic loop elimination for all of them, including nested blocks.

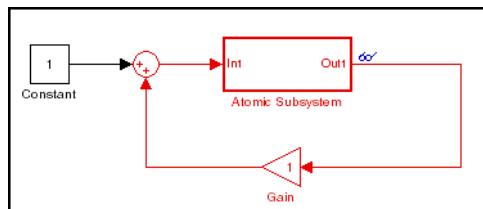
The Simulink **Minimize algebraic loop** solver diagnostic allows you to specify the action Simulink should take, for example, display a warning message, if it is unable to eliminate an algebraic loop involving a block for which algebraic loop elimination is enabled. See “The Diagnostics Pane” in the online Simulink documentation for more information.

Algebraic loop minimization is off by default because it is incompatible with conditional input branch optimization in Simulink (see “The Optimization Pane” in the online Simulink documentation) and with single output/update function optimization in Real-Time Workshop®. If you need these optimizations for an atomic or enabled subsystem or referenced model involved in an algebraic loop, you must eliminate the algebraic loop yourself.

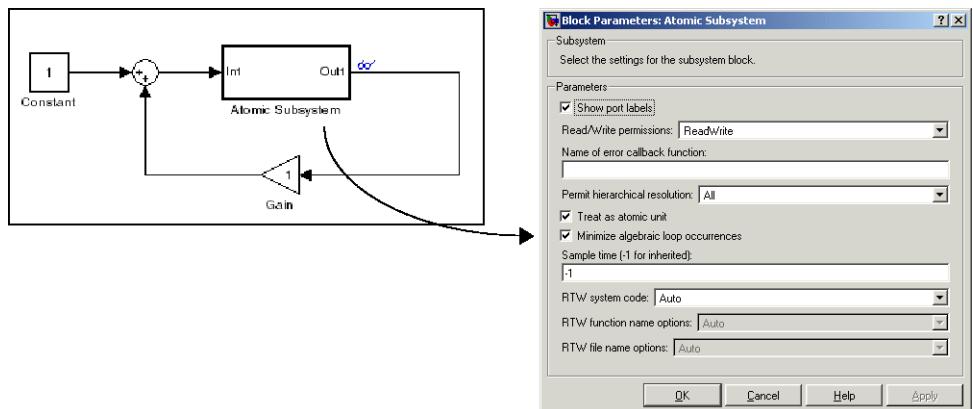
As an example of the ability of Simulink to eliminate algebraic loops, consider the following model.



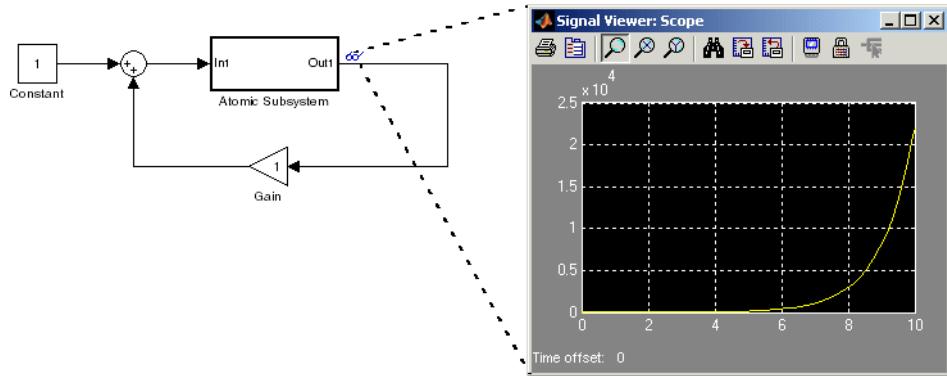
Simulating this model with the solver's Algebraic Loop diagnostic set to error (see "The Diagnostics Pane" in the online Simulink documentation for more information) reveals that this model contains an algebraic loop involving its atomic subsystem.



Checking the atomic subsystem's **Minimize algebraic loop occurrences** parameter causes Simulink to eliminate the algebraic loop from the compiled version of the model.

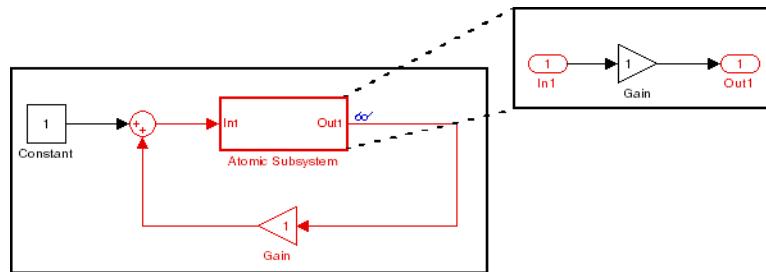


As a result, the model now simulates without error.



Note that Simulink is able to eliminate the algebraic loop involving this model's atomic subsystem because the atomic subsystem contains a block with a port that does not have direct feed through, i.e., the Integrator block.

If you remove the Integrator block from the atomic subsystem, Simulink is unable to eliminate the algebraic loop. Hence, attempting to simulate the model results in an error.



# Modeling and Simulating Discrete Systems

Simulink has the ability to simulate discrete (sampled data) systems, including systems whose components operate at different rates (*multirate systems*) and systems that mix discrete and continuous components (*hybrid systems*). This capability stems from two key Simulink features:

- SampleTime block parameter

Some Simulink blocks have a SampleTime parameter that you can use to specify the block's sample time, i.e., the rate at which it executes during simulation. All blocks have either an explicit or implicit sample time parameter. Continuous blocks are examples of blocks that have an implicit (continuous) sample time. It is possible for a block to have multiple sample times as provided with blocksets such as the Signal Processing Blockset or created by a user using the S-Function block.

- Sample-time inheritance

Most standard Simulink blocks can inherit their sample time from the blocks connected to their inputs. Exceptions include blocks in the Continuous library and blocks that do not have inputs (e.g., blocks from the Sources library). In some cases, source blocks can inherit the sample time of the block connected to their output.

The ability to specify sample times on a block-by-block basis, either directly through the SampleTime parameter or indirectly through inheritance, enables you to model systems containing discrete components operating at different rates and hybrid systems containing discrete and continuous components.

## Specifying Sample Time

Simulink allows you to specify the sample time of any block that has a SampleTime parameter. You can use the block's parameter dialog box to set this parameter. You do this by entering the sample time in the **Sample time** field on the dialog box. You can enter either the sample time alone or a vector whose first element is the sample time and whose second element is an offset:  $[T_s, T_o]$ . Various values of the sample time and offset have special meanings.

The following table summarizes valid values for this parameter and how Simulink interprets them to determine a block's sample time.

Sample Time	Usage
$[T_s, T_o]$ $0 > T_s < T_{sim}$ $ T_o  < T_p$	<p>Specifies that updates occur at simulation times</p> $t_n = n * T_s +  T_o $ <p>where <math>n</math> is an integer in the range <math>1 \dots T_{sim}/T_s</math> and <math>T_{sim}</math> is the length of the simulation. Blocks that have a sample time greater than 0 are said to have a <i>discrete sample time</i>.</p> <p>The offset allows you to specify that Simulink update the block later in the sample interval than other blocks operating at the same rate.</p>
$[0, 0], 0$	<p>Specifies that updates occur at every major and minor time step. A block that has a sample time of 0 is said to have a <i>continuous sample time</i>.</p>
$[0, 1]$	<p>Specifies that updates occur only at major time steps, skipping minor time steps (see “Minor Time Steps” on page 2-19). This setting avoids unnecessary computations for blocks whose sample time cannot change between major time steps. The sample time of a block that executes only at major time steps is said to be <i>fixed in minor time step</i>.</p>
$[-1, 0], -1$	<p>If the block is not in a triggered subsystem, this setting specifies that the block inherits its sample time from the block connected to its input (inheritance) or, in some cases, from the block connected to its output (back inheritance). If the block is in a triggered subsystem, you must set the SampleTime parameter to this setting.</p> <p>Note that specifying sample-time inheritance for a source block can cause Simulink to assign an inappropriate sample time to the block if the source drives more than one block. For this reason, you should avoid specifying sample-time inheritance for source blocks. If you do, Simulink displays a warning message when you update or simulate the model.</p>

Sample Time	Usage
[-2, $T_{vo}$ ]	Specifies that a block has a <i>variable sample time</i> , i.e., computes its output only at times when the output changes. Every block with variable sample time has a unique $T_{vo}$ determined by Simulink. The only built-in Simulink block that can have variable sample time is the Pulse Generator block. The sample time color (see “Displaying Sample Time Colors” on page 4-8) for variable sample time is yellow.
inf	<p>The meaning of this sample time depends on whether the active model configuration’s inline parameters optimization (see “Inline parameters” in the online Simulink documentation) is enabled.</p> <p>If the inline parameters optimization is enabled, inf signifies that the block’s output can never change (see “Constant Sample Time” on page 2-39). This speeds up simulation and the generated code by eliminating the need to recompute the block’s output at each time step. If the inline parameters optimization is disabled or the block with inf sample time drives an output port of a conditionally executed subsystem, Simulink treats inf as -1, i.e., as inherited sample time. This allows you to tune the block’s parameters during simulation.</p>

## Changing a Block’s Sample Time

You cannot change the SampleTime parameter of a block while a simulation is running. If you want to change a block’s sample time, you must stop and restart the simulation for the change to take effect.

## Compiled Sample Time

During the compilation phase of a simulation, Simulink determines the sample time of the block from its SampleTime parameter (if it has a SampleTime parameter), sample-time inheritance, or block type (Continuous blocks always have a continuous sample time). It is this compiled sample time that determines the sample rate of a block during simulation. You can

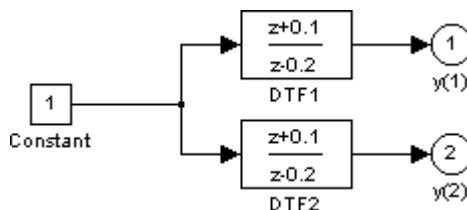
determine the compiled sample time of any block in a model by first updating the model and then getting the block's CompiledSampleTime parameter, using the `get_param` command.

## Purely Discrete Systems

Purely discrete systems can be simulated using any of the solvers; there is no difference in the solutions. To generate output points only at the sample hits, choose one of the discrete solvers.

## Multirate Systems

Multirate systems contain blocks that are sampled at different rates. These systems can be modeled with discrete blocks or with both discrete and continuous blocks. For example, consider this simple multirate discrete model.

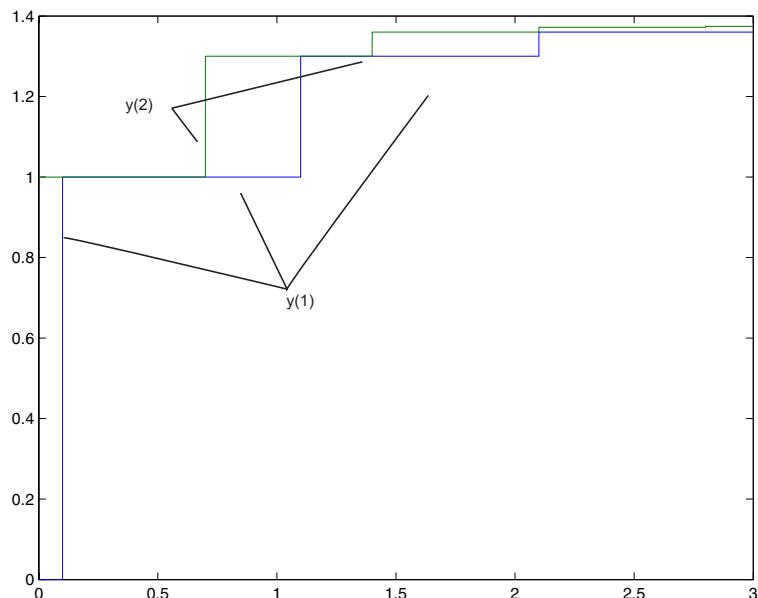


For this example the DTF1 Discrete Transfer Fcn block's **Sample time** is set to [1 0.1], which gives it an offset of 0.1. The DTF2 Discrete Transfer Fcn block's **Sample time** is set to 0.7, with no offset.

Starting the simulation and plotting the outputs using the `stairs` function

```
[t,x,y] = sim('multirate', 3);  
stairs(t,y)
```

produces this plot



See “Running a Simulation Programmatically” in the online Simulink documentation for information on the `sim` command.

For the DTF1 block, which has an offset of 0.1, there is no output until  $t = 0.1$ . Because the initial conditions of the transfer functions are zero, the output of DTF1,  $y(1)$ , is zero before this time.

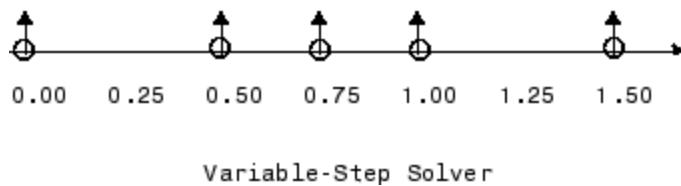
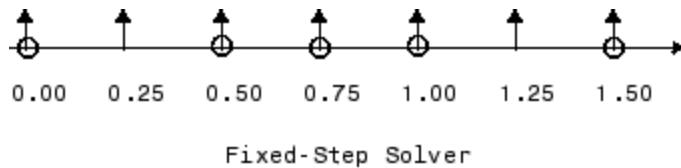
## Determining Step Size for Discrete Systems

Simulating a discrete system requires that the simulator take a simulation step at every *sample time hit*, that is, at integer multiples of the system's shortest sample time. Otherwise, the simulator might miss key transitions in the system's states. Simulink avoids this by choosing a simulation step size to ensure that steps coincide with sample time hits. The step size that Simulink chooses depends on the system's fundamental sample time and the type of solver used to simulate the system.

The *fundamental sample time* of a discrete system is the greatest integer divisor of the system's actual sample times. For example, suppose that a system has sample times of 0.25 and 0.5 second. The fundamental sample time in this case is 0.25 second. Suppose, instead, the sample times are 0.5 and 0.75 second. In this case, the fundamental sample time is again 0.25 second.

You can direct Simulink to use either a fixed-step or a variable-step discrete solver to solve a discrete system. A fixed-step solver sets the simulation step size equal to the discrete system's fundamental sample time. A variable-step solver varies the step size to equal the distance between actual sample time hits.

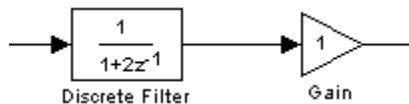
The following diagram illustrates the difference between a fixed-step and a variable-size solver.



In the diagram, arrows indicate simulation steps and circles represent sample time hits. As the diagram illustrates, a variable-step solver requires fewer simulation steps to simulate a system, if the fundamental sample time is less than any of the actual sample times of the system being simulated. On the other hand, a fixed-step solver requires less memory to implement and is faster if one of the system's sample times is fundamental. This can be an advantage in applications that entail generating code from a Simulink model (using Real-Time Workshop®).

## Sample Time Propagation

When updating a model's diagram, for example, at the beginning of a simulation, Simulink uses a process called sample time propagation to determine the sample times of blocks that inherit their sample times. The figure below illustrates a Discrete Filter block with a sample time of  $T_s$  driving a Gain block.



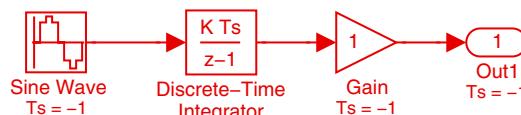
Because the Gain block's output is simply the input multiplied by a constant, its output changes at the same rate as the filter. In other words, the Gain block has an effective sample rate equal to that of the filter's sample rate. This is the fundamental mechanism behind sample time propagation in Simulink.

Simulink assigns an inherited sample time to a block based on the sample times of the blocks connected to its inputs, using the following rules.

- If all the inputs have the same sample time, Simulink assigns that sample time to the block.
- If the inputs have different sample times and if all the input sample times are integer multiples of the fastest input sample time, the block is assigned the sample time of the fastest input.
- If the inputs have different sample times and some of the input sample times are not integer multiples of the fastest sample time and a variable-step solver is being used, the block is assigned continuous sample time.

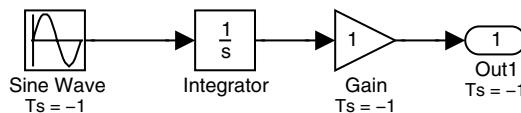
- If the inputs have different sample times and some of the input sample times are not integer multiples of the fastest sample time and a fixed-step solver is being used, and the greatest common divisor of the sample times (the fundamental sample time) can be computed, the block is assigned the fundamental sample time; otherwise, in this case, the block is assigned continuous sample time.

Under some circumstances, Simulink also back propagates sample times to source blocks if it can do so without affecting the output of a simulation. For instance, in the model below, Simulink recognizes that the Sine Wave block is driving a Discrete-Time Integrator block, so it assigns the Sine Wave block and the Gain block the same sample time as the Discrete-Time Integrator block.



You can verify this by selecting **Sample Time Colors** from the Simulink **Format** menu and noting that all blocks are colored red. Because the Discrete-Time Integrator block only looks at its input at its sample times, this change does not affect the outcome of the simulation but does result in a performance improvement.

Replacing the Discrete-Time Integrator block with a continuous Integrator block, as shown below, and recoloring the model by choosing **Update diagram** from the **Edit** menu cause the Sine Wave and Gain blocks to change to continuous blocks, as indicated by their being colored black.



## Constant Sample Time

A block whose output cannot change from its initial value during a simulation is said to have *constant sample time*. A block has constant sample time if it satisfies both of the following conditions:

- All of its parameters are nontunable, either because they are inherently nontunable or because they have been inlined (see “Inline parameters” in the online Simulink documentation).
- The block’s sample time has been declared infinite (`inf`) or its sample time is declared to be inherited and it inherits a constant sample time from another block to which it is connected.

When Simulink updates a model, for example, at the beginning of a simulation, Simulink determines which blocks, if any, have constant sample time, and computes the initial values of the output ports. During the simulation, Simulink uses the initial values whenever the outputs of blocks with constant sample time are required, thus avoiding unnecessary computations.

You can determine which blocks have constant sample time by selecting **Sample Time Colors** from the **Format** menu and updating the model. Blocks with constant sample time are colored magenta.

For example, in this model, as sample time colors show, both the Constant and Gain blocks have constant sample time.



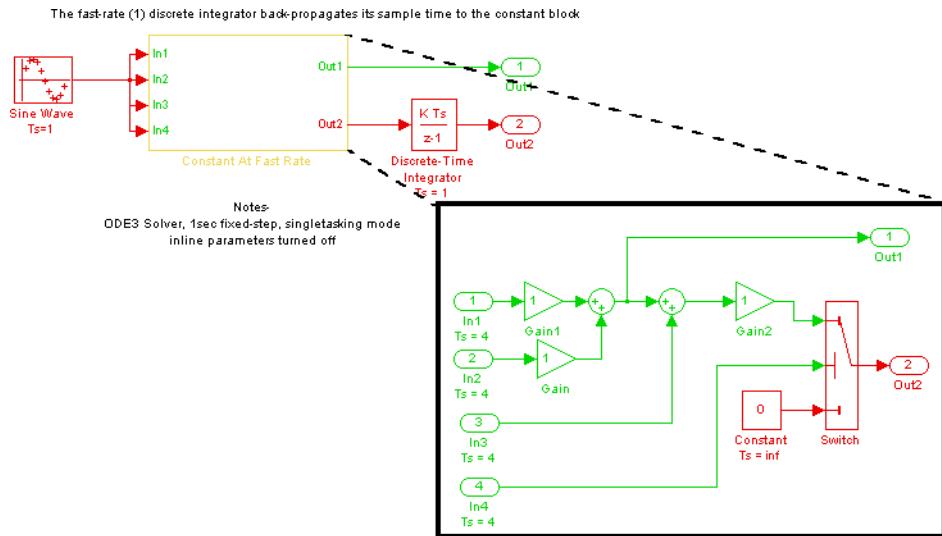
The Gain block has constant sample time because it inherits its sample time from the Constant block and all of the model’s parameters are inlined, i.e., nontunable.

**Note** The Simulink block library includes a few blocks, e.g., the S-Function, Level-2 M-File S-Function, Rate Transition, and Model block, whose ports can produce outputs at different sample rates. It is possible for some of the ports of such blocks to inherit a constant sample time. The ports with constant sample time produce output only once, at the beginning of the simulation. The other ports produce outputs at their sample rates.

### How Simulink Treats Blocks with Infinite Sample Times and Tunable Parameters

A block that has tunable parameters cannot have constant sample time even if its sample time is specified to be infinite. This is because the fact that a block has one or more tunable parameters means that you can change the values of its parameters during simulation and hence the value of its outputs. In this case, Simulink uses sample time propagation (see “Sample Time Propagation” on page 2-37) to determine the block’s actual sample time.

For example, consider the following model.



In this example, although the Constant block's sample time is specified to be infinite, it cannot have constant sample time because the inlined parameters option is off for this model and therefore the block's **Constant value** parameter is tunable. Since the Constant block's output can change during the simulation, Simulink has to determine a sample time for the block that ensures accurate simulation results. It does this by treating the Constant block's sample time as inherited and using sample time propagation to determine its sample time. The first nonvirtual block in the diagram branch to which the Constant block is connected is the Discrete-Time Integrator block. As a result, the block inherits its sample time (1 sec) via back propagation from the Discrete-Time Integrator block.

## Mixed Continuous and Discrete Systems

Mixed continuous and discrete systems are composed of both sampled and continuous blocks. Such systems can be simulated using any of the integration methods, although certain methods are more efficient and accurate than others. For most mixed continuous and discrete systems, the Runge-Kutta variable-step methods, `ode23` and `ode45`, are superior to the other methods in terms of efficiency and accuracy. Because of discontinuities associated with the sample and hold of the discrete blocks, the `ode15s` and `ode113` methods are not recommended for mixed continuous and discrete systems.



# Simulink Basics

---

The following sections explain how to perform basic Simulink tasks.

Starting Simulink (p. 3-2)	How to start Simulink.
Opening Models (p. 3-3)	How to open a Simulink model.
Model Editor (p. 3-5)	Overview of the Model Editor.
Updating a Block Diagram (p. 3-9)	How to update a diagram to reflect changes that you have made.
Saving a Model (p. 3-11)	How to save a Simulink model to disk.
Printing a Block Diagram (p. 3-15)	How to print a Simulink block diagram.
Generating a Model Report (p. 3-20)	How to generate an HTML report on a model's structure and content.
Summary of Mouse and Keyboard Actions (p. 3-23)	Lists key combinations and mouse actions that you can use to execute Simulink commands.
Ending a Simulink Session (p. 3-27)	How to end a Simulink session.

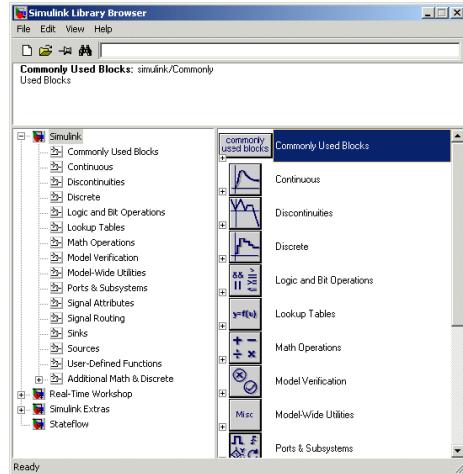
## Starting Simulink

To start Simulink, you must first start MATLAB. Consult your MATLAB documentation for more information. You can then start Simulink in two ways:



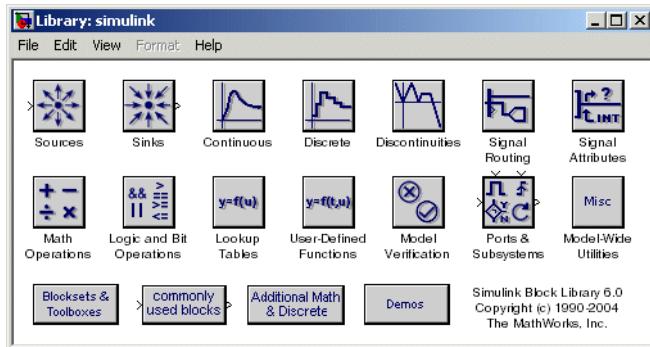
- Click the Simulink icon  on the MATLAB toolbar.
- Enter the `simulink` command at the MATLAB prompt.

On Microsoft Windows platforms, starting Simulink displays the Simulink Library Browser.



The Library Browser displays a tree-structured view of the Simulink block libraries installed on your system. You can build models by copying blocks from the Library Browser into a model window (see “Editing Blocks” on page 5-4).

On Macintosh or Linux platforms, starting Simulink displays the Simulink block library window.



The Simulink library window displays icons representing the block libraries that come with Simulink. You can create models by copying blocks from the library into a model window.

---

**Note** On Windows, you can display the Simulink library window by right-clicking the Simulink node in the Library Browser window.

---

## Opening Models

To edit an existing model diagram, either

- Click the **Open** button on the Library Browser's toolbar (Windows only) or select **Open** from the Simulink library window's **File** menu and then choose or enter the file name for the model to edit.
- Enter the name of the model (without the .mdl extension) in the MATLAB Command Window. The model must be in the current directory or on the path.

## Opening Models with Different Character Encodings

If you open a model created in a MATLAB session configured to support one character set encoding, for example, Shift\_JIS, in a MATLAB session

configured to support another character encoding, for example, US\_ASCII, Simulink displays a warning or an error message, depending on whether it can or cannot encode the model, using the current character encoding, respectively. The warning or error message specifies the encoding of the current session and the encoding used to create the model. To avoid corrupting the model (see “Saving Models with Different Character Encodings” on page 3-12) and ensure correct display of the model’s text, you should:

- 1** Close all models open in the current session.
- 2** Use the `s1CharacterEncoding` command to change the character encoding of the current MATLAB session to that of the model as specified in the warning message.
- 3** Reopen the model.

You can now safely edit and save the model.

## Avoiding Initial Model Open Delay

You may notice that the first model that you open in a MATLAB session takes longer to open than do subsequent models. This is because to reduce its own startup time and to avoid unnecessary consumption of your system’s memory, MATLAB does not load Simulink into memory until the first time you open a Simulink model. You can cause MATLAB to load Simulink at MATLAB startup, and thus avoid the initial model opening delay, using either the `-r` MATLAB command line option or your `MATLAB_startup.m` file to run either `load_simulink` (loads Simulink) or `simulink` (loads Simulink and opens the Simulink Library browser) at MATLAB startup. For example, to load Simulink at MATLAB startup on Microsoft Windows systems, create a desktop shortcut with the following target:

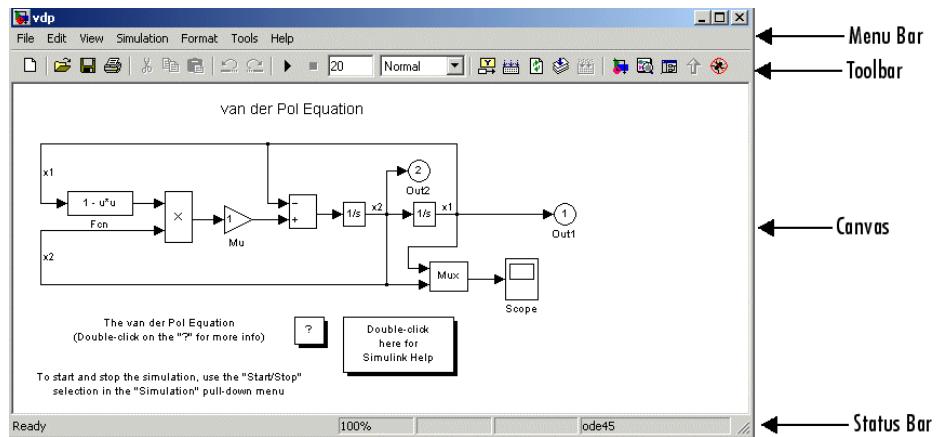
```
<matlabroot>\bin\win32\matlab.exe -r load_simulink
```

Similarly, the following command loads Simulink at MATLAB startup on Macintosh or Linux systems:

```
matlab -r load_simulink
```

# Model Editor

When you open a Simulink model or library, Simulink displays the model or library in an instance of the Model Editor.



## Editor Components

The Model Editor includes the following components.

### Menu Bar

The Simulink menu bar contains commands for creating, editing, viewing, printing, and simulating models. The menu commands apply to the model displayed in the editor. See Chapter 4, “Creating a Model” and Chapter 7, “Running Simulations” for more information.

### Toolbar

The toolbar allows you to execute Simulink’s most frequently used Simulink commands with a click of a mouse button. For example, to open a Simulink model, click the open folder icon on the toolbar. Letting the mouse cursor hover over a toolbar button or control causes a tooltip to appear. The tooltip describes the purpose of the button or control. You can hide the toolbar by clearing the **Toolbar** option on the Simulink **View** menu.

## Canvas

The canvas displays the model's block diagram. The canvas allows you to edit the block diagram. You can use your system's mouse and keyboard to create and connect blocks, select and move blocks, edit block labels, display block dialog boxes, and so on. See Chapter 5, "Working with Blocks" for more information.

## Context Menus

Simulink displays a context-sensitive menu when you click the right mouse button over the canvas. The contents of the menu depend on whether a block is selected. If a block is selected, the menu displays commands that apply only to the selected block. If no block is selected, the menu displays commands that apply to a model or library as a whole.

## Status Bar

The status bar appears only in the Windows version of the Model Editor. When a simulation is running, the status bar displays the status of the simulation, including the current simulation time and the name of the current solver. Regardless of the simulation state, the status bar also displays the zoom factor of the model editor window expressed as a percentage of normal (100%). You can display or hide the status bar by selecting or clearing the **Status Bar** option on the Simulink **View** menu.

## Undoing a Command

You can cancel the effects of up to 101 consecutive operations by choosing **Undo** from the **Edit** menu. You can undo these operations:

- Adding, deleting, or moving a block
- Adding, deleting, or moving a line
- Adding, deleting, or moving a model annotation
- Editing a block name
- Creating a subsystem (see "Undoing Subsystem Creation" on page 4-26 for more information)

You can reverse the effects of an **Undo** command by choosing **Redo** from the **Edit** menu.

## Zooming Block Diagrams

Simulink allows you to enlarge or shrink the view of the block diagram in the current Simulink window. To zoom a view:

- Select **Zoom In** from the **View** menu (or type **r**) to enlarge the view.
- Select **Zoom Out** from the **View** menu (or type **v**) to shrink the view.
- Select **Fit System To View** from the **View** menu (or press the space bar) to fit the diagram to the view.
- Select **Normal** from the **View** menu (or type **1**) to view the diagram at actual size.

By default, Simulink fits a block diagram to view when you open the diagram either in the model browser's content pane or in a separate window. If you change a diagram's zoom setting, Simulink saves the setting when you close the diagram and restores the setting the next time you open the diagram. If you want to restore the default behavior, choose **Fit System to View** from the **View** menu the next time you open the diagram.

## Panning Block Diagrams

You can use the mouse to pan model diagrams that are too large to fit in the Model Editor's window. To do this, position the mouse over the diagram, hold down the **p** or **q** key on the keyboard, then hold down the left mouse button. Moving the mouse now pans the model diagram in the editor window.

## View Command History

Simulink maintains a history of the modeling viewing commands, i.e., pan and zoom, that you execute for each model window. The history allows you to quickly return to a previous view in a window, using the following commands, accessible from the Model Editor's **View** menu and tool bar:

- **Back** ()—Displays the previous view in the view history.
- **Forward** ()—Displays the next view in the view history.

- **Go To Parent** ()—Opens, if necessary, the parent of the current subsystem and brings its window to the top of the desktop.

---

**Note** Simulink maintains a separate view history for each model window opened in the current session. As a result, the **View > Back** and **View > Forward** commands cannot cross window boundaries. For example, if window reuse is not on and you open a subsystem in another window, you cannot use the **View > Back** command to go to the window displaying the parent system. You must use the **View > Go To Parent** command in this case. On the other hand, if you enable window reuse and open a subsystem in the current window, you can use **View > Back** to restore the parent view.

---

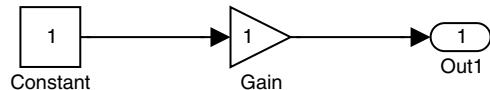
## Updating a Block Diagram

Simulink allows you to leave many attributes of a block diagram, such as signal data types and sample times, unspecified. Simulink then infers the values of block diagram attributes based on block connectivity and attributes that you do specify, a process known as *updating the diagram*. Simulink tries to infer the most appropriate value for an attribute that you do not specify. If Simulink cannot infer an attribute, it halts the update and displays an error dialog box.

Simulink updates a model's block diagram at the start of every simulation of a model. This assures that the simulation reflects the latest changes that you have made to a model. In addition, you can command Simulink to update a diagram at any time by selecting **Edit > Update Diagram** from the Model Editor's menu bar or context menu or by pressing **Ctrl+D**. This allows you to determine the values of block diagram attributes inferred by Simulink immediately after opening or editing a model.

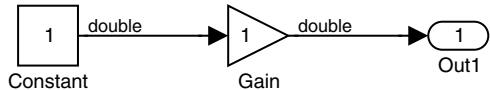
For example:

- 1 Create the following model.



- 2 Select **Format > Port/Signal Displays > Port Data Types** from the Model Editor's menu bar.

Simulink displays the data types of the output ports of the Constant and Gain blocks. Note that the data type of both ports is **double**, the default value.

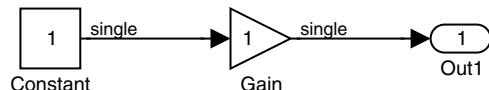


- 3 Set the **Signal Data Type** parameter of the Constant block (see the Constant block in the online Simulink reference documentation) to **single**.

Note that the output port data type displays on the block diagram do not reflect this change.

- 4 Select **Edit > Update Diagram** from the Model Editor's menu bar or press **Ctrl-D**.

Simulink updates the block diagram to reflect the change that you made previously.



Note that Simulink has inferred a data type for the output of the Gain block. This is because you did not specify a data type for the block. The data type inferred by Simulink is **single** because single precision is all that is necessary to simulate the model accurately, given that the precision of the block's input is **single**.

## Saving a Model

You can save a model by choosing either the **Save** or **Save As** command from the **File** menu. Simulink saves the model by generating a specially formatted file called the *model file* (with the .mdl extension) that contains the block diagram and block properties.

If you are saving a model for the first time, use the **Save** command to provide a name and location for the model file. Model file names must start with a letter and can contain no more than 63 letters, numbers, and underscores. The file name must not be the same as that of a MATLAB command.

If you are saving a model whose model file was previously saved, use the **Save** command to replace the file's contents or the **Save As** command to save the model with a new name or location. You can also use the **Save As** command to save the model in a format compatible with previous releases of Simulink (see “Saving a Model in Earlier Formats” on page 3-12).

Simulink follows this procedure while saving a model:

- 1 If the mdl file for the model already exists, it is renamed as a temporary file.
- 2 Simulink executes all block PreSaveFcn callback routines, then executes the block diagram's PreSaveFcn callback routine.
- 3 Simulink writes the model file to a new file using the same name and an extension of mdl.
- 4 Simulink executes all block PostSaveFcn callback routines, then executes the block diagram's PostSaveFcn callback routine.
- 5 Simulink deletes the temporary file.

If an error occurs during this process, Simulink renames the temporary file to the name of the original model file, writes the current version of the model to a file with an .err extension, and issues an error message. Simulink performs steps 2 through 4 even if an error occurs in an earlier step.

## Saving Models with Different Character Encodings

When Simulink saves a model, it uses the character encoding in effect when the model was created (the original encoding) to encode the text stored in the model's .mdl file, regardless of the character encoding in effect when the model is saved. This can lead to model corruption if you save a model whose original encoding differs from encoding currently in effect in the MATLAB session.

For example, it's possible you could have introduced characters that cannot be represented in the model's original encoding. If this is the case, Simulink saves the model as **model.err** where **model** is the model's name, leaving the original model file unchanged. Simulink also displays an error message that specifies the line and column number of the first unrepresentable character. To recover from this error without losing all the changes you've made to the model in the current session, use the following procedure. First, use a text editor to find the character in the .err file at the position specified by the save error message. Then, returning to Simulink, find and delete the corresponding character in the open model and resave the model . Repeat this process until you are able to save the model without error.

It's possible that your model's original encoding can represent all the text changes that you've made in the current session, albeit incorrectly. For example, suppose you open a model whose original encoding is A in a MATLAB session whose current encoding is B. Further, suppose you edit the model to include a character that has different encodings in A and B and then save the model. For example, suppose that the encoding for x in B is the same as the coding for y in A and you insert x in the model while B is in effect, save the model, and then reopen the model with A in effect. In this scenario, Simulink will display x as y. To alert you to the possibility of such corruptions, Simulink displays a warning message when you save a model and the current and original encoding differ but the original encoding can encode, possibly incorrectly, all the characters to be saved in the model file.

## Saving a Model in Earlier Formats

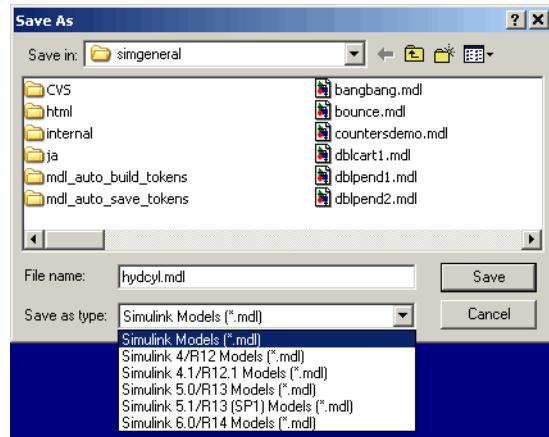
The **Save As** command allows you to save a model created with the latest version of Simulink in formats used by earlier versions of Simulink, including Simulink 4 (Release 12), Simulink 4.1 (Release 12.1), Simulink 5 (Release 13), Simulink 5.1 (Release 13SP1), and Simulink 6 (Release 14, compatible with Release 14, Release 14SP1, and Release 14SP2). You might want to do this,

for example, if you need to make a model available to colleagues who have access only to one of these earlier versions of Simulink.

To save a model in earlier format:

- 1 Select **Save As** from the Simulink **File** menu.

Simulink displays the **Save As** dialog box.



- 2 Select a format from the **Save as type** list on the dialog box.

- 3 Click the **Save** button.

When saving a model in an earlier version's format, Simulink saves the model in that format regardless of whether the model contains blocks and features that were introduced after that version. If the model does contain blocks or use features that postdate the earlier version, the model might not give correct results when run by the earlier version. For example, matrix and frame signals do not work in Release 11, because Release 11 does not have matrix and frame support. Similarly, models that contain unconditionally executed subsystems marked **Treat as atomic unit** might produce different results in Release 11, because Release 11 does not support unconditionally executed atomic subsystems.

The command converts blocks that postdate the earlier version into empty masked subsystem blocks colored yellow. For example, post-Release 11 blocks include

- Lookup Table (n-D)
- Assertion
- Rate Transition
- PreLookup Index Search
- Interpolation (n-D)
- Direct Lookup Table (n-D)
- Polynomial
- Matrix Concatenation
- Signal Specification
- Bus Creator
- If, WhileIterator, ForIterator, Assignment
- SwitchCase
- Bitwise Logical Operator

Post-Release 11 blocks from Simulink blocksets appear as unlinked blocks.

# Printing a Block Diagram

You can print a block diagram by selecting **Print** from the **File** menu (on a Microsoft Windows system) or by using the `print` command in the MATLAB Command Window (on all platforms).

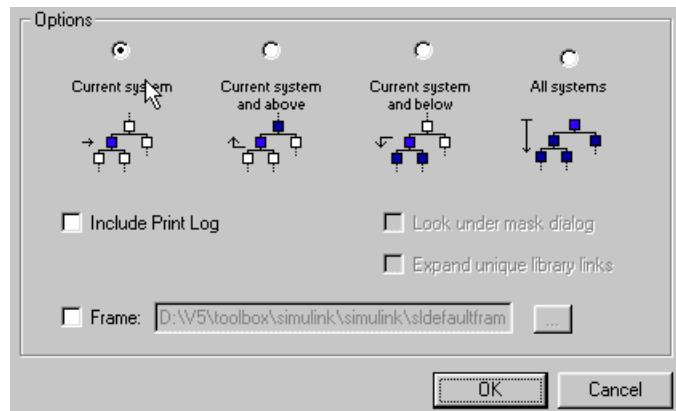
On a Microsoft Windows system, the **Print** menu item prints the block diagram in the current window.

## Print Dialog Box

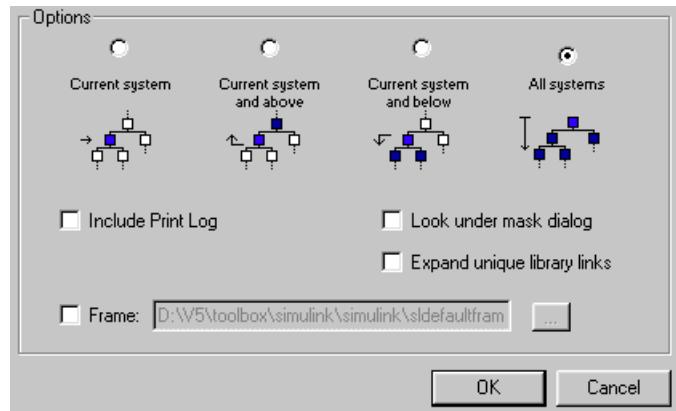
When you select the **Print** menu item, the **Print** dialog box appears. The **Print** dialog box enables you to selectively print systems within your model. Using the dialog box, you can print

- The current system only
- The current system and all systems above it in the model hierarchy
- The current system and all systems below it in the model hierarchy, with the option of looking into the contents of masked and library blocks
- All systems in the model, with the option of looking into the contents of masked and library blocks
- An overlay frame on each diagram

The portion of the **Print** dialog box that supports selective printing is similar on supported platforms. This figure shows how it looks on a Microsoft Windows system. In this figure, only the current system is to be printed.



When you select either the **Current system and below** or **All systems** option, two check boxes become enabled. In this figure, **All systems** is selected.



Selecting the **Look under mask dialog** check box prints the contents of masked subsystems when encountered at or below the level of the current block. When you are printing all systems, the top-level system is considered the current block, so Simulink looks under any masked blocks encountered.

Selecting the **Expand unique library links** check box prints the contents of library blocks when those blocks are systems. Only one copy is printed regardless of how many copies of the block are contained in the model. For more information about libraries, see “Working with Block Libraries” on page 5-21.

The print log lists the blocks and systems printed. To print the print log, select the **Include Print Log** check box.

Selecting the **Frame** check box prints a title block frame on each diagram. Enter the path to the title block frame in the adjacent edit box. You can create a customized title block frame, using the MATLAB frame editor. See `frameedit` in the MATLAB reference for information on using the frame editor to create title block frames.

## Print Command

The format of the print command is

```
print -ssys -device filename
```

*sys* is the name of the system to be printed. The system name must be preceded by the *s* switch identifier and is the only required argument. *sys* must be open or must have been open during the current session. If the system name contains spaces or takes more than one line, you need to specify the name as a string. See the examples below.

*device* specifies a device type. For a list and description of device types, see the documentation for the MATLAB `print` function.

*filename* is the PostScript file to which the output is saved. If *filename* exists, it is replaced. If *filename* does not include an extension, an appropriate one is appended.

For example, this command prints a system named `untitled`.

```
print -suntitled
```

This command prints the contents of a subsystem named Sub1 in the current system.

```
print -sSub1
```

This command prints the contents of a subsystem named Requisite Friction.

```
print ([-sRequisite Friction])
```

The next example prints a system named Friction Model, a subsystem whose name appears on two lines. The first command assigns the newline character to a variable; the second prints the system.

```
cr = sprintf('\n');
print ([-sFriction' cr 'Model'])
```

To print the currently selected subsystem, enter

```
print([-s', gcb])
```

## Specifying Paper Size and Orientation

Simulink lets you specify the type and orientation of the paper used to print a model diagram. You can do this on all platforms by setting the model's PaperType and PaperOrientation properties, respectively (see "Model and Block Properties" in the online Simulink reference), using the set\_param command. You can set the paper orientation alone, using the MATLAB orient command. On Windows, the **Print** and **Printer Setup** dialog boxes let you set the page type and orientation properties as well.

## Positioning and Sizing a Diagram

You can use a model's PaperPositionMode and PaperPosition parameters to position and size the model's diagram on the printed page. The value of the PaperPosition parameter is a vector of form [left bottom width height]. The first two elements specify the bottom-left corner of a rectangular area on the page, measured from the page's bottom-left corner. The last two elements specify the width and height of the rectangle. When the model's PaperPositionMode is manual, Simulink positions (and scales, if necessary)

the model's diagram to fit inside the specified print rectangle. For example, the following commands

```
vdp  
set_param('vdp', 'PaperType', 'usletter')  
set_param('vdp', 'PaperOrientation', 'landscape')  
set_param('vdp', 'PaperPositionMode', 'manual')  
set_param('vdp', 'PaperPosition', [0.5 0.5 4 4])  
print -svdp
```

print the block diagram of the vdp sample model in the lower-left corner of a U.S. letter-size page in landscape orientation.

If PaperPositionMode is auto, Simulink centers the model diagram on the printed page, scaling the diagram, if necessary, to fit the page.

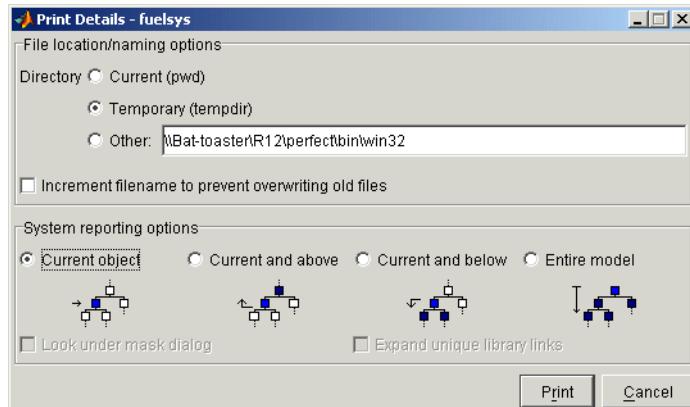
## Generating a Model Report

A Simulink model report is an HTML document that describes a model's structure and content. The report includes block diagrams of the model and its subsystems and the settings of its block parameters.

To generate a report for the current model:

- 1 Select **Print details** from the model's **File** menu.

The **Print Details** dialog box appears.

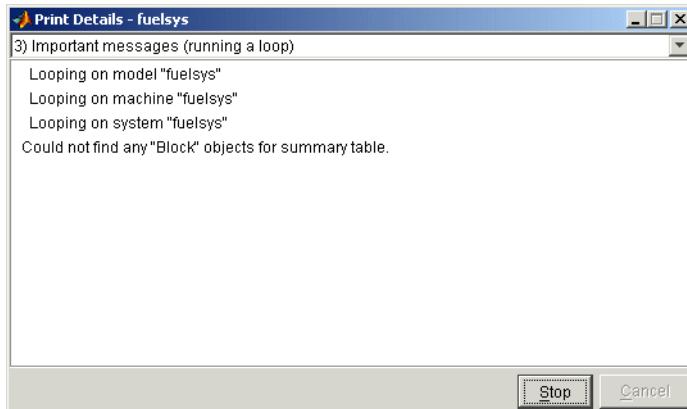


The dialog box allows you to select various report options (see "Model Report Options" on page 3-21).

- 2 Select the desired report options on the dialog box.
- 3 Select **Print**.

Simulink generates the HTML report and displays it in your system's default HTML browser.

While generating the report, Simulink displays status messages on a messages pane that replaces the options pane on the **Print Details** dialog box.



You can select the detail level of the messages from the list at the top of the messages pane. When the report generation process begins, the **Print** button on the **Print Details** dialog box changes to a **Stop** button. Clicking this button terminates the report generation. When the report generation process finishes, the **Stop** button changes to an **Options** button. Clicking this button redisplays the report generation options, allowing you to generate another report without having to reopen the **Print Details** dialog box.

## Model Report Options

The **Print Details** dialog box allows you to select the following report options.

### Directory

The directory where Simulink stores the HTML report that it generates. The options include your system's temporary directory (the default), your system's current directory, or another directory whose path you specify in the adjacent edit field.

### Increment filename to prevent overwriting old files

Creates a unique report file name each time you generate a report for the same model in the current session. This preserves each report.

**Current object**

Include only the currently selected object in the report.

**Current and above**

Include the current object and all levels of the model above the current object in the report.

**Current and below**

Include the current object and all levels below the current object in the report.

**Entire model**

Include the entire model in the report.

**Look under mask dialog**

Include the contents of masked subsystems in the report.

**Expand unique library links**

Include the contents of library blocks that are subsystems. The report includes a library subsystem only once even if it occurs in more than one place in the model.

# Summary of Mouse and Keyboard Actions

Simulink provides mouse and keyboard shortcuts for many of its commands. The following tables summarize these shortcuts.

- “Model Viewing Shortcuts” on page 3-23
- “Block Editing Shortcuts” on page 3-24
- “Line Editing Shortcuts” on page 3-25
- “Signal Label Editing Shortcuts” on page 3-25
- “Annotation Editing Shortcuts” on page 3-26

LMB means press the left mouse button; CMB, the center mouse button; and RMB, the right mouse button.

## Model Viewing Shortcuts

The following table lists keyboard shortcuts for viewing models.

Task	Microsoft Windows	Macintosh or Linux
Zoom in	<b>r</b>	<b>r</b>
Zoom out	<b>v</b>	<b>v</b>
Zoom to normal (100%)	<b>1</b>	<b>1</b>
Pan left	<b>d</b> or <b>Ctrl+Left Arrow</b>	<b>d</b> or <b>Ctrl+Left Arrow</b>
Pan right	<b>g</b> or <b>Ctrl+Right Arrow</b>	<b>g</b> or <b>Ctrl+Right Arrow</b>
Pan up	<b>e</b> or <b>Ctrl+Up Arrow</b>	<b>e</b> or <b>Ctrl+Up Arrow</b>
Pan down	<b>c</b> or <b>Ctrl+Down Arrow</b>	<b>c</b> or <b>Ctrl+Down Arrow</b>
Fit selection to screen	<b>f</b>	<b>f</b>
Fit diagram to screen	<b>Space</b>	<b>Space</b>
Pan with mouse	Hold down <b>p</b> or <b>q</b> and drag mouse	Hold down <b>p</b> or <b>q</b> and drag mouse
Go back in pan/zoom history	<b>b</b> or <b>Shift+Left Arrow</b>	<b>b</b> or <b>Shift+Left Arrow</b>

<b>Task</b>	<b>Microsoft Windows</b>	<b>Macintosh or Linux</b>
Go forward in pan/zoom history	t or Shift+Right Arrow	t or Shift+Right Arrow
Delete selection	<b>Delete or Back Space</b>	<b>Delete or Back Space</b>
Move selection	Use arrow keys	Use arrow keys

## Block Editing Shortcuts

The following table lists mouse and keyboard actions that apply to blocks.

<b>Task</b>	<b>Microsoft Windows</b>	<b>Macintosh or Linux</b>
Select one block	LMB	LMB
Select multiple blocks	<b>Shift + LMB</b>	<b>Shift + LMB; or CMB alone</b>
Copy block from another window	Drag block	Drag block
Move block	Drag block	Drag block
Duplicate block	<b>Ctrl + LMB and drag; or RMB and drag</b>	<b>Ctrl + LMB and drag; or RMB and drag</b>
Connect blocks	LMB	LMB
Disconnect block	<b>Shift + drag block</b>	<b>Shift + drag block; or CMB and drag</b>
Open selected subsystem	<b>Enter</b>	<b>Return</b>
Go to parent of selected subsystem	<b>Esc</b>	<b>Esc</b>

## Line Editing Shortcuts

The following table lists mouse and keyboard actions that apply to lines.

<b>Task</b>	<b>Microsoft Windows</b>	<b>Macintosh or Linux</b>
Select one line	LMB	LMB
Select multiple lines	<b>Shift</b> + LMB	<b>Shift</b> + LMB; or CMB alone
Draw branch line	<b>Ctrl</b> + drag line; or RMB and drag line	<b>Ctrl</b> + drag line; or RMB + drag line
Route lines around blocks	<b>Shift</b> + draw line segments	<b>Shift</b> + draw line segments; or CMB and draw segments
Move line segment	Drag segment	Drag segment
Move vertex	Drag vertex	Drag vertex
Create line segments	<b>Shift</b> + drag line	<b>Shift</b> + drag line; or CMB + drag line

## Signal Label Editing Shortcuts

The next table lists mouse and keyboard actions that apply to signal labels.

<b>Action</b>	<b>Microsoft Windows</b>	<b>Macintosh or Linux</b>
Create signal label	Double-click line, then enter label	Double-click line, then enter label
Copy signal label	<b>Ctrl</b> + drag label	<b>Ctrl</b> + drag label
Move signal label	Drag label	Drag label
Edit signal label	Click in label, then edit	Click in label, then edit
Delete signal label	<b>Shift</b> + click label, then press <b>Delete</b>	<b>Shift</b> + click label, then press <b>Delete</b>

## Annotation Editing Shortcuts

The next table lists mouse and keyboard actions that apply to annotations.

Action	Microsoft Windows	Macintosh or Linux
Create annotation	Double-click in diagram, then enter text	Double-click in diagram, then enter text
Copy annotation	<b>Ctrl</b> + drag label	<b>Ctrl</b> + drag label
Move annotation	Drag label	Drag label
Edit annotation	Click in text, then edit	Click in text, then edit
Delete annotation	<b>Shift</b> + select annotation, then press <b>Delete</b>	<b>Shift</b> + select annotation, then press <b>Delete</b>

## Ending a Simulink Session

Terminate a Simulink session by closing all Simulink windows.

Terminate a MATLAB session by choosing one of these commands from the **File** menu:

- On a Microsoft Windows system: **Exit MATLAB**
- On a Macintosh or Linux system: **Quit MATLAB**



# Creating a Model

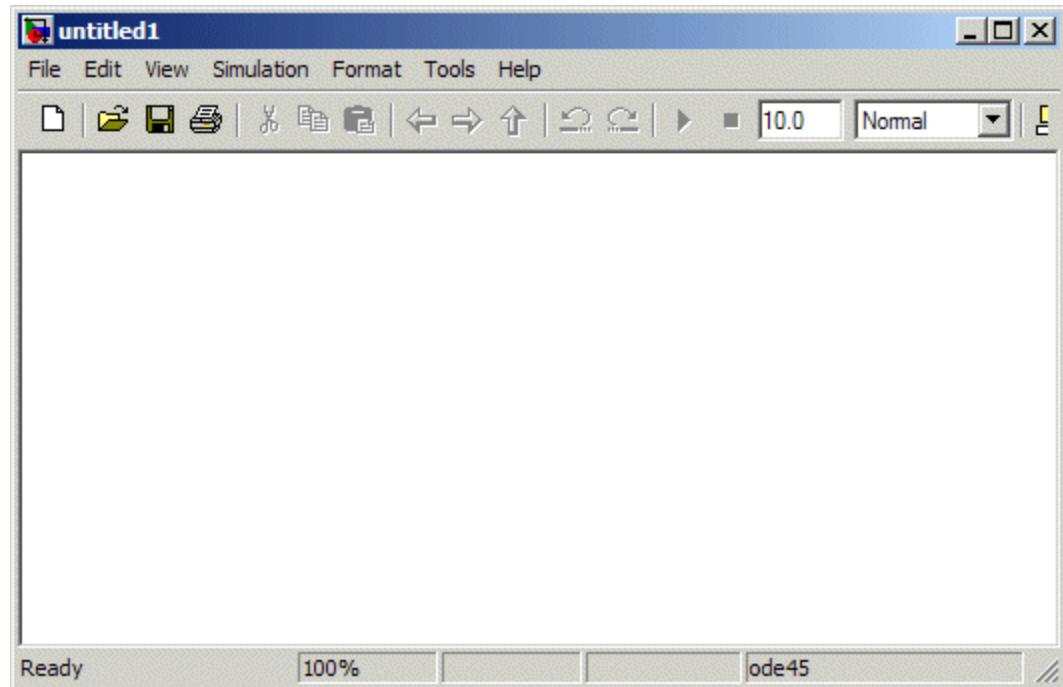
---

The following sections explain how to perform tasks required to create Simulink models.

Creating an Empty Model (p. 4-2)	How to create a new model.
Selecting Objects (p. 4-4)	How to select objects in a model.
Specifying Block Diagram Colors (p. 4-6)	How to specify the colors of blocks, lines, and annotations and the background of the diagram.
Connecting Blocks (p. 4-10)	How to draw connections between blocks.
Annotating Diagrams (p. 4-19)	How to add annotations to a block diagram.
Creating Subsystems (p. 4-24)	How to create subsystems.
Creating Conditionally Executed Subsystems (p. 4-29)	How to create subsystems that are executed only when specified events occur or conditions are satisfied.
Using Callback Functions (p. 4-41)	How to use callback routines to customize a model.

## Creating an Empty Model

To create an empty model, click the **New** button on the Library Browser's toolbar (Windows only) or choose **New** from the library window's **File** menu and select **Model**. Simulink creates an empty model in memory and displays it in a new model editor window.



## Creating a Model Template

When you create a new model, Simulink configures it with default settings. For instance, by default, new models have a white canvas, the `ode45` solver, and a visible toolbar. You can change these settings to your liking after creating the new model.

Alternatively, you can write a function that creates a model that has settings you prefer, using the commands listed in “Model Construction Commands”

in the online Simulink Reference. Executing this function generates a new model with your customized parameter settings.

For example, the following function creates a Simulink model featuring a green canvas, the `ode3` solver, and a hidden toolbar:

```
function new_model(modelname)
% NEW_MODEL Create a new, empty Simulink model
%   NEW_MODEL('MODELNAME') creates a new model with
%   the name 'MODELNAME'. Without the 'MODELNAME'
%   argument, the new model is named 'my_untitled'.

if nargin == 0
    modelname = 'my_untitled';
end

% create and open the model
open_system(new_system(modelname));

% set default screen color
set_param(modelname, 'ScreenColor', 'green');

% set default solver
set_param(modelname, 'Solver', 'ode3');

% set default toolbar visibility
set_param(modelname, 'Toolbar', 'off');

% save the model
save_system(modelname);
```

## Selecting Objects

Many model building actions, such as copying a block or deleting a line, require that you first select one or more blocks and lines (objects).

### Selecting an Object

To select an object, click it. Small black square handles appear at the corners of a selected block and near the end points of a selected line. For example, the figure below shows a selected Sine Wave block and a selected line.



When you select an object by clicking it, any other selected objects are deselected.

### Selecting Multiple Objects

You can select more than one object either by selecting objects one at a time, by selecting objects located near each other using a bounding box, or by selecting the entire model.

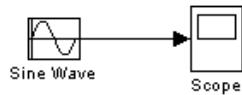
#### Selecting Multiple Objects One at a Time

To select more than one object by selecting each object individually, hold down the **Shift** key and click each object to be selected. To deselect a selected object, click the object again while holding down the **Shift** key.

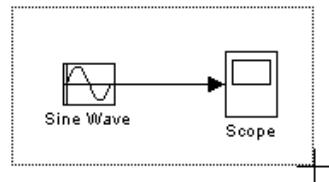
#### Selecting Multiple Objects Using a Bounding Box

An easy way to select more than one object in the same area of the window is to draw a bounding box around the objects:

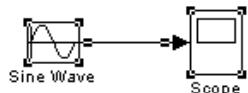
- 1 Define the starting corner of a bounding box by positioning the pointer at one corner of the box, then pressing and holding down the mouse button. Notice the shape of the cursor.



- 2 Drag the pointer to the opposite corner of the box. A dotted rectangle encloses the selected blocks and lines.



- 3 Release the mouse button. All blocks and lines at least partially enclosed by the bounding box are selected.



### Selecting All Objects

To select all objects in the active window, select **Select All** from the **Edit** menu. You cannot create a subsystem by selecting blocks and lines in this way. For more information, see “Creating Subsystems” on page 4-24.

## Specifying Block Diagram Colors

Simulink allows you to specify the foreground and background colors of any block or annotation in a diagram, as well as the diagram's background color. To set the background color of a block diagram, select **Screen color** from the Simulink **Format** menu. To set the background color of a block or annotation or group of such items, first select the item or items. Then select **Background color** from the Simulink **Format** menu. To set the foreground color of a block or annotation, first select the item. Then select **Foreground color** from the Simulink **Format** menu.

In all cases, Simulink displays a menu of color choices. Choose the desired color from the menu. If you select a color other than **Custom**, Simulink changes the background or foreground color of the diagram or diagram element to the selected color.

### Choosing a Custom Color

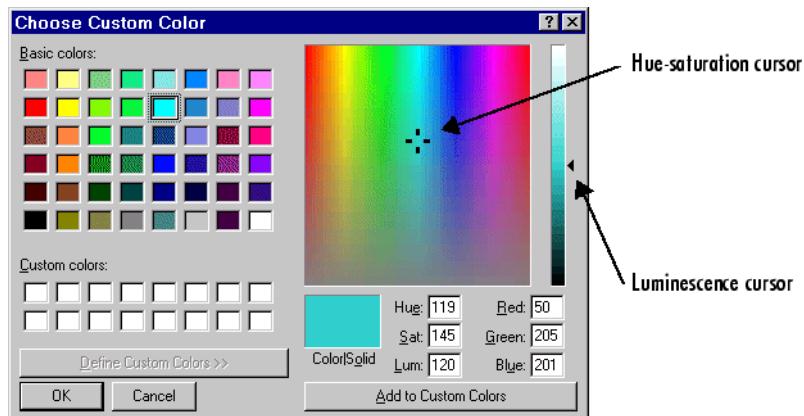
If you choose **Custom**, Simulink displays the Simulink **Choose Custom Color** dialog box.



The dialog box displays a palette of basic colors and a palette of custom colors that you previously defined. If you have not previously created any custom colors, the custom color palette is all white. To choose a color from either palette, click the color, and then click the **OK** button.

## Defining a Custom Color

To define a custom color, click the **Define Custom Colors** button on the **Choose Custom Color** dialog box. The dialog box expands to display a custom color definer.



The color definer allows you to specify a custom color by

- Entering the red, green, and blue components of the color as values between 0 (darkest) and 255 (brightest)
- Entering hue, saturation, and luminescence components of the color as values in the range 0 to 255
- Moving the hue-saturation cursor to select the hue and saturation of the desired color and the luminescence cursor to select the luminescence of the desired color

The color that you have defined in any of these ways appears in the **Color | Solid** box. To redefine a color in the **Custom colors** palette, select the color and define a new color, using the color definer. Then click the **Add to Custom Colors** button on the color definer.

## Specifying Colors Programmatically

You can use the `set_param` command at the MATLAB command line or in an M-file program to set parameters that determine the background color of a diagram and the background color and foreground color of diagram elements.

The following table summarizes the parameters that control block diagram colors.

Parameter	Determines
ScreenColor	Background color of block diagram
BackgroundColor	Background color of blocks and annotations
ForegroundColor	Foreground color of blocks and annotations

You can set these parameters to any of the following values:

- 'black', 'white', 'red', 'green', 'blue', 'cyan', 'magenta', 'yellow', 'gray', 'lightBlue', 'orange', 'darkGreen'
- '[r,g,b]'

where r, g, and b are the red, green, and blue components of the color normalized to the range 0.0 to 1.0.

For example, the following command sets the background color of the currently selected system or subsystem to a light green color:

```
set_param(gcs, 'ScreenColor', '[0.3, 0.9, 0.5]')
```

## Displaying Sample Time Colors

Simulink can color code the blocks and lines in your model to indicate the sample rates at which the blocks operate.

Color	Use
Black	Continuous sample time
Magenta	Constant sample time
Red	Fastest discrete sample time
Green	Second fastest discrete sample time
Blue	Third fastest discrete sample time
Light Blue	Fourth fastest discrete sample time
Dark Green	Fifth fastest discrete sample time

Color	Use
Orange	Sixth fastest discrete sample time
Yellow	<p>Can indicate one of the following:</p> <ul style="list-style-type: none"> <li>• A block with hybrid sample time, e.g., subsystems grouping blocks and Mux or Demux blocks grouping signals with different sample times, Data Store Memory blocks updated and read by different tasks.</li> <li>• Variable sample time. See the Pulse Generator block and “Specifying Sample Time” on page 2-31 for more information.</li> <li>• A block with the seventh, eighth, etc., sample time.</li> </ul>
Cyan	Blocks in triggered subsystems
Gray	Fixed in minor step

To enable the sample time colors feature, select **Sample Time Colors** from the **Format** menu.

Simulink does not automatically recolor the model with each change you make to it, so you must select **Update Diagram** from the **Edit** menu to explicitly update the model coloration. To return to your original coloring, disable sample time coloration by again choosing **Sample Time Colors**.

It is important to note that Mux and Demux blocks are simply grouping operators; signals passing through them retain their timing information. For this reason, the lines emanating from a Demux block can have different colors if they are driven by sources having different sample times. In this case, the Mux and Demux blocks are color coded as hybrids (yellow) to indicate that they handle signals with multiple rates.

Similarly, Subsystem blocks that contain blocks with differing sample times are also colored as hybrids, because there is no single rate associated with them. If all the blocks within a subsystem run at a single rate, the Subsystem block is colored according to that rate.

## Connecting Blocks

Simulink block diagrams use lines to represent pathways for signals among blocks in a model (see Chapter 6, “Working with Signals” for information on signals). Simulink can connect blocks for you or you can connect the blocks yourself by drawing lines from their output ports to their input ports.

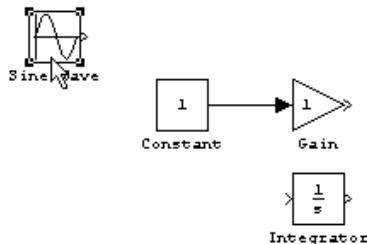
### Automatically Connecting Blocks

You can command Simulink to connect blocks automatically. This eliminates the need for you to draw the connecting lines yourself. When connecting blocks, Simulink routes lines around intervening blocks to avoid cluttering the diagram.

### Connecting Two Blocks

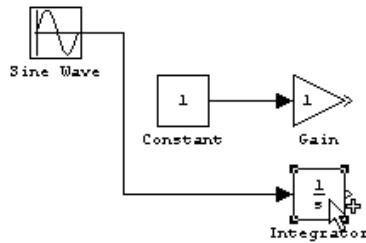
To autoconnect two blocks:

- 1 Select the source block.

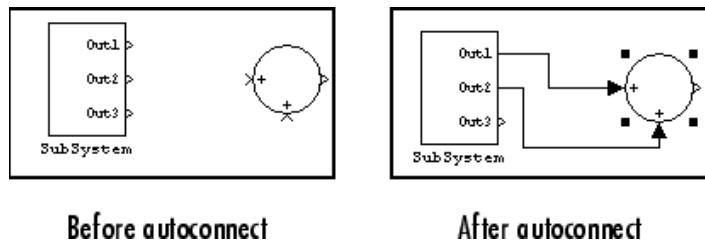


- 2** Hold down **Ctrl** and left-click the destination block.

Simulink connects the source block to the destination block, routing the line around intervening blocks if necessary.



When connecting two blocks, Simulink draws as many connections as possible between the two blocks as illustrated in the following example.

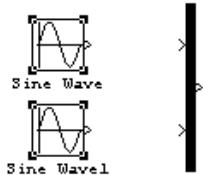


## Connecting Groups of Blocks

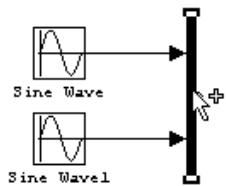
Simulink can connect a group of source blocks to a destination block or a source block to a group of destination blocks.

To connect a group of source blocks to a destination block:

- 1 Select the source blocks.

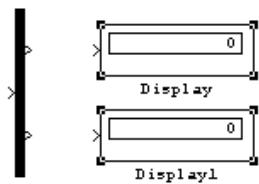


- 2 Hold down **Ctrl** and left-click the destination block.

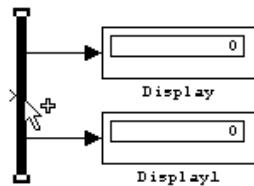


To connect a source block to a group of destination blocks:

- 1 Select the *destination* blocks.



- 2 Hold down **Ctrl** and left-click the *source* block.



## Manually Connecting Blocks

Simulink allows you to draw lines manually between blocks or between lines and blocks. You might want to do this if you need to control the path of the line or to create a branch line.

### Drawing a Line Between Blocks

To connect the output port of one block to the input port of another block:

- 1 Position the cursor over the first block's output port. It is not necessary to position the cursor precisely on the port. The cursor shape changes to crosshairs.



- 2 Press and hold down the mouse button.
- 3 Drag the pointer to the second block's input port. You can position the cursor on or near the port or in the block. If you position the cursor in the block, the line is connected to the closest input port. The cursor shape changes to double crosshairs.



- 4 Release the mouse button. Simulink replaces the port symbols by a connecting line with an arrow showing the direction of the signal flow. You can create lines either from output to input, or from input to output. The arrow is drawn at the appropriate input port, and the signal is the same.

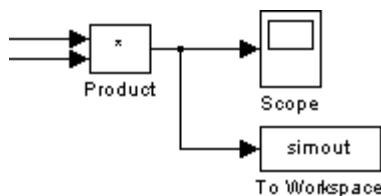


Simulink draws connecting lines using horizontal and vertical line segments. To draw a diagonal line, hold down the **Shift** key while drawing the line.

## Drawing a Branch Line

A *branch line* is a line that starts from an existing line and carries its signal to the input port of a block. Both the existing line and the branch line carry the same signal. Using branch lines enables you to cause one signal to be carried to more than one block.

In this example, the output of the Product block goes to both the Scope block and the To Workspace block.



To add a branch line:

- 1 Position the pointer on the line where you want the branch line to start.
- 2 While holding down the **Ctrl** key, press and hold down the left mouse button.
- 3 Drag the pointer to the input port of the target block, then release the mouse button and the **Ctrl** key.

You can also use the right mouse button instead of holding down the left mouse button and the **Ctrl** key.

## Drawing a Line Segment

You might want to draw a line with segments exactly where you want them instead of where Simulink draws them. Or you might want to draw a line before you copy the block to which the line is connected. You can do either by drawing line segments.

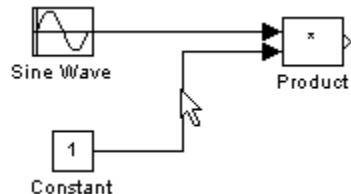
To draw a line segment, you draw a line that ends in an unoccupied area of the diagram. An arrow appears on the unconnected end of the line. To add another line segment, position the cursor over the end of the segment and draw another segment. Simulink draws the segments as horizontal and

vertical lines. To draw diagonal line segments, hold down the **Shift** key while you draw the lines.

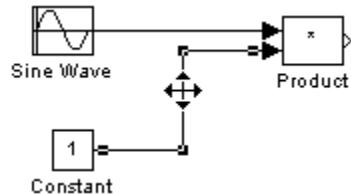
### Moving a Line Segment

To move a line segment:

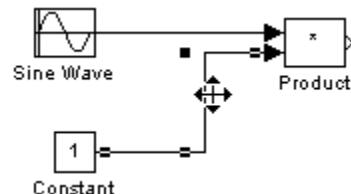
- 1 Position the pointer on the segment you want to move.



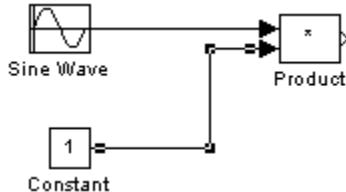
- 2 Press and hold down the left mouse button.



- 3 Drag the pointer to the desired location.



- 4** Release the mouse button.



To move the segment connected to an input port, position the pointer over the port and drag the end of the segment to the new location. You cannot move the segment connected to an output port.

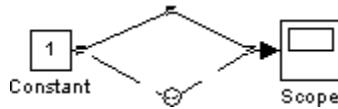
### Moving a Line Vertex

To move a vertex of a line:

- 1** Position the pointer on the vertex, then press and hold down the mouse button. The cursor changes to a circle that encloses the vertex.



- 2** Drag the pointer to the desired location.



- 3** Release the mouse button.

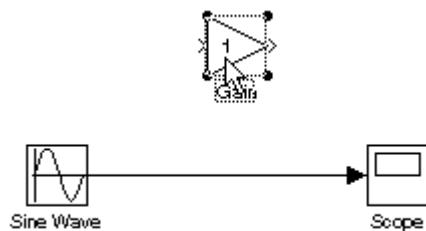


## Inserting Blocks in a Line

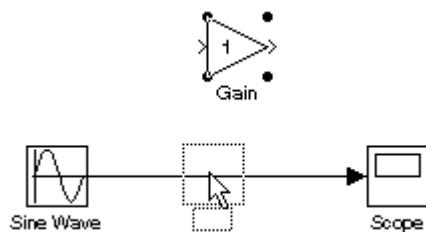
You can insert a block in a line by dropping the block on the line. Simulink inserts the block for you at the point where you drop the block. The block that you insert can have only one input and one output.

To insert a block in a line:

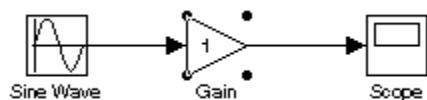
- 1 Position the pointer over the block and press the left mouse button.



- 2 Drag the block over the line in which you want to insert the block.



- 3 Release the mouse button to drop the block on the line. Simulink inserts the block where you dropped it.

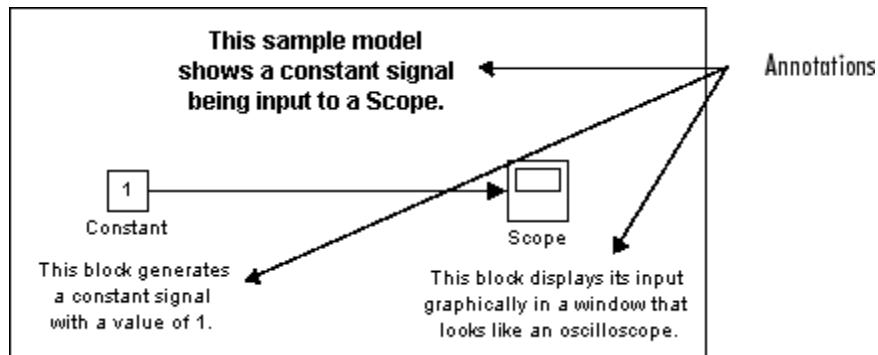


## Disconnecting Blocks

To disconnect a block from its connecting lines, hold down the **Shift** key, then drag the block to a new location.

## Annotating Diagrams

Annotations provide textual information about a model. You can add an annotation to any unoccupied area of your block diagram.



To create a model annotation, double-click an unoccupied area of the block diagram. A small rectangle appears and the cursor changes to an insertion point. Start typing the annotation contents. Each line is centered within the rectangle that surrounds the annotation.

To move an annotation, drag it to a new location.

To edit an annotation, select it:

- To replace the annotation, click the annotation, then double-click or drag the cursor to select it. Then, enter the new annotation.
- To insert characters, click between two characters to position the insertion point, then insert text.
- To replace characters, drag the mouse to select a range of text to replace, then enter the new text.

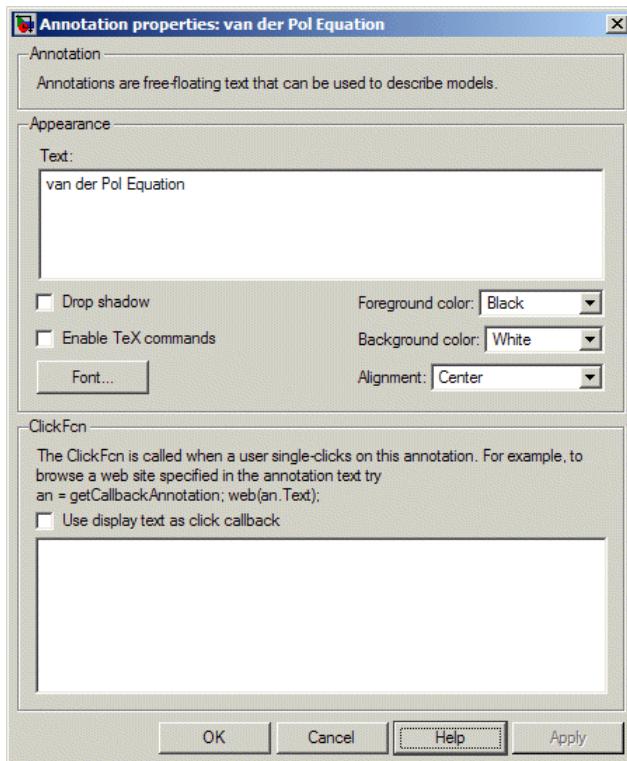
To delete an annotation, hold down the **Shift** key while you select the annotation, then press the **Delete** or **Backspace** key.

To change the font of all or part of an annotation, select the text in the annotation you want to change, then choose **Font** from the **Format** menu. Select a font and size from the dialog box.

To change the text alignment (e.g., left, center, or right) of the annotation, select the annotation and choose **Text Alignment** from the model window's **Format** or context menu. Then choose one of the alignment options (e.g., **Center**) from the **Text Alignment** submenu.

## Annotations Properties Dialog Box

The **Annotation Properties** dialog box allows you to specify the contents and format of the currently selected annotation and to associate a click function with the annotation. To display the **Annotation Properties** dialog box for an annotation, select the annotation and then select **Annotation Properties** from the model editor's **Edit** or context menu. The dialog box appears.



The dialog box includes the following controls.

### Text

Displays the current text of the annotation. Edit this field to change the annotation text.

**Drop shadow**

Checking this option causes Simulink to display a drop shadow around the annotation, giving it a 3-D appearance.

**Enable TeX commands**

Checking this option enables use of TeX formatting commands in this annotation. See “Using TeX Formatting Commands in Annotations” in the online Simulink documentation for more information.

**Font**

Clicking this button displays a font chooser dialog box. Use the font chooser to change the font used to render the annotation’s text.

**Foreground color**

Specifies the color of the annotation text.

**Background color**

Specifies the color of the background of the annotation’s bounding box.

**Alignment**

Specifies the alignment of the annotation’s text relative to its bounding box.

**ClickFcn**

Specifies MATLAB code to be executed when a user single-clicks this annotation. Simulink stores the code entered in this field with the model. See “Associating Click Functions with Annotations” in the online Simulink documentation for more information.

**Use display text as click callback**

Checking this option causes Simulink to treat the text in the **Text** field as the annotation’s click function. The specified text must be a valid MATLAB expression comprising symbols that are defined in the MATLAB workspace when the user clicks this annotation. See “Associating Click Functions with

Annotations” for more information. Note that selecting this option disables the **ClickFcn** edit field.

## Annotations API

Simulink provides an application program interface (API) that enables you to use M programs to get and set the properties of annotations. The API comprises the following elements:

- `Simulink.Annotation` class

Allows M-code, e.g., annotation load functions (see “Load Function” in the online Simulink documentation), to set the properties of annotations

- `getCallbackAnnotation` function

Gets the `Simulink.Annotation` object for the annotation associated with the currently executing annotation callback function

## Creating Subsystems

As your model increases in size and complexity, you can simplify it by grouping blocks into subsystems. Using subsystems has these advantages:

- It helps reduce the number of blocks displayed in your model window.
- It allows you to keep functionally related blocks together.
- It enables you to establish a hierarchical block diagram, where a Subsystem block is on one layer and the blocks that make up the subsystem are on another.

You can create a subsystem in two ways:

- Add a Subsystem block to your model, then open that block and add the blocks it contains to the subsystem window.
- Add the blocks that make up the subsystem, then group those blocks into a subsystem.

### Creating a Subsystem by Adding the Subsystem Block

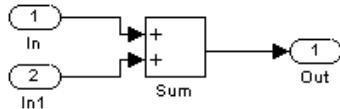
To create a subsystem before adding the blocks it contains, add a Subsystem block to the model, then add the blocks that make up the subsystem:

- 1 Copy the Subsystem block from the Ports & Subsystems library into your model.
- 2 Open the Subsystem block by double-clicking it.

Simulink opens the subsystem in the current or a new model window, depending on the model window reuse mode that you selected (see “Window Reuse” on page 4-27).

- 3 In the empty Subsystem window, create the subsystem. Use Import blocks to represent input from outside the subsystem and Outport blocks to represent external output.

For example, the subsystem shown includes a Sum block and Import and Outport blocks to represent input to and output from the subsystem.

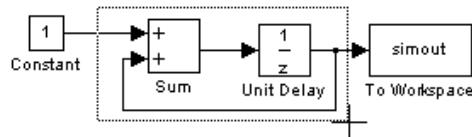


## Creating a Subsystem by Grouping Existing Blocks

If your model already contains the blocks you want to convert to a subsystem, you can create the subsystem by grouping those blocks:

- 1 Enclose the blocks and connecting lines that you want to include in the subsystem within a bounding box. You cannot specify the blocks to be grouped by selecting them individually or by using the **Select All** command. For more information, see “Selecting Multiple Objects Using a Bounding Box” on page 4-4.

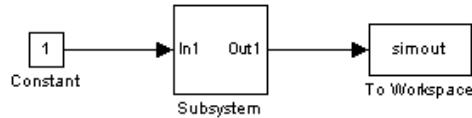
For example, this figure shows a model that represents a counter. The Sum and Unit Delay blocks are selected within a bounding box.



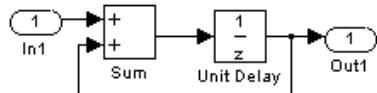
When you release the mouse button, the two blocks and all the connecting lines are selected.

- 2 Choose **Create Subsystem** from the **Edit** menu. Simulink replaces the selected blocks with a Subsystem block.

This figure shows the model after you choose the **Create Subsystem** command (and resize the Subsystem block so the port labels are readable).



If you open the Subsystem block, Simulink displays the underlying system, as shown below. Notice that Simulink adds Inport and Outport blocks to represent input from and output to blocks outside the subsystem.



As with all blocks, you can change the name of the Subsystem block. You can also use the masking feature to customize the block's appearance and dialog box. See Chapter 9, “Creating Masked Subsystems”.

### **Undoing Subsystem Creation**

To undo creation of a subsystem by grouping blocks, select **Undo** from the **Edit** menu. You can undo creation of a subsystem that you have subsequently edited. However, the **Undo** command does not undo any nongraphical changes that you made to the blocks, such as changing the value of a block parameter or the name of a block. Simulink alerts you to this limitation by displaying a warning dialog box before undoing creation of a modified subsystem.

## **Model Navigation Commands**

Subsystems allow you to create a hierarchical model comprising many layers. You can navigate this hierarchy using the Simulink Model Browser (see “The Model Browser” on page 8-22) and/or the following model navigation commands:

- **Open Block**

The **Open Block** command opens the currently selected subsystem. To execute the command, select **Open Block** from either the Simulink **Edit** menu or the subsystem’s context (right-click) menu, press **Enter**, or double-click the subsystem.

- **Open Block In New Window**

Opens the currently selected subsystem regardless of the Simulink window reuse settings (see “Window Reuse” on page 4-27). To execute the command, select **Open Block In New Window** from the subsystem’s context (right-click) menu.

- **Go To Parent**

The **Go To Parent** command displays the parent of the subsystem displayed in the current window. To execute the command, press **Esc** or select **Go To Parent** from the Simulink **View** menu.

## Window Reuse

You can specify whether Simulink model navigation commands use the current window or a new window to display a subsystem or its parent. Reusing windows avoids cluttering your screen with windows. Creating a window for each subsystem allows you to view subsystems side by side with their parents or siblings. To specify your preference regarding window reuse, select **Preferences** from the Simulink **File** menu and then select one of the following **Window reuse type** options listed in the Simulink **Preferences** dialog box.

<b>Reuse Type</b>	<b>Open Action</b>	<b>Go to Parent (Esc) Action</b>
none	Subsystem appears in a new window.	Parent window moves to the front.
reuse	Subsystem replaces the parent in the current window.	Parent window replaces subsystem in current window
replace	Subsystem appears in a new window. Parent window disappears.	Parent window appears. Subsystem window disappears.
mixed	Subsystem appears in its own window.	Parent window rises to front. Subsystem window disappears.

## Labeling Subsystem Ports

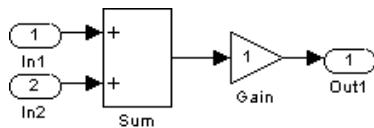
Simulink labels ports on a Subsystem block. The labels are the names of Inport and Outport blocks that connect the subsystem to blocks outside the subsystem through these ports.

You can hide (or show) the port labels by

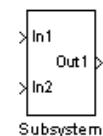
- Selecting the Subsystem block, then choosing **Hide Port Labels** (or **Show Port Labels**) from the **Format** menu

- Selecting an Import or Outport block in the subsystem and choosing **Hide Name** (or **Show Name**) from the **Format** menu
- Selecting the **Show port labels** option in the Subsystem block's parameter dialog

This figure shows two models. The subsystem on the left contains two Import blocks and one Outport block. The Subsystem block on the right shows the labeled ports.



Subsystem with Import and Outport blocks



Subsystem with labeled ports

## Creating Conditionally Executed Subsystems

A *conditionally executed subsystem* is a subsystem whose execution depends on the value of an input signal. The signal that controls whether a subsystem executes is called the *control signal*. The signal enters the Subsystem block at the *control input*.

Conditionally executed subsystems can be very useful when you are building complex models that contain components whose execution depends on other components.

Simulink supports the following types of conditionally executed subsystems:

- An *enabled subsystem* executes while the control signal is positive. It starts execution at the time step where the control signal crosses zero (from the negative to the positive direction) and continues execution while the control signal remains positive. Enabled subsystems are described in more detail in “Enabled Subsystems” on page 4-30.
- A *triggered subsystem* executes once each time a trigger event occurs. A trigger event can occur on the rising or falling edge of a trigger signal, which can be continuous or discrete. Triggered subsystems are described in more detail in “Triggered Subsystems” on page 4-34.
- A *triggered and enabled subsystem* executes once on the time step when a trigger event occurs if the enable control signal has a positive value at that step. See “Triggered and Enabled Subsystems” on page 4-37 for more information.
- A *control flow subsystem* executes one or more times at the current time step when enabled by a control flow block that implements control logic similar to that expressed by programming language control flow statements (e.g., *if-then*, *while*, *do*, and *for*). See “Modeling with Control Flow Blocks” in the online documentation for more information.

---

**Note** Simulink displays an error if you connect a Constant, Model, or S-Function block with constant sample time (see “Constant Sample Time” on page 2-39) to the output port of a conditionally executed subsystem. To avoid the error, either change the sample time of the block to a nonconstant sample time or insert a Signal Conversion block between the block with constant sample time and the output port.

---

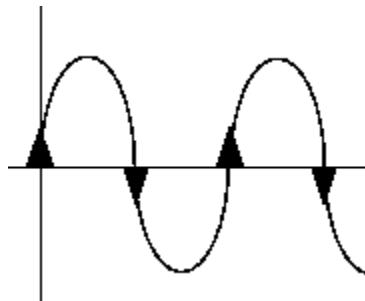
## Enabled Subsystems

Enabled subsystems are subsystems that execute at each simulation step where the control signal has a positive value.

An enabled subsystem has a single control input, which can be scalar or vector valued.

- If the input is a scalar, the subsystem executes if the input value is greater than zero.
- If the input is a vector, the subsystem executes if *any* of the vector elements is greater than zero.

For example, if the control input signal is a sine wave, the subsystem is alternately enabled and disabled, as shown in this figure. An up arrow signifies enable, a down arrow disable.



Simulink uses the zero-crossing slope method to determine whether an enable is to occur. If the signal crosses zero and the slope is positive, the subsystem

is enabled. If the slope is negative at the zero crossing, the subsystem is disabled.

## Creating an Enabled Subsystem

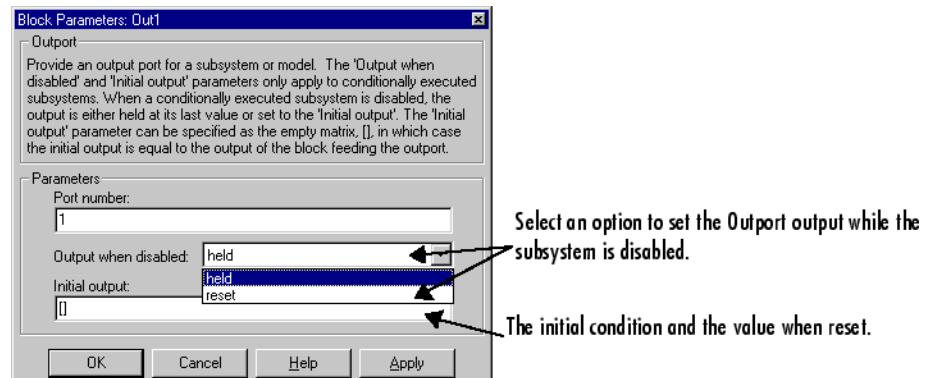
You create an enabled subsystem by copying an Enable block from the Ports & Subsystems library into a subsystem. Simulink adds an enable symbol and an enable control input port to the Subsystem block.



**Setting Output Values While the Subsystem Is Disabled.** Although an enabled subsystem does not execute while it is disabled, the output signal is still available to other blocks. While an enabled subsystem is disabled, you can choose to hold the subsystem outputs at their previous values or reset them to their initial conditions.

Open each Outport block's dialog box and select one of the choices for the **Output when disabled** parameter, as shown in the following dialog box:

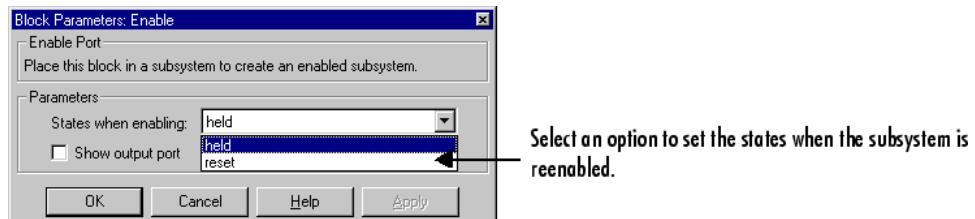
- Choose **hold** to cause the output to maintain its most recent value.
- Choose **reset** to cause the output to revert to its initial condition. Set the **Initial output** to the initial value of the output.



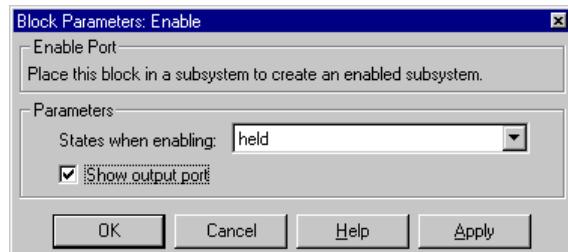
**Setting States When the Subsystem Becomes Reenabled.** When an enabled subsystem executes, you can choose whether to hold the subsystem states at their previous values or reset them to their initial conditions.

To do this, open the Enable block dialog box and select one of the choices for the **States when enabling** parameter, as shown in the dialog box following:

- Choose **held** to cause the states to maintain their most recent values.
- Choose **reset** to cause the states to revert to their initial conditions.



**Outputting the Enable Control Signal.** An option on the Enable block dialog box lets you output the enable control signal. To output the control signal, select the **Show output port** check box.



This feature allows you to pass the control signal down into the enabled subsystem, which can be useful where logic within the enabled subsystem is dependent on the value or values contained in the control signal.

### Blocks an Enabled Subsystem Can Contain

An enabled subsystem can contain any block, whether continuous or discrete. Discrete blocks in an enabled subsystem execute only when the subsystem executes, and only when their sample times are synchronized with the

simulation sample time. Enabled subsystems and the model use a common clock.

---

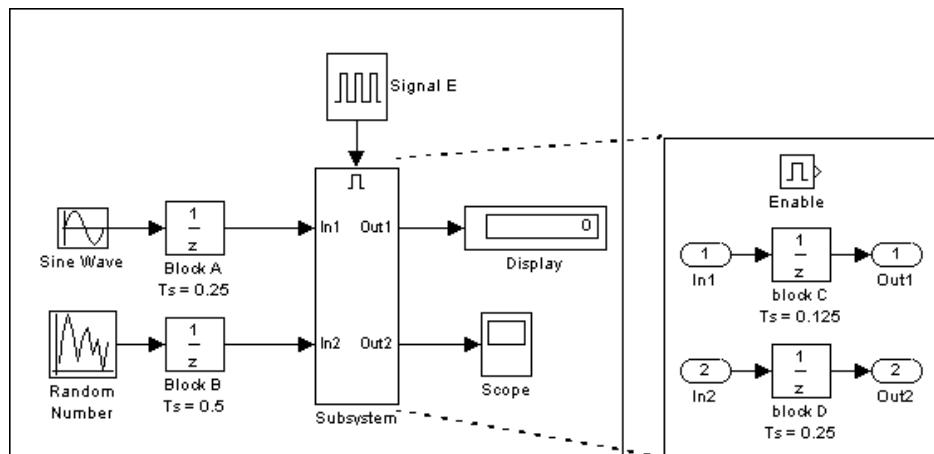
**Note** Enabled subsystems can contain Goto blocks. However, only state ports can connect to Goto blocks in an enabled subsystem. See the Simulink demo model, clutch, for an example of how to use Goto blocks in an enabled subsystem.

---

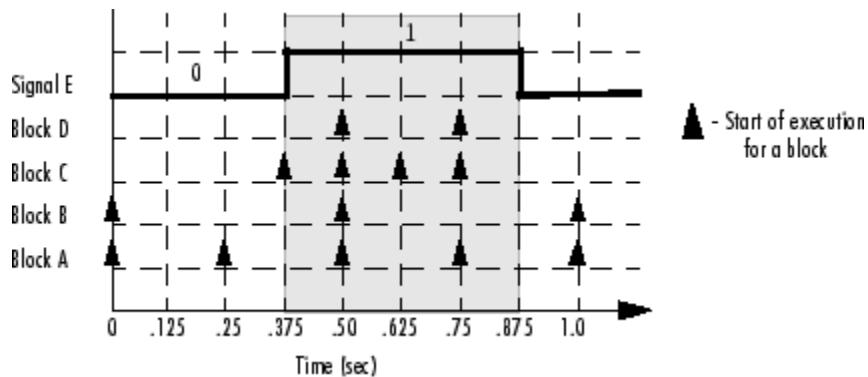
For example, this system contains four discrete blocks and a control signal. The discrete blocks are

- Block A, which has a sample time of 0.25 second
- Block B, which has a sample time of 0.5 second
- Block C, within the enabled subsystem, which has a sample time of 0.125 second
- Block D, also within the enabled subsystem, which has a sample time of 0.25 second

The enable control signal is generated by a Pulse Generator block, labeled Signal E, which changes from 0 to 1 at 0.375 second and returns to 0 at 0.875 second.



The chart below indicates when the discrete blocks execute.



Blocks A and B execute independently of the enable control signal because they are not part of the enabled subsystem. When the enable control signal becomes positive, blocks C and D execute at their assigned sample rates until the enable control signal becomes zero again. Note that block C does not execute at 0.875 second when the enable control signal changes to zero.

## Triggered Subsystems

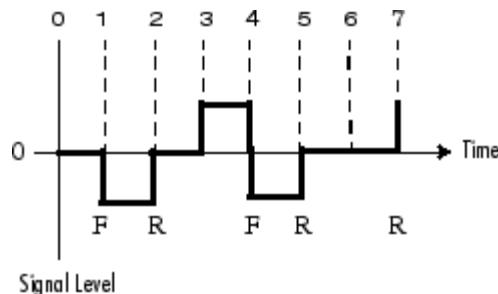
Triggered subsystems are subsystems that execute each time a trigger event occurs.

A triggered subsystem has a single control input, called the *trigger input*, that determines whether the subsystem executes. You can choose from three types of trigger events to force a triggered subsystem to begin execution:

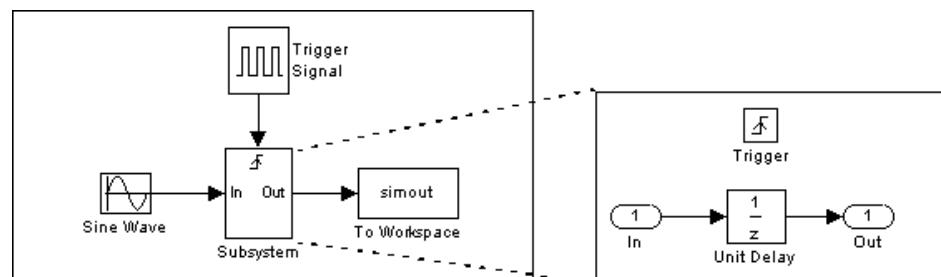
- **rising** triggers execution of the subsystem when the control signal rises from a negative or zero value to a positive value (or zero if the initial value is negative).
- **falling** triggers execution of the subsystem when the control signal falls from a positive or a zero value to a negative value (or zero if the initial value is positive).
- **either** triggers execution of the subsystem when the signal is either rising or falling.

**Note** In the case of discrete systems, a signal's rising or falling from zero is considered a trigger event only if the signal has remained at zero for more than one time step preceding the rise or fall. This eliminates false triggers caused by control signal sampling.

For example, in the following timing diagram for a discrete system, a rising trigger (R) does not occur at time step 3 because the signal has remained at zero for only one time step when the rise occurs.



A simple example of a triggered subsystem is illustrated.



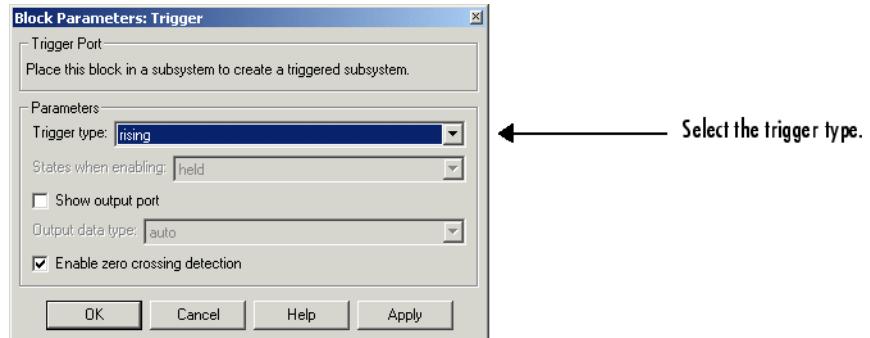
In this example, the subsystem is triggered on the rising edge of the square wave trigger control signal.

## Creating a Triggered Subsystem

You create a triggered subsystem by copying the Trigger block from the Ports & Subsystems library into a subsystem. Simulink adds a trigger symbol and a trigger control input port to the Subsystem block.



To select the trigger type, open the Trigger block dialog box and select one of the choices for the **Trigger type** parameter, as shown in the following dialog box:

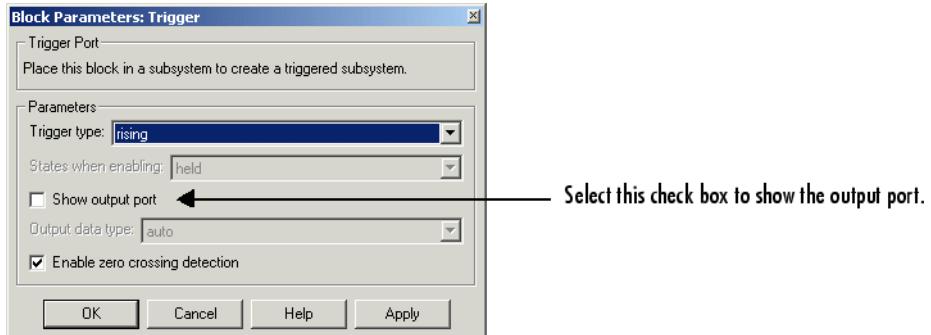


Simulink uses different symbols on the Trigger and Subsystem blocks to indicate rising and falling triggers (or either). This figure shows the trigger symbols on Subsystem blocks.



**Outputs and States Between Trigger Events.** Unlike enabled subsystems, triggered subsystems always hold their outputs at the last value between triggering events. Also, triggered subsystems cannot reset their states when triggered; states of any discrete blocks are held between trigger events.

**Outputting the Trigger Control Signal.** An option on the Trigger block dialog box lets you output the trigger control signal. To output the control signal, select the **Show output port** check box.



The **Output data type** field allows you to specify the data type of the output signal as `auto`, `int8`, or `double`. The `auto` option causes the data type of the output signal to be set to the data type (either `int8` or `double`) of the port to which the signal is connected.

## Function-Call Subsystems

You can use a Trigger block to create a subsystem whose execution is determined by logic internal to an S-function instead of by the value of a signal. These subsystems are called *function-call subsystems*. For more information about function-call subsystems, see “Function-Call Subsystems” in “Writing S-Functions” in the online Simulink documentation.

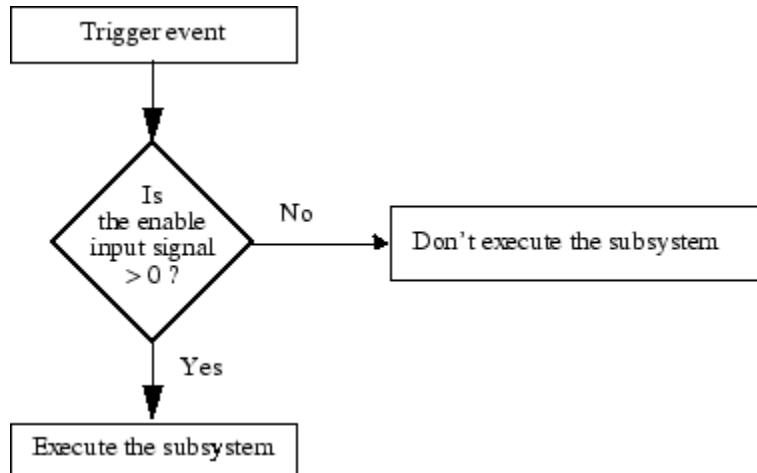
## Blocks That a Triggered Subsystem Can Contain

All blocks in a triggered systems must have either inherited (-1) or constant (`inf`) sample time. This is to indicate that the blocks in the triggered subsystem run only when the triggered subsystem itself runs, i.e., when it is triggered. This requirement means that a triggered subsystem cannot contain continuous blocks, such as the Integrator block.

## Triggered and Enabled Subsystems

A third kind of conditionally executed subsystem combines both types of conditional execution. The behavior of this type of subsystem, called a

*triggered and enabled* subsystem, is a combination of the enabled subsystem and the triggered subsystem, as shown by this flow diagram.



A triggered and enabled subsystem contains both an enable input port and a trigger input port. When the trigger event occurs, Simulink checks the enable input port to evaluate the enable control signal. If its value is greater than zero, Simulink executes the subsystem. If both inputs are vectors, the subsystem executes if at least one element of each vector is nonzero.

The subsystem executes once at the time step at which the trigger event occurs.

### Creating a Triggered and Enabled Subsystem

You create a triggered and enabled subsystem by dragging both the Enable and Trigger blocks from the Ports & Subsystems library into an existing subsystem. Simulink adds enable and trigger symbols and enable and trigger and enable control inputs to the Subsystem block.

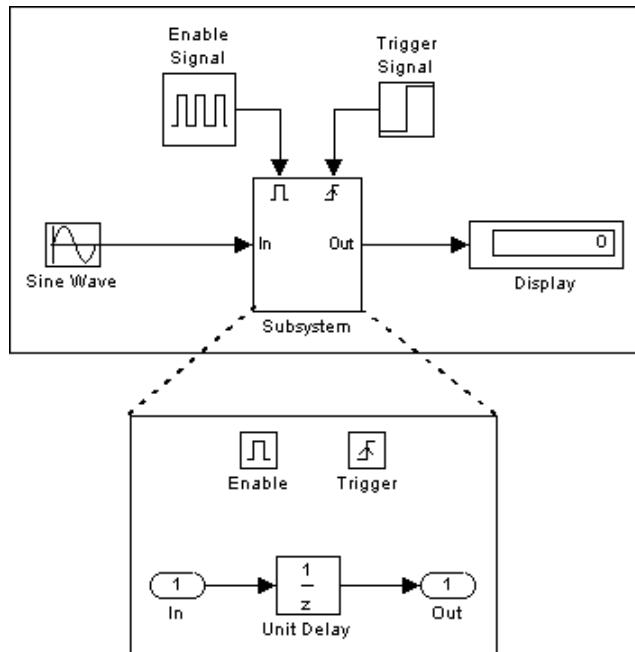


You can set output values when a triggered and enabled subsystem is disabled as you would for an enabled subsystem. For more information, see “Setting Output Values While the Subsystem Is Disabled” on page 4-31. Also, you can specify what the values of the states are when the subsystem is reenabled. See “Setting States When the Subsystem Becomes Reenabled” on page 4-32.

Set the parameters for the Enable and Trigger blocks separately. The procedures are the same as those described for the individual blocks.

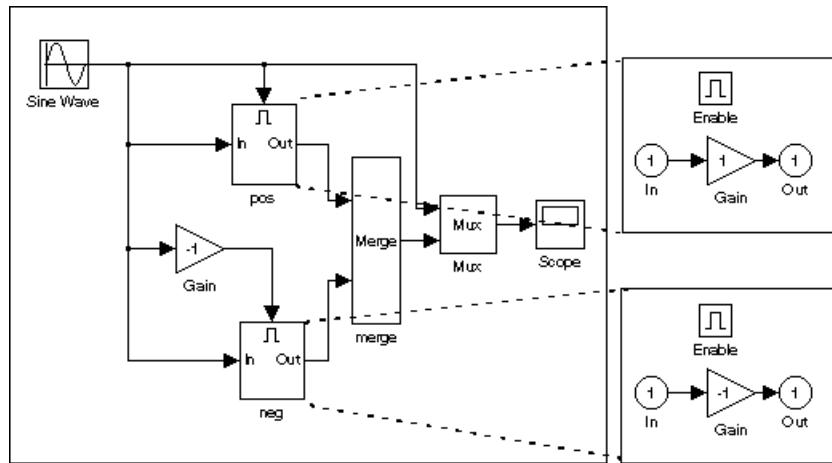
### A Sample Triggered and Enabled Subsystem

A simple example of a triggered and enabled subsystem is illustrated in the model below.

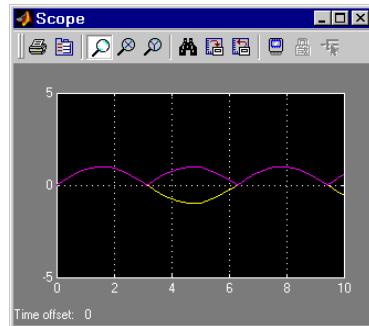


## Creating Alternately Executing Subsystems

You can use conditionally executed subsystems in combination with Merge blocks to create sets of subsystems that execute alternately, depending on the current state of the model. The following figure shows a model that uses two enabled blocks and a Merge block to model a full-wave rectifier – a device that converts AC current to pulsating DC current.



The block labeled “pos” is enabled when the AC waveform is positive; it passes the waveform unchanged to its output. The block labeled “neg” is enabled when the waveform is negative; it inverts the waveform. The Merge block passes the output of the currently enabled block to the Mux block, which passes the output, along with the original waveform, to the Scope block. The Scope creates the following display.



# Using Callback Functions

You can define MATLAB expressions that execute when the block diagram or a block is acted upon in a particular way. These expressions, called *callback functions*, are specified by block, port, or model parameters. For example, the function specified by a block's OpenFcn parameter is executed when you double-click that block's name or its path changes.

## Tracing Callbacks

Callback tracing allows you to determine the callbacks Simulink invokes and in what order Simulink invokes them when you open or simulate a model. To enable callback tracing, select the **Callback tracing** option on the Simulink **Preferences** dialog box or execute `set_param(0, 'CallbackTracing', 'on')`. This option causes Simulink to list callbacks in the MATLAB Command Window as they are invoked.

## Creating Model Callback Functions

You can create model callback functions interactively or programmatically. Use the **Callbacks** pane of the model's **Model Properties** dialog box (see "Callbacks Pane" in the online Simulink documentation) to create model callbacks interactively. To create a callback programmatically, use the `set_param` command to assign a MATLAB expression that implements the function to the model parameter corresponding to the callback (see "Model Callback Functions" on page 4-41).

For example, this command evaluates the variable `testvar` when the user double-clicks the Test block in `mymodel`:

```
set_param('mymodel/Test', 'OpenFcn', testvar)
```

You can examine the clutch system (`clutch.mdl`) for routines associated with many model callbacks.

## Model Callback Functions

The following table describes callback functions associated with models.

Parameter	When Executed
CloseFcn	Before the block diagram is closed.
PostLoadFcn	After the model is loaded. Defining a callback routine for this parameter might be useful for generating an interface that requires that the model has already been loaded.
InitFcn	Called at start of model simulation.
PostSaveFcn	After the model is saved.
PreLoadFcn	Before the model is loaded. Defining a callback routine for this parameter might be useful for loading variables used by the model.
PreSaveFcn	Before the model is saved.
StartFcn	Before the simulation starts.
StopFcn	After the simulation stops. Output is written to workspace variables and files before the StopFcn is executed.

---

**Note** Beware of adverse interactions between callback functions of models referenced by other models. For example, suppose that model A references model B and that model A's OpenFcn creates variables in the MATLAB workspace and model B's CloseFcn clears the MATLAB workspace. Now suppose that simulating model A requires rebuilding model B. Rebuilding B entails opening and closing model B and hence invoking model B's CloseFcn, which clears the MATLAB workspace, including the variables created by A's OpenFcn.

---

## Creating Block Callback Functions

You can create model callback functions interactively or programmatically. Use the **Callbacks** pane of the model's **Block Properties** dialog box (see “Callbacks Pane” on page 5-13) to create model callbacks interactively. To create a callback programmatically, use the `set_param` command to assign a MATLAB expression that implements the function to the block parameter corresponding to the callback (see “Block Callback Parameters” on page 4-43).

---

**Note** A callback for a masked subsystem cannot directly reference the parameters of the masked subsystem (see “About Masks” on page 9-2). The reason? Simulink evaluates block callbacks in a model’s base workspace whereas the mask parameters reside in the masked subsystem’s private workspace. A block callback, however, can use `get_param` to obtain the value of a mask parameter, e.g., `get_param(gcb, 'gain')`, where `gain` is the name of a mask parameter of the current block.

---

## Block Callback Parameters

This table lists the parameters for which you can define block callback routines, and indicates when those callback routines are executed. Routines that are executed before or after actions take place occur immediately before or after the action.

Parameter	When Executed
<code>ClipboardFcn</code>	When the block is copied or cut to the system clipboard.
<code>CloseFcn</code>	When the block is closed using the <code>close_system</code> command.
<code>CopyFcn</code>	After a block is copied. The callback is recursive for Subsystem blocks (that is, if you copy a Subsystem block that contains a block for which the <code>CopyFcn</code> parameter is defined, the routine is also executed). The routine is also executed if an <code>add_block</code> command is used to copy the block.
<code>DeleteChildFcn</code>	After a block is deleted from a subsystem.
<code>DeleteFcn</code>	Before a block is deleted, e.g., when the user deletes the block or closes the model containing the block. This callback is recursive for Subsystem blocks.
<code>DestroyFcn</code>	When the block has been destroyed.
<code>InitFcn</code>	Before the block diagram is compiled and before block parameters are evaluated.

Parameter	When Executed
ErrorFcn	<p>When an error has occurred in a subsystem. The callback function should have the following form:</p> <pre>errorMsg = errorHandler(subsys, errorType)</pre> <p>where <code>errorHandler</code> is the name of the callback function, <code>subsys</code> is a handle to the subsystem in which the error occurred, <code>errorType</code> is a string that indicates the type of error that occurred, and <code>errorMsg</code> is a string specifying the text of an error message to be displayed to the user. Simulink displays the error message returned by the callback function.</p>
LoadFcn	After the block diagram is loaded. This callback is recursive for Subsystem blocks.
ModelCloseFcn	Before the block diagram is closed. This callback is recursive for Subsystem blocks.
MoveFcn	When the block is moved or resized.
NameChangeFcn	After a block's name and/or path changes. When a Subsystem block's path is changed, it recursively calls this function for all blocks it contains after calling its own <code>NameChangeFcn</code> routine.
OpenFcn	When the block is opened. This parameter is generally used with Subsystem blocks. The routine is executed when you double-click the block or when an <code>open_system</code> command is called with the block as an argument. The <code>OpenFcn</code> parameter overrides the normal behavior associated with opening a block, which is to display the block's dialog box or to open the subsystem.
ParentCloseFcn	Before closing a subsystem containing the block or when the block is made part of a new subsystem using the <code>new_system</code> command (see <code>new_system</code> in the online Simulink reference).
PreSaveFcn	Before the block diagram is saved. This callback is recursive for Subsystem blocks.

Parameter	When Executed
PostSaveFcn	After the block diagram is saved. This callback is recursive for Subsystem blocks.
StartFcn	After the block diagram is compiled and before the simulation starts. In the case of an S-Function block, StartFcn executes immediately before the first execution of the block's <code>mdlProcessParameters</code> function. See "S-Function Callback Methods" in the online Simulink documentation for more information.
StopFcn	At any termination of the simulation. In the case of an S-Function block, StopFcn executes after the block's <code>mdlTerminate</code> function executes. See "S-Function Callback Methods" in the online Simulink documentation for more information.
UndoDeleteFcn	When a block delete is undone.

## Port Callback Parameters

Block input and output ports have a single callback parameter, `ConnectionCallback`. This parameter allows you to set callbacks on ports that are triggered every time the connectivity of those ports changes. Examples of connectivity changes include deletion of blocks connected to the port and deletion, disconnection, or connection of branches or lines to the port.

Use `get_param` to get the port handle of a port and `set_param` to set the callback on the port. For example, suppose the currently selected block has a single input port. The following code fragment sets `foo` as the connection callback on the input port.

```
phs = get_param(gcb, 'PortHandles');
set_param(phs.Inport, 'ConnectionCallback', 'foo');
```

The first argument of the callback function must be a port handle. The callback function can have other arguments (and a return value) as well. For example, the following is a valid callback function signature.

```
function foo(port, otherArg1, otherArg2)
```



# Working with Blocks

---

This section explores the following block-related topics.

**About Blocks (p. 5-2)**

Explains the difference between virtual and nonvirtual blocks.

**Editing Blocks (p. 5-4)**

How to cut and paste blocks.

**Working with Block Parameters (p. 5-7)**

How to set parameters that determine a block's behavior.

**Changing a Block's Appearance (p. 5-15)**

How to change the size, orientation, color, and labeling of a block.

**Displaying Block Outputs (p. 5-19)**

How to display the values of block outputs on the block diagram during simulation.

**Working with Block Libraries (p. 5-21)**

How to create and use block libraries.

## About Blocks

Blocks are the elements from which Simulink models are built. You can model virtually any dynamic system by creating and interconnecting blocks in appropriate ways. This section discusses how to use blocks to build models of dynamic systems.

### Block Data Tips

On Microsoft Windows, Simulink displays information about a block in a pop-up window when you allow the pointer to hover over the block in the diagram view. To disable this feature or control what information a data tip includes, select **Block data tips options** from the Simulink **View** menu.

### Virtual Blocks

When creating models, you need to be aware that Simulink blocks fall into two basic categories: nonvirtual and virtual blocks. Nonvirtual blocks play an active role in the simulation of a system. If you add or remove a nonvirtual block, you change the model's behavior. Virtual blocks, by contrast, play no active role in the simulation; they help organize a model graphically. Some Simulink blocks are virtual in some circumstances and nonvirtual in others. Such blocks are called conditionally virtual blocks. The following table lists Simulink virtual and conditionally virtual blocks.

Block Name	Condition Under Which Block Is Virtual
Bus Selector	Virtual if input bus is virtual.
Demux	Always virtual.
Enable	Virtual unless connected directly to an Outport block.
From	Always virtual.
Goto	Always virtual.
Goto Tag Visibility	Always virtual.
Ground	Always virtual.

Block Name	Condition Under Which Block Is Virtual
Import	Virtual <i>unless</i> the block resides in a conditionally executed or atomic subsystem <i>and</i> has a direct connection to an Outport block.
Mux	Always virtual.
Outport	Virtual when the block resides within any subsystem block (conditional or not), and does <i>not</i> reside in the root (top-level) Simulink window.
Selector	Virtual except in matrix mode.
Signal Specification	Always virtual.
Subsystem	Virtual unless the block is conditionally executed and/or the block's <b>Treat as Atomic Unit</b> option is selected.
Terminator	Always virtual.
Trigger	Virtual when the Outport port is <i>not</i> present.

## Editing Blocks

The Simulink Editor allows you to cut and paste blocks in and between models.

### Copying and Moving Blocks from One Window to Another

As you build your model, you often copy blocks from Simulink block libraries or other libraries or models into your model window. To do this:

- 1** Open the appropriate block library or model window.
- 2** Drag the block to copy into the target model window. To drag a block, position the cursor over the block, then press and hold down the mouse button. Move the cursor into the target window, then release the mouse button.

You can also drag blocks from the Simulink Library Browser into a model window. See “Browsing Block Libraries” on page 5-28 for more information.

---

**Note** Simulink hides the names of Sum, Mux, Demux, Bus Creator, and Bus Selector blocks when you copy them from the Simulink block library to a model. This is done to avoid unnecessarily cluttering the model diagram. (The shapes of these blocks clearly indicate their respective functions.)

---

You can also copy blocks by using the **Copy** and **Paste** commands from the **Edit** menu:

- 1** Select the block you want to copy.
- 2** Choose **Copy** from the **Edit** menu.
- 3** Make the target model window the active window.
- 4** Choose **Paste** from the **Edit** menu.

Simulink assigns a name to each copied block. If it is the first block of its type in the model, its name is the same as its name in the source window. For

example, if you copy the Gain block from the Math library into your model window, the name of the new block is Gain. If your model already contains a block named Gain, Simulink adds a sequence number to the block name (for example, Gain1, Gain2). You can rename blocks; see “Manipulating Block Names” on page 5-17.

When you copy a block, the new block inherits all the original block’s parameter values.

Simulink uses an invisible five-pixel grid to simplify the alignment of blocks. All blocks within a model snap to a line on the grid. You can move a block slightly up, down, left, or right by selecting the block and pressing the arrow keys.

You can display the grid in the model window by typing the following command in the MATLAB window.

```
set_param('<model name>', 'showgrid', 'on')
```

To change the grid spacing, enter

```
set_param('<model name>', 'gridspacing', <number of pixels>)
```

For example, to change the grid spacing to 20 pixels, enter

```
set_param('<model name>', 'gridspacing', 20)
```

For either of the above commands, you can also select the model, then enter `gcs` instead of `<model name>`.

You can copy or move blocks to compatible applications (such as word processing programs) using the **Copy**, **Cut**, and **Paste** commands. These commands copy only the graphic representation of the blocks, not their parameters.

Moving blocks from one window to another is similar to copying blocks, except that you hold down the **Shift** key while you select the blocks.

You can use the **Undo** command from the **Edit** menu to remove an added block.

## Moving Blocks in a Model

To move a single block from one place to another in a model window, drag the block to a new location. Simulink automatically repositions lines connected to the moved block.

To move more than one block, including connecting lines:

- 1** Select the blocks and lines. If you need information about how to select more than one block, see “Selecting Multiple Objects” on page 4-4.
- 2** Drag the objects to their new location and release the mouse button.

## Copying Blocks in a Model

You can copy blocks in a model as follows. While holding down the **Ctrl** key, select the block with the left mouse button, then drag it to a new location. You can also do this by dragging the block using the right mouse button. Duplicated blocks have the same parameter values as the original blocks. Sequence numbers are added to the new block names.

## Deleting Blocks

To delete one or more blocks, select the blocks to be deleted and press the **Delete** or **Backspace** key. You can also choose **Clear** or **Cut** from the **Edit** menu. The **Cut** command writes the blocks into the clipboard, which enables you to paste them into a model. Using the **Delete** or **Backspace** key or the **Clear** command does not enable you to paste the block later.

You can use the **Undo** command from the **Edit** menu to replace a deleted block.

# Working with Block Parameters

Every Simulink block has a set of attributes, called parameters or properties, that govern its appearance and its behavior during simulation. Some types of attributes are common to all blocks. For example, all blocks have a block name attribute. Other attributes are specific to a particular type of block. For example, only Gain blocks have a Gain parameter. Simulink allows you to specify values for many of a block's attributes, thus enabling you to customize the block's appearance or behavior to fit the requirements of a particular application.

Simulink provides the following means for specifying block parameters:

- **Format** menu

The Model Editor's **Format** menu allows you to specify attributes of the currently selected block that are visible on the model's block diagram, such as the block's name and color (see "Changing a Block's Appearance" on page 5-15 for more information).

- **Block Properties** dialog box

Specifies various attributes that are common to all blocks (see "Block Properties Dialog Box" on page 5-10 for more information).

- **Block Parameter** dialog box

Every block has a dialog box that allows you to specify values for attributes that are specific to that type of block. See "Displaying a Block's Parameter Dialog Box" on page 5-8 for information on displaying a block's parameter dialog box. For information on the parameter dialog of a specific block, see "Simulink Blocks" in the online Simulink reference.

- **Model Explorer**

The Model Explorer allows you to quickly find one or more blocks and set their properties, thus facilitating global changes to a model, for example, changing the gain of all of a model's Gain blocks. See "The Model Explorer" on page 8-2 for more information.

- `set_param` command

`set_param` enables you to use M-file programs and scripts to specify block attributes. See `set_param` in the online Simulink reference for more information.

### Displaying a Block's Parameter Dialog Box

You can also display a block's parameter dialog box by double-clicking it in the model or library window.

---

**Note** This holds true for all blocks with parameter dialog boxes except for Subsystem blocks. You must use the Model Editor's **Edit** menu or context menu to display a Subsystem block's parameter dialog box.

---

You can also display a block's parameter dialog box by selecting the block in the model's block diagram and choosing **BLOCK Parameters** from the model window's **Edit** menu or from the model window's context (right-click) menu, where **BLOCK** is the name of the block you selected, e.g., **Constant Parameters**.

### Specifying Parameter Values

You can use a MATLAB constant, model or base workspace variable, or expression that evaluates to a numerical value to specify the value of a numeric parameter.

### Working with Tunable Parameters

Simulink lets you change the values of many block parameters during simulation. Such parameters are called tunable parameters. In general, only parameters that represent mathematical variables, such as the Gain parameter of the Gain block, are tunable. Parameters that specify the appearance or structure of a block, e.g., the number of inputs of a Sum block, or when it is evaluated, e.g., a block's sample time or priority, are not tunable. You can tell whether a particular parameter is tunable by examining its edit control in the block's dialog box or Model Explorer during simulation. If the control is disabled, the parameter is nontunable.

## Tuning Block Parameters

You can use a block's dialog box or the Model Explorer to modify the tunable parameters of any block, except a source block (see “Changing Source Block Parameters” on page 5-9). You can also use the MATLAB Command Line to tune block parameters.

To use the block's parameter dialog box, open the block's parameter dialog box, change the value displayed in the dialog box, and click the dialog box's **OK** or **Apply** button. You can use the `set_param` command to change the value of a block parameter at the MATLAB Command Line during simulation. Or, if the model uses a MATLAB workspace variable to specify the parameter's value, you can change the parameter's value by assigning a new value to the variable. In either case, you must update the model's block diagram for the change to take effect (see “Updating a Block Diagram” on page 3-9).

## Changing Source Block Parameters

Opening the dialog box of a source block with tunable parameters (see “Source Blocks with Tunable Parameters” on page 5-10) causes a running simulation to pause. While the simulation is paused, you can edit the parameter values displayed on the dialog box. However, you must close the dialog box to have the changes take effect and allow the simulation to continue. Similarly, starting a simulation causes any open dialog boxes associated with source blocks with tunable parameters to close.

---

**Note** If you enable the **Inline parameters** option, Simulink does not pause the simulation when you open a source block's dialog box because all of the parameter fields are disabled and can be viewed but cannot be changed.

---

The Model Explorer disables the parameter fields that it displays in the list view and the dialog pane for a source block with tunable parameters while a simulation is running. As a result, you cannot use the Model Explorer to change the block's parameters. However, while the simulation is running, the Model Explorer displays a **Modify** button in the dialog view for the block. Clicking the **Modify** button opens the block's dialog box. Note that this causes the simulation to pause. You can then change the block's parameters. You must close the dialog box to have the changes take effect and allow the

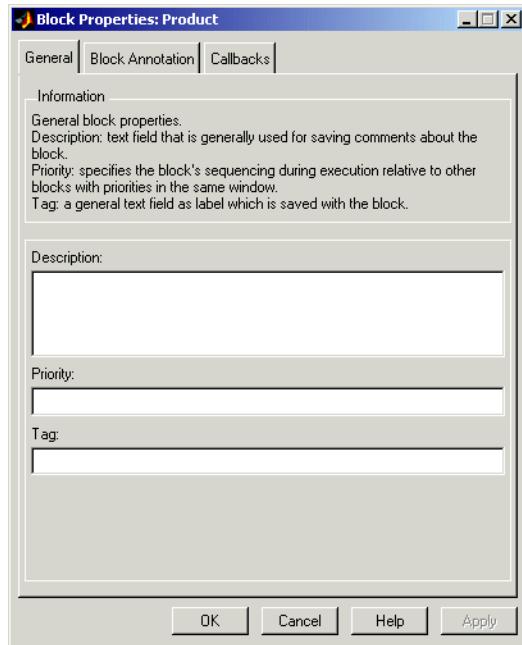
simulation to continue. Your changes appear in the Model Explorer after you close the dialog box.

**Source Blocks with Tunable Parameters.** Source blocks with tunable parameters include the following blocks.

- Simulink source blocks, including
  - Band-Limited White Noise
  - Chirp Signal
  - Constant
  - Pulse Generator
  - Ramp
  - Random Number
  - Repeating Sequence
  - Signal Generator
  - Sine Wave
  - Step
  - Uniform Random Number
- User-developed masked subsystem blocks that have one or more tunable parameters and one or more output ports, but no input ports.
- S-Function and M-file (level 2) S-Function blocks that have one or more tunable parameters and one or more output ports but no input ports.

## **Block Properties Dialog Box**

This dialog box lets you set a block's properties. To display this dialog, select the block in the model window and then select **Block Properties** from the **Edit** menu.



The dialog box contains the following tabbed panes.

### General Pane

This pane allows you to set the following properties.

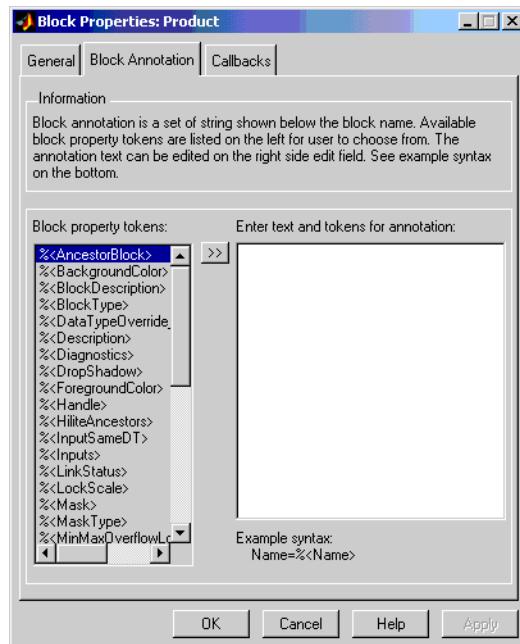
**Description.** Brief description of the block's purpose.

**Priority.** Execution priority of this block relative to other blocks in the model. See "Assigning Block Priorities" in the online Simulink documentation for more information.

**Tag.** Text that is assigned to the block's Tag parameter and saved with the block in the model. You can use the tag to create your own block-specific label for a block.

### Block Annotation Pane

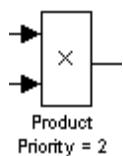
The block annotation pane allows you to display the values of selected parameters of a block in an annotation that appears beneath the block's icon.



Enter the text of the annotation in the text field that appears on the right side of the pane. The text can include block property tokens, for example

```
%<Name>
Priority = %<priority>
```

of the form %<param> where param is the name of a parameter of the block. When displaying the annotation, Simulink replaces the tokens with the values of the corresponding parameters, e.g.,

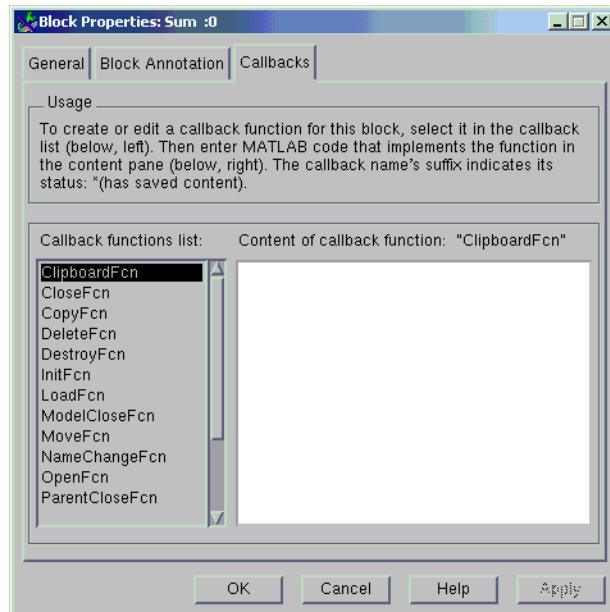


The block property tag list on the left side of the pane lists all the tags that are valid for the currently selected block. To include one of the listed tags in the annotation, select the tag and then click the button between the tag list and the annotation field.

You can also create block annotations programmatically. See “Creating Block Annotations Programmatically” on page 5-14.

## Callbacks Pane

The **Callbacks Pane** allows you to specify implementations for a block’s callbacks (see “Using Callback Functions” on page 4-41).



To specify an implementation for a callback, select the callback in the callback list on the left side of the pane. Then enter MATLAB commands that implement the callback in the right-hand field. Click **OK** or **Apply** to save the change. Simulink appends an asterisk to the name of the saved callback to indicate that it has been implemented.

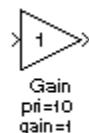
### Creating Block Annotations Programmatically

You can use a block's `AttributesFormatString` parameter to display selected parameters of a block beneath the block as an "attributes format string," i.e., a string that specifies values of the block's attributes (parameters). "Model and Block Parameters" in "Simulink Reference" describes the parameters that a block can have. Use the `Simulink.set_param` command to set this parameter to the desired attributes format string.

The attributes format string can be any text string that has embedded parameter names. An embedded parameter name is a parameter name preceded by %< and followed by >, for example, %<priority>. Simulink displays the attributes format string beneath the block's icon, replacing each parameter name with the corresponding parameter value. You can use line-feed characters (\n) to display each parameter on a separate line. For example, specifying the attributes format string

```
pri=%<priority>\ngain=%<Gain>
```

for a Gain block displays



If a parameter's value is not a string or an integer, Simulink displays N/S (not supported) for the parameter's value. If the parameter name is invalid, Simulink displays "???" as the parameter value.

## Changing a Block's Appearance

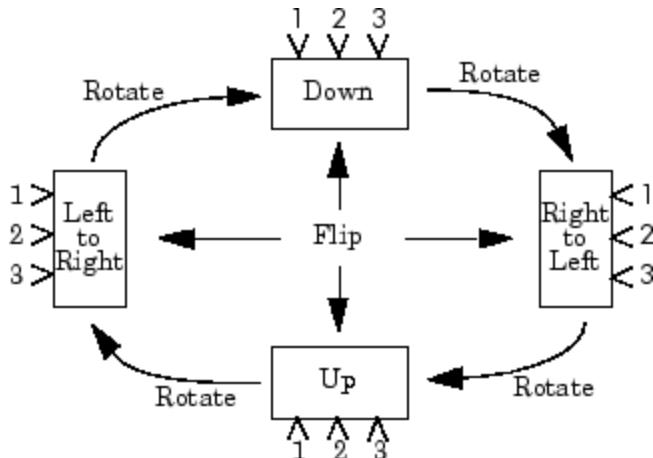
The Simulink Editor allows you to change the size, orientation, color, and label location of a block in a block diagram.

### Changing the Orientation of a Block

By default, signals flow through a block from left to right. Input ports are on the left, and output ports are on the right. You can change the orientation of a block by selecting one of these commands from the **Format** menu:

- The **Flip Block** command rotates the block 180 degrees.
- The **Rotate Block** command rotates a block clockwise 90 degrees.

The following figure shows how Simulink orders ports after changing the orientation of a block using the **Rotate Block** and **Flip Block** menu items. The text in the blocks shows their orientation.

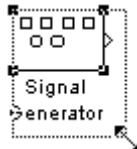


### Resizing a Block

To change the size of a block, select it, then drag any of its selection handles. While you hold down the mouse button, a dotted rectangle shows the new block size. When you release the mouse button, the block is resized.

For example, the following figure below shows a Signal Generator block being resized. The lower-right handle was selected and dragged to the cursor position. When the mouse button is released, the block takes its new size.

This figure shows a block being resized:



### Displaying Parameters Beneath a Block

You can cause Simulink to display one or more of a block's parameters beneath the block. You specify the parameters to be displayed in the following ways:

- By entering an attributes format string in the **Attributes format string** field of the block's **Block Properties** dialog box (see “Block Properties Dialog Box” on page 5-10)
- By setting the value of the block's `AttributesFormatString` property to the format string, using `set_param`

### Using Drop Shadows

You can add a drop shadow to a block by selecting the block, then choosing **Show Drop Shadow** from the **Format** menu. When you select a block with a drop shadow, the menu item changes to **Hide Drop Shadow**. The following figure shows a Subsystem block with a drop shadow:



## Manipulating Block Names

All block names in a model must be unique and must contain at least one character. By default, block names appear below blocks whose ports are on the sides, and to the left of blocks whose ports are on the top and bottom, as the following figure shows:




---

**Note** Simulink commands interprets a forward slash, i.e., /, as a block path delimiter. For example, the path vdp/Mu designates a block named Mu in the model named vdp. Therefore, avoid using forward slashes (/) in block names to avoid causing Simulink to interpret the names as paths.

---

## Changing Block Names

You can edit a block name in one of these ways:

- To replace the block name, click the block name, double-click or drag the cursor to select the entire name, then enter the new name.
- To insert characters, click between two characters to position the insertion point, then insert text.
- To replace characters, drag the mouse to select a range of text to replace, then enter the new text.

When you click the pointer anywhere else in the model or take any other action, the name is accepted or rejected. If you try to change the name of a block to a name that already exists or to a name with no characters, Simulink displays an error message.

You can modify the font used in a block name by selecting the block, then choosing the **Font** menu item from the **Format** menu. Select a font from the **Set Font** dialog box. This procedure also changes the font of any text that appears inside the block.

You can cancel edits to a block name by choosing **Undo** from the **Edit** menu.

---

**Note** If you change the name of a library block, all links to that block become unresolved.

---

### Changing the Location of a Block Name

You can change the location of the name of a selected block in two ways:

- By dragging the block name to the opposite side of the block.
- By choosing the **Flip Name** command from the **Format** menu. This command changes the location of the block name to the opposite side of the block.

For more information about block orientation, see “Changing the Orientation of a Block” on page 5-15.

### Changing Whether a Block Name Appears

To change whether the name of a selected block is displayed, choose a menu item from the **Format** menu:

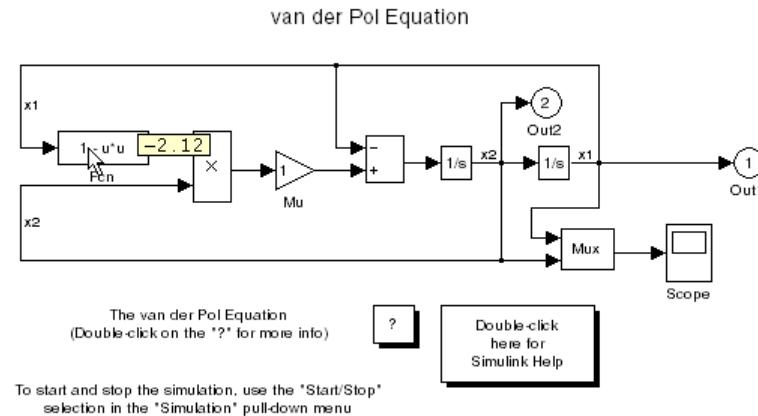
- The **Hide Name** menu item hides a visible block name. When you select **Hide Name**, it changes to **Show Name** when that block is selected.
- The **Show Name** menu item shows a hidden block name.

### Specifying a Block’s Color

See “Specifying Block Diagram Colors” on page 4-6 for information on how to set the color of a block.

# Displaying Block Outputs

Simulink can display block outputs as data tips on the block diagram while a simulation is running.



You can specify whether and when to display block outputs (see “Enabling Port Values Display” on page 5-19) and the size and format of the output displays and the rate at which Simulink updates them during a simulation (see “Port Values Display Options” on page 5-20).

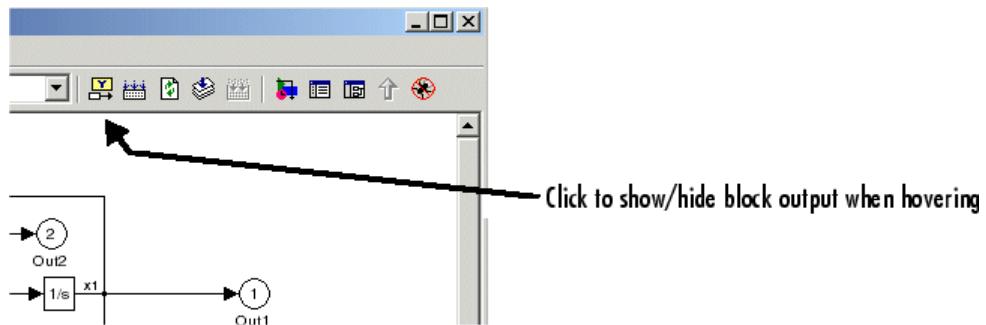
## Enabling Port Values Display

To turn display of port output values on or off, select **Port Values** from the Model Editor’s **View** menu. A menu of display options appears. Select one of the following display options from the menu:

- **Show none**  
Turns port value displaying off.
- **Show when hovering**  
Displays output port values for the block under the mouse cursor.
- **Toggle when selected**

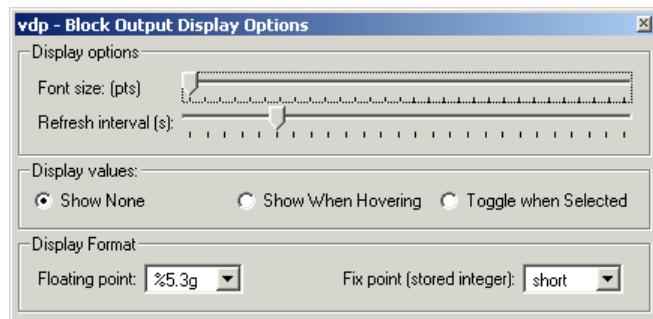
Selecting a block displays its outputs. Reselecting the block turns the display off.

When using the Microsoft Windows version of Simulink, you can turn block output display when hovering on or off from the Model Editor's toolbar. To do this, select the block output display button on the toolbar.



### Port Values Display Options

To specify other display options, select **Port Values > Options** from the Model Editor's **View** menu. The **Block Output Display Options** dialog box appears.



To increase the size of the output display text, move the **Font size** slider to the right. To increase the rate at which Simulink updates the displays, move the **Refresh interval** slider to the left.

# Working with Block Libraries

Libraries are collections of blocks that can be copied into models. Blocks copied from a library remain linked to their originals such that changes in the originals automatically propagate to the copies in a model. Libraries ensure that your models automatically include the most recent versions of blocks developed by yourself or others.

## Terminology

It is important to understand the terminology used with this feature.

*Library* - A collection of library blocks. A library must be explicitly created using **New Library** from the **File** menu.

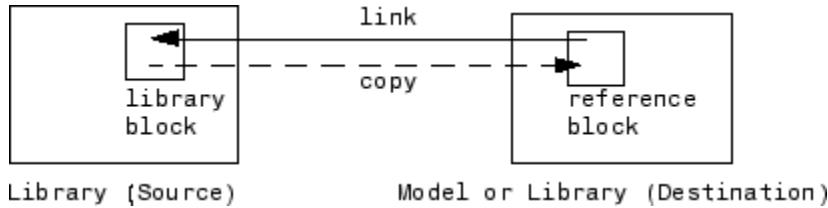
*Library block* - A block in a library.

*Reference block* - A copy of a library block.

*Link* - The connection between the reference block and its library block that allows Simulink to update the reference block when the library block changes.

*Copy* - The operation that creates a reference block from either a library block or another reference block.

This figure illustrates this terminology.



## Simulink Block Library

Simulink comes with a library of standard blocks called the Simulink block library. See “Starting Simulink” on page 3-2 for information on displaying and using this library.

## Creating a Library Link

To create a link to a library block in a model, copy the block from the library to the model (see “Copying and Moving Blocks from One Window to Another” on page 5-4) or by dragging the block from the Library Browser (see “Browsing Block Libraries” on page 5-28) into the model window.

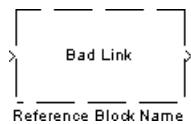
When you drag a library block into a model or another library, Simulink creates a copy of the library block, called the reference block, in the model or the other library. You can change the values of the reference block’s parameters but you cannot mask the block or, if it is masked, edit the mask. Also, you cannot set callback parameters for a reference block. If the link is to a subsystem, you can make nonstructural changes to the contents of the reference subsystem (see “Modifying a Linked Subsystem” on page 5-23).

The library and reference blocks are linked *by name*; that is, the reference block is linked to the specific block and library whose names are in effect at the time the copy is made.

If Simulink is unable to find either the library block or the source library on your MATLAB path when it attempts to update the reference block, the link becomes *unresolved*. Simulink issues an error message and displays these blocks using red dashed lines. The error message is

```
Failed to find block "source-block-name"
in library "source-library-name"
referenced by block
"reference-block-path".
```

The unresolved reference block is displayed like this (colored red).



To fix a bad link, you must do one of the following:

- Delete the unlinked reference block and copy the library block back into your model.

- Add the directory that contains the required library to the MATLAB path and select **Update Diagram** from the **Edit** menu.
- Double-click the reference block. On the dialog box that appears, correct the pathname and click **Apply** or **Close**.

## Disabling Library Links

Simulink allows you to disable linked blocks in a model. Simulink ignores disabled links when simulating a model. To disable a link, select the link, choose **Link options** from the model window's **Edit** or context menu, then choose **Disable link**. To restore a disabled link, choose **Restore link** from the **Link Options** menu.

## Modifying a Linked Subsystem

Simulink allows you to make any change to the local copy of an active library link that does not alter the structure of the local copy. Examples of nonstructural changes include changes to parameter values that do not affect the structure of the subsystem. Examples of structural modifications include adding or deleting a block or line or changing the number of ports on a block.

---

**Note** Simulink displays “parameterized link” on the parameter dialog box of a masked subsystem whose parameters differ in value from those of the library version to which it is linked. This allows you to determine whether the local copy differs from the library version simply by opening the local copy’s dialog box.

---

When you save the model containing the modified subsystem, Simulink saves the changes to the local copy of the subsystem together with the path to the library copy in the model’s model (.mdl) file. When you reopen the system, Simulink copies the library subsystem into the loaded model and applies the saved changes.

**Note** If you attempt to use the Simulink GUI to make a structural change to the local copy of an active library link, for example, by editing the subsystem's diagram, Simulink offers to disable the link to the subsystem. If you accept, Simulink makes the change. Otherwise, it does not allow you to make the structural change. Simulink does not prevent you from using `set_param` to attempt to make a structural change to an active link. However, the results of the change are undefined.

### Propagating Link Modifications

If you restore a disabled link that has structural changes, Simulink prompts you to either propagate or discard the changes. If you choose to propagate the changes, Simulink updates the library version of the subsystem specified by the restored link with the changes made in the model's version of that subsystem. If you choose to discard the changes, Simulink replaces the version of the subsystem in the model with the version in the library. In either case, the end result is that the versions of the subsystem in the library and the model are exactly the same.

If you restore a disabled link to a block with nonstructural changes, Simulink enables the link without prompting you to propagate or discard the changes. To see the nonstructural parameter differences between the model's version of a library block and the library block itself, choose **View changes** from the **Link options** menu.

If you want to propagate or discard nonstructural changes, select the modified copy of the library block in the model, choose **Link options** from the model window's **Edit** or context menu, then choose **Propagate/Discard changes**. A dialog box appears that asks whether you want to propagate or discard the changes. If you elect to propagate the changes, Simulink applies the changes made to the model's copy of the library block to the library block itself. If you elect to discard the changes, Simulink removes the changes from the model's copy of the block. In either case, the library and model versions of the block become the same.

## Updating a Linked Block

Simulink updates out-of-date reference blocks in a model or library at these times:

- When the model or library is loaded
- When you select **Update Diagram** from the **Edit** menu or run the simulation
- When you query the `LinkStatus` parameter of a block, using the `get_param` command (see “Library Link Status” on page 5-26)
- When you use the `find_system` command

## Breaking a Link to a Library Block

You can break the link between a reference block and its library block to cause the reference block to become a simple copy of the library block, unlinked to the library block. Changes to the library block no longer affect the block. Breaking links to library blocks may enable you to transport a model as a stand-alone model, without the libraries.

To break the link between a reference block and its library block, first disable the block. Then select the block and choose **Break Library Link** from the **Link options** menu. You can also break the link between a reference block and its library block from the command line by changing the value of the `LinkStatus` parameter to 'none' using this command:

```
set_param('refblock', 'LinkStatus', 'none')
```

You can save a system and break all links between reference blocks and library blocks using this command:

```
save_system('sys', 'newname', 'BreakLinks')
```

**Note** Breaking library links in a model does not guarantee that you can run the model stand-alone, especially if the model includes blocks from third-party libraries or optional Simulink blocksets. It is possible that a library block invokes functions supplied with the library and hence can run only if the library is installed on the system running the model. Further, breaking a link can cause a model to fail when you install a new version of the library on a system. For example, suppose a block invokes a function that is supplied with the library. Now suppose that a new version of the library eliminates the function. Running a model with an unlinked copy of the block results in invocation of a now nonexistent function, causing the simulation to fail. To avoid such problems, you should generally avoid breaking links to third-party libraries and optional Simulink blocksets.

## Finding the Library Block for a Reference Block

To find the source library and block linked to a reference block, select the reference block. Then choose **Go To Library Link** from the **Link options** submenu of the model window's **Edit** or context menu. If the library is open, Simulink selects and highlights the library block and makes the source library the active window. If the library is not open, Simulink opens it and selects the library block.

## Library Link Status

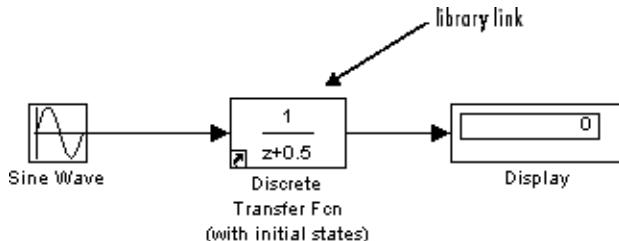
All blocks have a **LinkStatus** parameter that indicates whether the block is a reference block. The parameter can have these values.

Status	Description
none	Block is not a reference block.
resolved	Link is resolved.
unresolved	Link is unresolved.

<b>Status</b>	<b>Description</b>
implicit	Block resides in library block and is itself not a link to a library block. For example, suppose that A is a link to a subsystem in a library that contains a Gain block. Further, suppose that you open A and select the Gain block. Then, <code>get_param(gcb, 'LinkStatus')</code> returns <code>implicit</code> .
inactive	Link is disabled.
restore	Restores a broken link to a library block and discards any changes made to the local copy of the library block. For example, <code>set_param(gcb, 'LinkStatus', 'restore')</code> replaces the selected block with a link to a library block of the same type, discarding any changes in the local copy of the library block. Note that this parameter is a “write-only” parameter, i.e., it is usable only with <code>set_param</code> . You cannot use <code>get_param</code> to get it.
propagate	Restores a broken link to a library block and propagates any changes made to the local copy to the library.

## Displaying Library Links

Simulink optionally displays an arrow in the bottom left corner of each block that represents a library link in a model.



This arrow allows you to tell at a glance whether a block represents a link to a library block or a local instance of a block. To enable display of library links, select **Library Link Display** from the model window's **Format** menu.

and then select either **User** (displays only links to user libraries) or **All** (displays all links).

The color of the link arrow indicates the status of the link.

Color	Status
Black	Active link
Grey	Inactive link
Red	Active and modified

## Getting Information About Library Blocks

Use the `libinfo` command to get information about reference blocks in a system

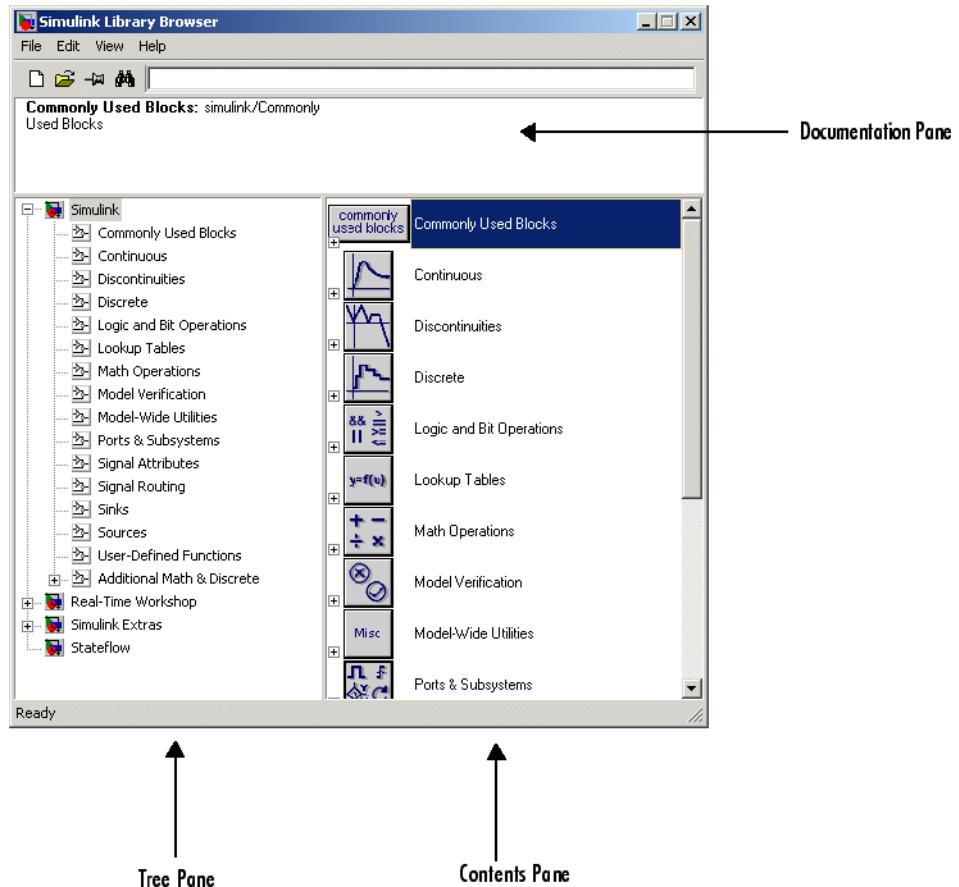
## Browsing Block Libraries

The Library Browser lets you quickly locate and copy library blocks into a model. To display the Library Browser, click the **Library Browser** button in the toolbar of the MATLAB desktop or Simulink model window or enter `simulink` at the MATLAB command line.

---

**Note** The Library Browser is available only on Microsoft Windows platforms.

The Library Browser contains three panes.



The tree pane displays all the block libraries installed on your system. The contents pane displays the blocks that reside in the library currently selected in the tree pane. The documentation pane displays documentation for the block selected in the contents pane.

You can locate blocks either by navigating the Library Browser's library tree or by using the Library Browser's search facility.

### Navigating the Library Tree

The library tree displays a list of all the block libraries installed on the system. You can view or hide the contents of libraries by expanding or collapsing the tree using the mouse or keyboard. To expand/collapse the tree, click the +/- buttons next to library entries or select an entry and press the +/- or right/left arrow key on your keyboard. Use the up/down arrow keys to move up or down the tree.

### Searching Libraries

To find a particular block, enter the block's name in the edit field next to the Library Browser's **Find** button, then click the **Find** button.

### Opening a Library

To open a library, right-click the library's entry in the browser. Simulink displays an **Open Library** button. Select the **Open Library** button to open the library.

### Creating and Opening Models

To create a model, select the **New** button on the Library Browser's toolbar. To open an existing model, select the **Open** button on the toolbar.

### Copying Blocks

To copy a block from the Library Browser into a model, select the block in the browser, drag the selected block into the model window, and drop it where you want to create the copy.

### Displaying Help on a Block

To display help on a block, right-click the block in the Library Browser and select the button that subsequently pops up.

### Pinning the Library Browser

To keep the Library Brower above all other windows on your desktop, select the **PushPin** button on the browser's toolbar.

# Working with Signals

---

This section describes how to create and use Simulink signals.

Signal Basics (p. 6-2)

Explores key signal concepts, including signal data types, signal buses, virtual signals, signal dimensions, and signal properties.

Determining Output Signal Dimensions (p. 6-14)

Explains the rules that determine the dimensions of signals that blocks output.

Displaying Signal Properties (p. 6-19)

How to display signal properties on a block diagram.

## Signal Basics

This section provides an overview of Simulink signals and explains how to specify, display, and check the validity of signal connections.

### About Signals

Simulink defines signals as the outputs of dynamic systems represented by blocks in a Simulink diagram and by the diagram itself. The lines in a block diagram represent mathematical relationships among the signals defined by the block diagram. For example, a line connecting the output of block A to the input of block B indicates that the signal output by B depends on the signal output by A.

---

**Note** It is tempting but misleading to think of Simulink signals as traveling along the lines that connect blocks the way electrical signals travel along a telephone wire. This analogy is misleading because it suggests that a block diagram represents physical connections between blocks, which is not the case. Simulink signals are mathematical, not physical, entities and the lines in a block diagram represent mathematical, not physical, relationships among signals.

---

### Creating Signals

You can create signals by creating source blocks in your model. For example, you can create a signal that varies sinusoidally with time by dragging an instance of the Sine block from the Simulink Sources library into the model. See “Sources” in the online Simulink block reference for information on blocks that you can use to create signals in a model. You can also use the Signal & Scope Manager to create signals in your model without using blocks. See “The Signal & Scope Manager” in the online Simulink documentation for more information.

### Signal Line Styles

Simulink uses a variety of line styles to display different types of signals in the model window. Assorted line styles help you to differentiate the signal types in Simulink diagrams. The signal types and their line styles are as follows:

Signal Type	Line Style	Description
Scalar and Nonscalar		Simulink uses a thin, solid line to represent the diagram's scalar and nonscalar signals.
Nonscalar		When the <b>Wide nonscalar lines</b> option is enabled, Simulink uses a thick, solid line to represent the diagram's nonscalar signals.
Control		Simulink uses a thin, dash-dot line to represent the diagram's control signals.
Bus		Simulink uses a thick, composite line to represent the diagram's signal buses.

Other than using the **Wide nonscalar lines** option to display nonscalar signals as thick, solid lines, you cannot customize or control the line style with which Simulink displays signals. See “Wide nonscalar lines” on page 6-19 for more information about this option.

---

**Note** As you construct a block diagram, Simulink uses a thin, solid line to represent all signal types. The lines are then redrawn using the specified line styles only after you update or start simulation of the block diagram.

---

## Signal Labels

A signal label is text that appears next to the line that represents a signal that has a name. The signal label displays the signal's name. In addition, if the signal is a virtual signal (see “Virtual Signals” on page 6-5) and its **Show propagated signals** property is on (see “Show propagated signals” in the online Simulink documentation), the label displays the names of the signals that make up the virtual signal.

Simulink creates a label for a signal when you assign it a name in the **Signal Properties** dialog box (see “Signal Properties Dialog Box” in the online Simulink documentation). You can change the signal's name by editing its label on the block diagram. To edit the label, left-click the label. Simulink replaces the label with an edit field. Edit the name in the edit field, then click outside the label to apply the change.

## Displaying Signal Values

As with creating signals, you can use either blocks or the Signal & Scope Manager to display the values of signals during a simulation. For example, you can use either the Scope block or the Signal & Scope Manager to graph time-varying signals on an oscilloscope-like display during simulation. See “Sinks” in the online Simulink block reference for information on blocks that you can use to display signals in a model.

## Signal Dimensions

Simulink blocks can output one- or two-dimensional signals. A one-dimensional (1-D) signal consists of a stream of one-dimensional arrays output at a frequency of one array (vector) per simulation time step. A two-dimensional (2-D) signal consists of a stream of two-dimensional arrays emitted at a frequency of one 2-D array (matrix) per block sample time. The Simulink user interface and documentation generally refer to 1-D signals as *vectors* and 2-D signals as *matrices*. A one-element array is frequently referred to as a *scalar*. A *row vector* is a 2-D array that has one row. A *column vector* is a 2-D array that has one column.

Simulink blocks vary in the dimensionality of the signals they can accept or output. Some blocks can accept or output signals of any dimensions. Some can accept or output only scalar or vector signals. To determine the signal dimensionality of a particular block, see the block’s description in “Blocks — Alphabetical List” in the online Simulink reference. See “Determining Output Signal Dimensions” on page 6-14 for information on what determines the dimensions of output signals for blocks that can output nonscalar signals.

---

**Note** Simulink does not support dynamic signal dimensions during simulation. That is, the size of a signal must remain constant while the simulation executes. You can alter a signal’s size only after terminating the simulation.

---

## Complex Signals

The values of Simulink signals can be complex numbers. A signal whose values are complex numbers is called a complex signal. You can introduce a complex-valued signal into a model in the following ways:

- Load complex-valued signal data from the MATLAB workspace into the model via a root-level Inport block.
- Create a Constant block in your model and set its value to a complex number.
- Create real signals corresponding to the real and imaginary parts of a complex signal, then combine the parts into a complex signal, using the Real-Imag to Complex conversion block.

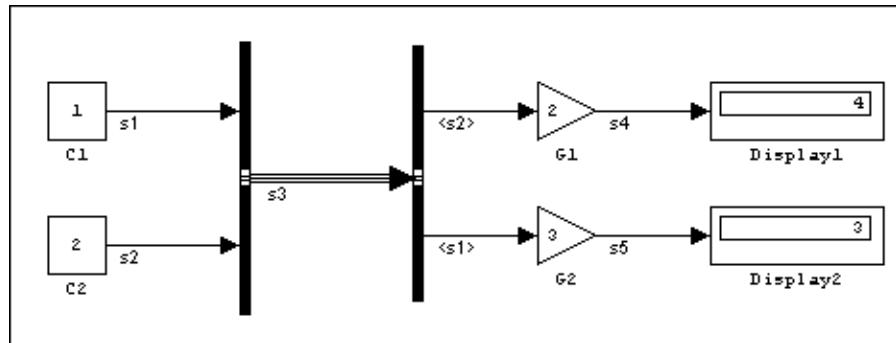
You can manipulate complex signals via blocks that accept them. If you are not sure whether a block accepts complex signals, see the documentation for the block in “Blocks — Alphabetical List” in the online Simulink reference.

## Virtual Signals

A *virtual signal* is a signal that represents another signal graphically. Some blocks, such as Bus Creator, Inport, and Outport blocks (see “Virtual Blocks” on page 5-2), generate virtual signals either exclusively or optionally (see “Virtual Versus Nonvirtual Buses” on page 6-9). Virtual signals are purely graphical entities. They have no mathematical or physical significance. Simulink ignores them when simulating a model.

Whenever you update or run a model, Simulink determines the nonvirtual signal(s) represented by the model’s virtual signal(s), using a procedure known as *signal propagation*. When running the model, Simulink uses the corresponding nonvirtual signal(s), determined via signal propagation, to drive the blocks to which the virtual signals are connected.

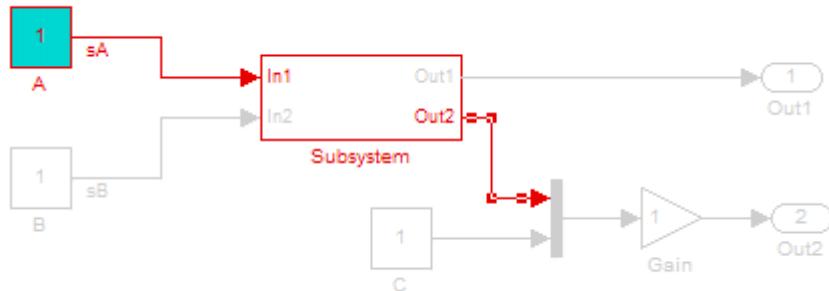
Consider, for example, the following model.



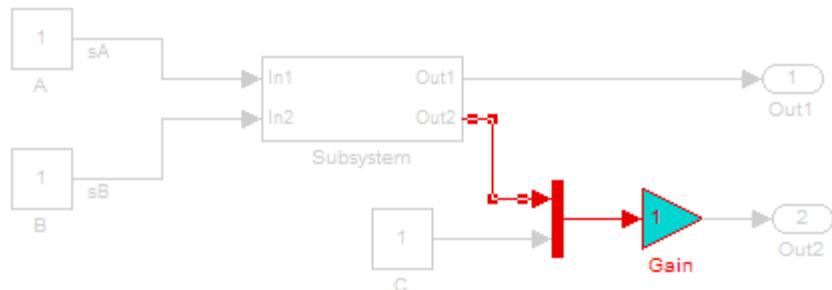
The signals driving Gain blocks G1 and G2 are virtual signals corresponding to signals s2 and s1, respectively. Simulink determines this automatically whenever you update or simulate the model.

### Displaying Virtual Signal Sources and Destinations

To display the nonvirtual block whose output is the source of a signal, select the signal and then select **Highlight to Source** from the signal's context menu. Simulink highlights the path from the signal's source block to the signal itself.



To display the nonvirtual block that is the destination of a signal, select the signal and then select **Highlight to Destination** from the signal's context menu. Simulink highlights the path from the selected signal to the nonvirtual block that it feeds.



To remove the highlighting, select **Remove Highlighting** from the block diagram's context menu.

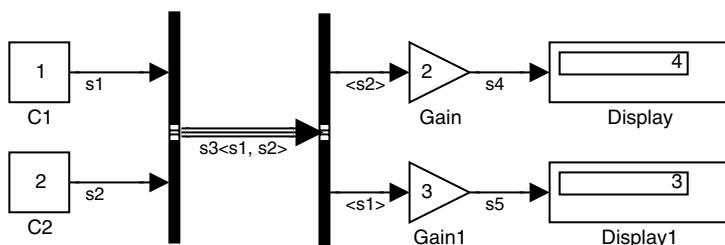
---

**Note** If the path from a source block or to a destination block includes an unresolved reference to a library block, Simulink highlights the path only from or to the library block, respectively. This is to avoid time-consuming library reference resolution while you are editing a model. To permit Simulink to display the complete path, first update the diagram (e.g., by pressing **Ctrl+D**). This causes Simulink to resolve all library references and hence display the complete path to a destination block or from a source block.

---

## Displaying the Nonvirtual Components of Virtual Signals

The **Show propagated signals** option (see “Signal Properties Dialog Box” in the online Simulink documentation) displays the nonvirtual signals represented by virtual signals in the labels of the virtual signals.



When you change the name of a nonvirtual signal, Simulink immediately updates the labels of all virtual signals that represent the nonvirtual signal and whose **Show propagated signals** is on, except if the path from the nonvirtual signal to the virtual signal includes an unresolved reference to a library block. In such cases, to avoid time-consuming library reference resolutions while you are editing a block diagram, Simulink defers updating the virtual signal's label until you update the model's block diagram either directly (e.g., by pressing **Ctrl+D**) or by simulating the model.

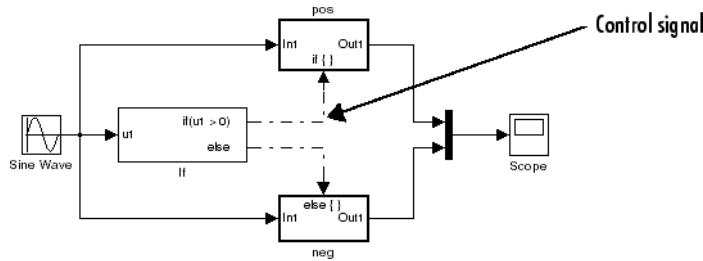
---

**Note** Virtual signals can represent virtual as well as nonvirtual signals. For example, you can use a Bus Creator block to combine multiple virtual and nonvirtual signals into a single virtual signal. If during signal propagation, Simulink determines that a component of a virtual signal is itself virtual, Simulink uses signal propagation to determine the nonvirtual components of the virtual component. This process continues until Simulink has determined all nonvirtual components of a virtual signal.

---

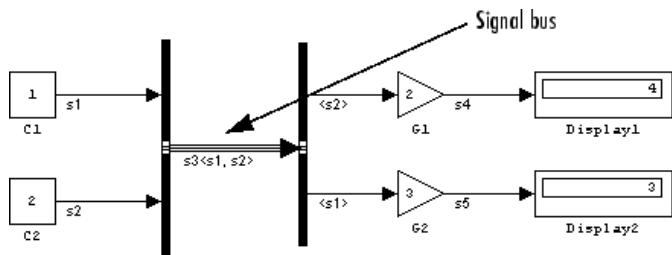
## Control Signals

A *control signal* is a signal used by one block to initiate execution of another block, e.g., a function-call or action subsystem. When you update or start simulation of a block diagram, Simulink uses a dash-dot pattern to redraw lines representing the diagram's control signals as illustrated in the following example.



## Signal Buses

A bus is a composite signal comprising a set of signals represented graphically by a bundle of lines. It is analogous to a bundle of wires held together by tie wraps. The components of a bus can have different data types and can themselves be composite signals (i.e., buses or muxed signals). You can use Bus Creator and Inport blocks to create signal buses and Bus Selector blocks to access a bus's components.



Selecting a bus and then **Signal Dimensions** from the model editor's **Format** menu displays the number of signal components carried by the bus.

## Virtual Versus Nonvirtual Buses

Buses may be either virtual or nonvirtual. During simulation, blocks connected to a virtual bus read their inputs from memory allocated to the component signals, which may reside in noncontiguous areas of memory. By contrast, blocks connected to a nonvirtual bus read their inputs from a copy of the component signals maintained by Simulink in a contiguous area of memory allocated to the bus.

Some Simulink features, require use of nonvirtual buses. Others require virtual buses. Nonvirtual buses also facilitate code generation by enabling buses to be represented as data structures. In general, virtual buses can save memory where nonvirtual buses are not required.

The Bus Creator and Import blocks output virtual buses by default. To cause them to output a nonvirtual bus, select the **Output as structure** option on their parameter dialog boxes. You can also use the Signal Conversion block to convert nonvirtual to virtual buses, and vice versa.

---

**Note** If a bus itself contains buses, the nested buses must all be either virtual or nonvirtual. A bus cannot contain a mixture of virtual and nonvirtual nested buses.

---

### Bus-Capable Blocks

A *bus-capable block* is a block through which both virtual and nonvirtual buses can pass. All virtual blocks are bus capable. Further, the following nonvirtual blocks are also bus-capable:

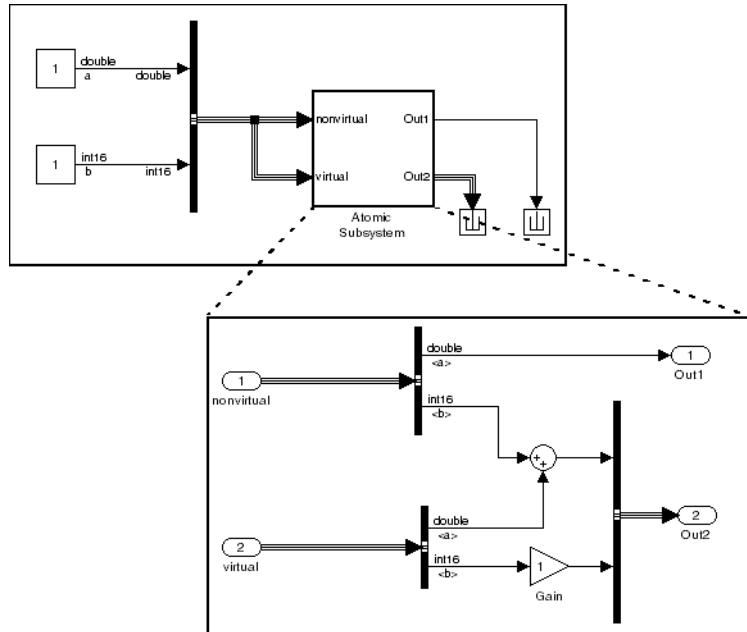
- Memory
- Merge
- Switch
- Multiport Switch
- Rate Transition
- Unit Delay
- Zero-Order Hold

Some bus-capable blocks impose constraints on bus propagation through them. See the documentation for the blocks in “Blocks-Alphabetical List” in the online Simulink reference for more information.

### Connecting Buses to Subsystem Imports

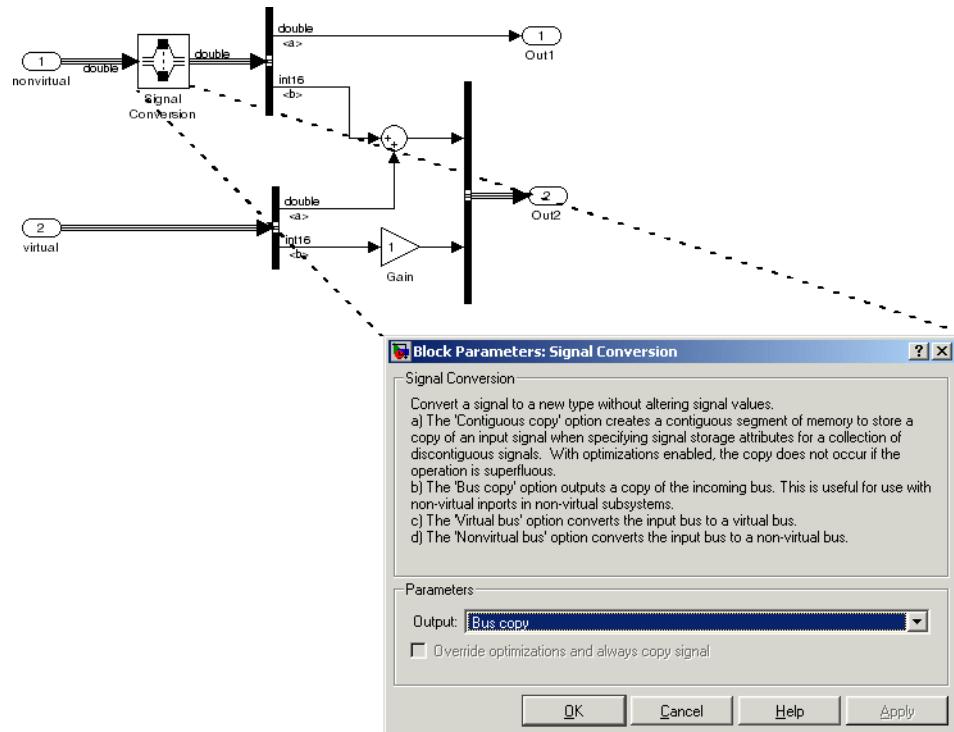
Generally, an Import block is a virtual block and hence accepts a bus as input. However, an Import block is nonvirtual if it resides in a conditionally executed or atomic subsystem and it or any of its components is directly connected to an output of the subsystem. In such a case, the Import block can accept a bus only if its components have the same data type. If the components are of differing data types, attempting to simulate the model causes Simulink to halt the simulation and display an error message. You can avoid this problem, without changing the semantics of your model, by inserting a Signal Conversion block between the Import block and the Outport block to which it was originally connected.

Consider, for example, the following model:



In this model, the Input labeled nonvirtual is nonvirtual because it resides in an atomic subsystem and one of its components (labeled a) is directly connected to one of the subsystem's outputs. Further, the bus connected to the subsystem's inputs has components of differing data types. As a result, Simulink cannot simulate this model.

Inserting a Signal Conversion block with the bus copy option selected breaks the direct connection to the subsystem's output and hence enables Simulink to simulate the model.



### Connecting Buses to Model Imports

If you want a root level Import of a model to be able to accept a bus signal, you must set the Import's bus object parameter to the name of a bus object that defines the type of bus that the Import accepts. See “Working with Data Objects” in the online Simulink documentation and `Simulink.Bus` class in the online Simulink reference for more information.

### Checking Signal Connections

Many Simulink blocks have limitations on the types of signals they can accept. Before simulating a model, Simulink checks all blocks to ensure that

they can accommodate the types of signals output by the ports to which they are connected. If any incompatibilities exist, Simulink reports an error and terminates the simulation. To detect such errors before running a simulation, choose **Update Diagram** from the Simulink **Edit** menu. Simulink reports any invalid connections found in the process of updating the diagram.

## Signal Glossary

The following table summarizes the terminology used to describe signals in the Simulink user interface and documentation.

Term	Meaning
Complex signal	Signal whose values are complex numbers.
Data type	Format used to represent signal values internally.
Matrix	Two-dimensional signal array.
Real signal	Signal whose values are real (as opposed to complex) numbers.
Scalar	One-element array, i.e., a one-element, 1-D or 2-D array.
Signal bus	A composite signal made up of other signals, including other buses. You can use Bus Creator and Import blocks to create signal buses.
Signal propagation	Process used by Simulink to determine attributes of signals and blocks, such as data types, labels, sample time, dimensionality, and so on, that are determined by connectivity.
Size	Number of elements that a signal contains. The size of a matrix (2-D) signal is generally expressed as M-by-N, where M is the number of columns and N is the number of rows making up the signal.
Vector	One-dimensional signal array.
Virtual signal	Signal that represents another signal or set of signals.
Width	Size of a vector signal.

## Determining Output Signal Dimensions

If a block can emit nonscalar signals, the dimensions of the signals that the block outputs depend on the block's parameters, if the block is a source block; otherwise, the output dimensions depend on the dimensions of the block's input and parameters.

### Determining the Output Dimensions of Source Blocks

A *source* block is a block that has no inputs. Examples of source blocks include the Constant block and the Sine Wave block. See the “Sources Library” table in the online Simulink block reference for a complete listing of Simulink source blocks. The output dimensions of a source block are the same as those of its output value parameters if the block's **Interpret Vector Parameters as 1-D** parameter is off (i.e., not selected in the block's parameter dialog box). If the **Interpret Vector Parameters as 1-D** parameter is on, the output dimensions equal the output value parameter dimensions unless the parameter dimensions are N-by-1 or 1-by-N. In the latter case, the block outputs a vector signal of width N.

As an example of how a source block's output value parameter(s) and **Interpret Vector Parameters as 1-D** parameter determine the dimensionality of its output, consider the Constant block. This block outputs a constant signal equal to its **Constant value** parameter. The following table illustrates how the dimensionality of the **Constant value** parameter and the setting of the **Interpret Vector Parameters as 1-D** parameter determine the dimensionality of the block's output.

<b>Constant Value</b>	<b>Interpret Vector Parameters as 1-D</b>	<b>Output</b>
2-D scalar	off	2-D scalar
2-D scalar	on	1-D scalar
1-by-N matrix	off	1-by-N matrix
1-by-N matrix	on	N-element vector
N-by-1 matrix	off	N-by-1 matrix
N-by-1 matrix	on	N-element vector

<b>Constant Value</b>	<b>Interpret Vector Parameters as 1-D</b>	<b>Output</b>
M-by-N matrix	off	M-by-N matrix
M-by-N matrix	on	M-by-N matrix

Simulink source blocks allow you to specify the dimensions of the signals that they output. You can therefore use them to introduce signals of various dimensions into your model.

## Determining the Output Dimensions of Nonsource Blocks

If a block has inputs, the dimensions of its outputs are, after scalar expansion, the same as those of its inputs. (All inputs must have the same dimensions, as discussed in “Signal and Parameter Dimension Rules” on page 6-15).

## Signal and Parameter Dimension Rules

When creating a Simulink model, you must observe the following rules regarding signal and parameter dimensions.

### Input Signal Dimension Rule

All nonscalar inputs to a block must have the same dimensions.

A block can have a mix of scalar and nonscalar inputs as long as all the nonscalar inputs have the same dimensions. Simulink expands the scalar inputs to have the same dimensions as the nonscalar inputs (see “Scalar Expansion of Inputs” on page 6-17) thus preserving the general rule.

### Block Parameter Dimension Rule

In general, a block’s parameters must have the same dimensions as the corresponding inputs.

Two seeming exceptions exist to this general rule:

- A block can have scalar parameters corresponding to nonscalar inputs. In this case, Simulink expands a scalar parameter to have the same dimensions as the corresponding input (see “Scalar Expansion of Parameters” on page 6-17) thus preserving the general rule.
- If an input is a vector, the corresponding parameter can be either an N-by-1 or a 1-by-N matrix. In this case, Simulink applies the N matrix elements to the corresponding elements of the input vector. This exception allows use of MATLAB row or column vectors, which are actually 1-by-N or N-by-1 matrices, respectively, to specify parameters that apply to vector inputs.

### **Vector or Matrix Input Conversion Rules**

Simulink converts vectors to row or column matrices and row or column matrices to vectors under the following circumstances:

- If a vector signal is connected to an input that requires a matrix, Simulink converts the vector to a one-row or one-column matrix.
- If a one-column or one-row matrix is connected to an input that requires a vector, Simulink converts the matrix to a vector.
- If the inputs to a block consist of a mixture of vectors and matrices and the matrix inputs all have one column or one row, Simulink converts the vectors to matrices having one column or one row, respectively.

---

**Note** You can configure Simulink to display a warning or error message if a vector or matrix conversion occurs during a simulation. See the “Vector/matrix block input conversion” diagnostic in the online Simulink documentation for more information.

---

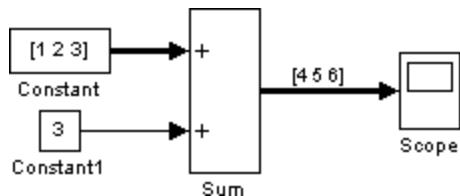
### **Scalar Expansion of Inputs and Parameters**

*Scalar expansion* is the conversion of a scalar value into a nonscalar array of the same dimensions. Many Simulink blocks support scalar expansion of inputs and parameters. Block descriptions in the online Simulink block reference indicate whether Simulink applies scalar expansion to a block’s inputs and parameters.

## Scalar Expansion of Inputs

Scalar expansion of inputs refers to the expansion of scalar inputs to match the dimensions of other nonscalar inputs or nonscalar parameters. When the input to a block is a mix of scalar and nonscalar signals, Simulink expands the scalar inputs into nonscalar signals having the same dimensions as the other nonscalar inputs. The elements of an expanded signal equal the value of the scalar from which the signal was expanded.

The following model illustrates scalar expansion of inputs. This model adds scalar and vector inputs. The input from block Constant1 is scalar expanded to match the size of the vector input from the Constant block. The input is expanded to the vector [3 3 3].

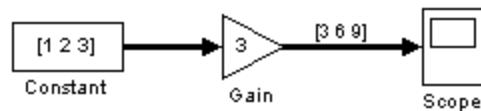


When a block's output is a function of a parameter and the parameter is nonscalar, Simulink expands a scalar input to match the dimensions of the parameter. For example, Simulink expands a scalar input to a Gain block to match the dimensions of a nonscalar gain parameter.

## Scalar Expansion of Parameters

If a block has a nonscalar input and a corresponding parameter is a scalar, Simulink expands the scalar parameter to have the same number of elements as the input. Each element of the expanded parameter equals the value of the original scalar. Simulink then applies each element of the expanded parameter to the corresponding input element.

This example shows that a scalar parameter (the Gain) is expanded to a vector of identically valued elements to match the size of the block input, a three-element vector.



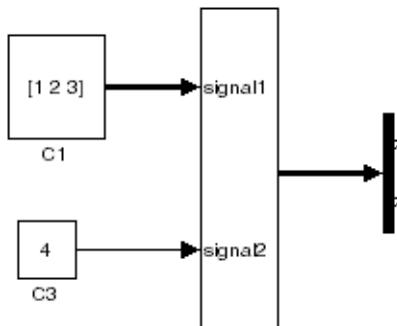
# Displaying Signal Properties

A model window's **Format** menu and its model context (right-click) menu offer the following options for displaying signal properties on the block diagram.

## Display Options

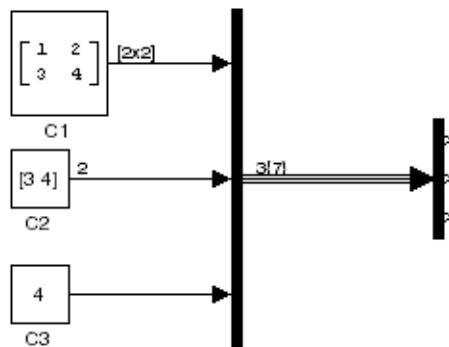
### Wide nonscalar lines

Draws lines that carry vector or matrix signals wider than lines that carry scalar signals.



## Signal dimensions

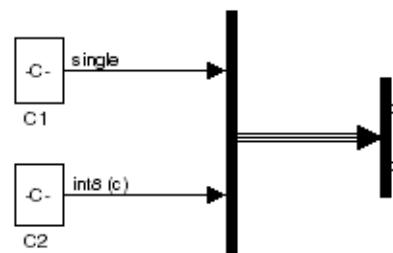
Display the dimensions of nonscalar signals next to the line that carries the signal.



The format of the display depends on whether the line represents a single signal or a bus. If the line represents a single vector signal, Simulink displays the width of the signal. If the line represents a single matrix signal, Simulink displays its dimensions as  $[N_1 \times N_2]$  where  $N_i$  is the size of the  $i$ th dimension of the signal. If the line represents a bus carrying signals of the same data type, Simulink displays  $N\{M\}$  where  $N$  is the number of signals carried by the bus and  $M$  is the total number of signal elements carried by the bus. If the bus carries signals of different data types, Simulink displays only the total number of signal elements  $\{M\}$ .

## Port data types

Displays the data type of a signal next to the output port that emits the signal.



The notation (c) following the data type of a signal indicates that the signal is complex.

## Signal Names

You can assign names to signals by

- Editing the signal's label
- Editing the **Name** field of the signal's property dialog (see "Signal Properties Dialog Box" in the online Simulink documentation)
- Setting the name parameter of the port or line that represents the signal, e.g.,

```
p = get_param(gcb, 'PortHandles')
l = get_param(p.Inport, 'Line')
set_param(l, 'Name', 's9')
```

## Signal Labels

A signal's label displays the signal's name. A virtual signal's label optionally displays the signals it represents in angle brackets. You can edit a signal's label, thereby changing the signal's name.

To create a signal label (and thereby name the signal), double-click the line that represents the signal. The text cursor appears. Enter the name and click anywhere outside the label to exit label editing mode.

---

**Note** When you create a signal label, take care to double-click the line. If you click in an unoccupied area close to the line, you will create a model annotation instead.

---

Labels can appear above or below horizontal lines or line segments, and left or right of vertical lines or line segments. Labels can appear at either end, at the center, or in any combination of these locations.

To move a signal label, drag the label to a new location on the line. When you release the mouse button, the label fixes its position near the line.

To copy a signal label, hold down the **Ctrl** key while dragging the label to another location on the line. When you release the mouse button, the label appears in both the original and the new locations.

To edit an existing signal label, select it:

- To replace the label, click the label, double-click or drag the cursor to select the entire label, then enter the new label.
- To insert characters, click between two characters to position the insertion point, then insert text.
- To replace characters, drag the mouse to select a range of text to replace, then enter the new text.

To delete all occurrences of a signal label, delete all the characters in the label. When you click outside the label, the labels are deleted. To delete a single occurrence of the label, hold down the **Shift** key while you select the label, then press the **Delete** or **Backspace** key.

To change the font of a signal label, select the signal, choose **Font** from the **Format** menu, then select a font from the **Set Font** dialog box.

### Displaying Signals Represented by Virtual Signals

To display the signal(s) represented by a virtual signal, click the signal's label and enter an angle bracket (<) after the signal's name. (If the signal has no name, simply enter the angle bracket.) Click anywhere outside the signal's label. Simulink exits label editing mode and displays the signals represented by the virtual signal in brackets in the label.

You can also display the signals represented by a virtual signal by selecting the **Show Propagated Signals** option on the signal's property dialog (see "Signal Properties Dialog Box" in the online Simulink documentation).

# Running Simulations

---

The following sections explain how to use Simulink to simulate a dynamic system.

Simulation Basics (p. 7-2)	How to start, suspend, stop, interact with, and diagnose errors in a simulation.
Specifying a Simulation Start and Stop Time (p. 7-6)	How to specify the start and stop time for a simulation.
Choosing a Solver (p. 7-7)	How to select the optimal solver for simulating a model.
Importing and Exporting Simulation Data (p. 7-17)	How to specify options for importing and exporting simulation data to the MATLAB workspace.
Configuration Sets (p. 7-29)	How to specify interchangeable sets of simulation configuration parameters for a model.
Configuration Parameters Dialog Box (p. 7-36)	How to use the Configuration Parameters dialog box to specify a simulation configuration.
Diagnosing Simulation Errors (p. 7-38)	How to use the Simulation Diagnostics Viewer to diagnose simulation errors.
Improving Simulation Performance and Accuracy (p. 7-40)	Tips on improving simulation performance and accuracy.

## Simulation Basics

You can simulate a Simulink model at any time simply by clicking the **Start** button on the Model Editor displaying the model (see “Starting a Simulation” on page 7-3). However, before starting the simulation, you may want to specify various simulation options, such as the simulation’s start and stop time and the type of solver used to solve the model at each simulation time step. Specifying simulation options is called configuring the model. Simulink enables you to create multiple model configurations, called configuration sets, modify existing configuration sets, and switch configuration sets with a click of a mouse button (see “Configuration Sets” on page 7-29 for information on creating and selecting configuration sets).

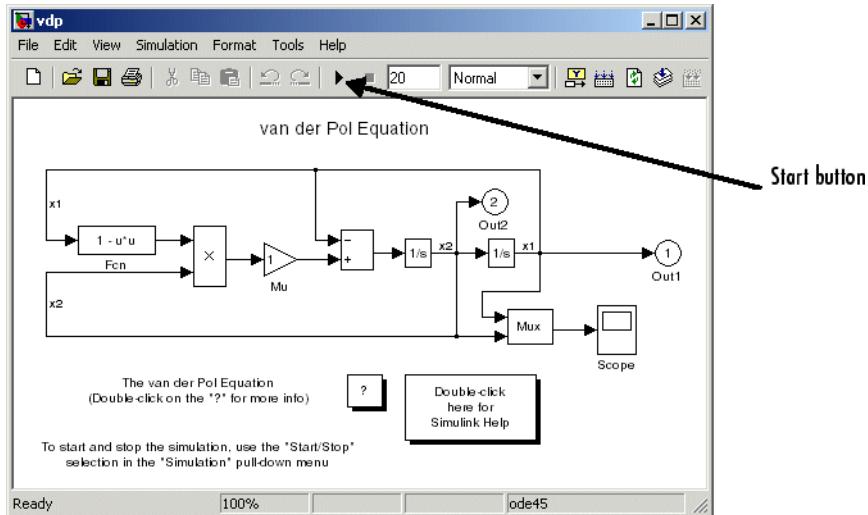
Once you have defined or selected a model configuration set that meets your needs, you can start the simulation. Simulink then runs the simulation from the specified start time to the specified stop time. While the simulation is running, you can interact with the simulation in various ways, stop or pause the simulation (see “Pausing or Stopping a Simulation” on page 7-4), and launch simulations of other models. If an error occurs during a simulation, Simulink halts the simulation and pops up a diagnostic viewer that helps you to determine the cause of the error.

### Controlling Execution of a Simulation

The Simulink graphical interface includes menu commands and toolbar buttons that enable you to start, stop, and pause a simulation.

## Starting a Simulation

To start execution of a model, select **Start** from the model editor's **Simulation** menu or click the **Start** button on the model's toolbar.



You can also use the keyboard shortcut, **Ctrl+T**, to start the simulation.

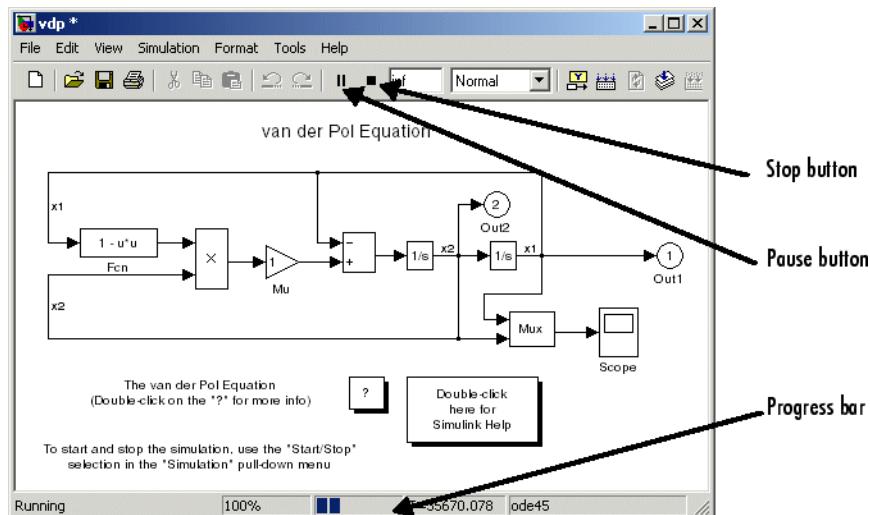
---

**Note** A common mistake that new Simulink users make is to start a simulation while the Simulink block library is the active window. Make sure your model window is the active window before starting a simulation.

---

Simulink starts executing the model at the start time specified on the **Configuration Parameters** dialog box. Execution continues until the simulation reaches the final time step specified on the **Configuration Parameters** dialog box, an error occurs, or you pause or terminate the simulation (see “Configuration Parameters Dialog Box” on page 7-36).

While the simulation is running, a progress bar at the bottom of the model window shows how far the simulation has progressed. A **Stop** command replaces the **Start** command on the **Simulation** menu. A **Pause** command appears on the menu and replaces the **Start** button on the model toolbar.



Your computer beeps to signal the completion of the simulation.

### Pausing or Stopping a Simulation

Select the **Pause** command or button to pause the simulation. Simulink completes execution of the current time step and suspends execution of the simulation. When you select **Pause**, the menu item and button change to **Continue**. (The button has the same appearance as the **Start** button). You can resume a suspended simulation at the next time step by choosing **Continue**.

To terminate execution of the model, select the **Stop** command or button. The keyboard shortcut for stopping a simulation is **Ctrl+T**, the same as for starting a simulation. Simulink completes execution of the current time step before terminating the model. Subsequently selecting the **Start** command or button restarts the simulation at the first time step specified on the **Configuration Parameters** dialog box.

If the model includes any blocks that write output to a file or to the workspace, or if you select output options on the **Configuration Parameters** dialog box, Simulink writes the data when the simulation is terminated or suspended.

## Interacting with a Running Simulation

You can perform certain operations interactively while a simulation is running. You can

- Modify some configuration parameters, including the stop time and the maximum step size
- Click a line to see the signal carried on that line on a floating (unconnected) Scope or Display block
- Modify the parameters of a block, as long as you do not cause a change in
  - Number of states, inputs, or outputs
  - Sample time
  - Number of zero crossings
  - Vector length of any block parameters
  - Length of the internal block work vectors
  - Dimension of any signals

You cannot make changes to the structure of the model, such as adding or deleting lines or blocks, during a simulation. If you need to make these kinds of changes, you need to stop the simulation, make the change, then start the simulation again to see the results of the change.

## Specifying a Simulation Start and Stop Time

Simulink simulations start by default at 0.0 seconds and end at 10.0 seconds. The **Solver** configuration pane allows you to specify other start and stop times for the currently selected simulation configuration. See “Solver Pane” in the online Simulink documentation for more information.

---

**Note** Simulation time and actual clock time are not the same. For example, running a simulation for 10 seconds usually does not take 10 seconds. The amount of time it takes to run a simulation depends on many factors, including the model’s complexity, the solver’s step sizes, and the computer’s speed.

---

# Choosing a Solver

A solver is a Simulink software component that determines the next time step that a simulation needs to take to meet target accuracy requirements that you specify. Simulink provides an extensive set of solvers, each adept at choosing the next time step for specific types of applications. The following sections explain how to choose the solver best suited to your application.

- “Choosing a Solver Type” on page 7-7
- “Choosing a Fixed-Step Solver” on page 7-8
- “Choosing a Variable-Step Solver” on page 7-13

For information on tailoring the selected solver to your model, see “Improving Simulation Accuracy” on page 7-41.

## Choosing a Solver Type

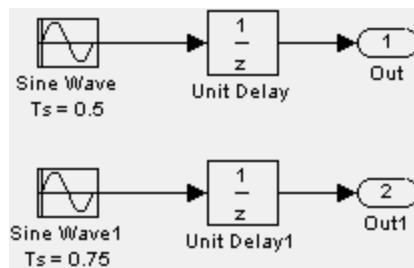
Simulink divides solvers into two types: fixed-step and variable-step. Both types of solvers compute the next simulation time as the sum of the current simulation time and a quantity known as the step size. With a fixed-step solver, the step size remains constant throughout the simulation. By contrast, with a variable-step solver, the step size can vary from step to step, depending on the model’s dynamics. In particular, a variable-step solver reduces the step size when a model’s states are changing rapidly to maintain accuracy and increases the step size when the system’s states are changing slowly in order to avoid taking unnecessary steps. The **Type** control on the **Simulink Solver** configuration pane allows you to select either of these two types of solvers (see “Solver Pane” in the online Simulink documentation).

The choice between the two types depends on how you plan to deploy your model and the model’s dynamics. If you plan to generate code from your model and run the code on a real-time computer system, you should choose a fixed-step solver to simulate the model. This is because real-time computer systems operate at fixed-size signal sample rates. A variable-step solver may cause the simulation to miss error conditions that can occur on a real-time computer system.

If you do not plan to deploy your model as generated code, the choice between a variable-step and a fixed-step solver depends on the dynamics of your model.

If your model's states change rapidly or contain discontinuities, a variable-step solver can shorten the time required to simulate your model significantly. This is because, for such a model, a variable-step solver can require fewer time steps than a fixed-step solver to achieve a comparable level of accuracy.

The following model illustrates how a variable-step solver can shorten simulation time for a multirate discrete model.



This model generates outputs at two different rates, every 0.5 second and every 0.75 second. To capture both outputs, the fixed-step solver must take a time step every 0.25 second (the *fundamental sample time* for the model).

`[0.0 0.25 0.5 0.75 1.0 1.25 ...]`

By contrast, the variable-step solver need take a step only when the model actually generates an output.

`[0.0 0.5 0.75 1.0 1.5 2.0 2.25 ...]`

This significantly reduces the number of time steps required to simulate the model.

The variable-step discrete solver uses zero-crossing detection (see “Zero-Crossing Detection” on page 2-19) to handle continuous signals. Simulink uses this solver by default if you specify a continuous solver and your model has no continuous states.

## Choosing a Fixed-Step Solver

When the **Type** control of the **Solver** configuration pane is set to **fixed-step**, the configuration pane’s **Solver** control allows you to choose one of the set

of fixed-step solvers that Simulink provides. The set of fixed-step solvers comprises two types of solvers: discrete and continuous.

### About the Fixed-Step Discrete Solver

The fixed-step discrete solver computes the time of the next time step by adding a fixed step size to the time of the current time. The accuracy and length of time of the resulting simulation depends on the size of the steps taken by the simulation: the smaller the step size, the more accurate the results but the longer the simulation takes. You can allow Simulink to choose the size of the step size (the default) or you can choose the step size yourself. If you allow Simulink to choose the step size, Simulink sets the step size to the fundamental sample time of the model if the model has discrete states or to the result of dividing the difference between the simulation start and stop time by 50 if the model has no discrete states. This choice ensures that the simulation will hit every simulation time required to update the model's discrete states at the model's specified sample times.

The fixed-step discrete solver has a fundamental limitation. It cannot be used to simulate models that have continuous states. That's because the fixed-step discrete solver relies on a model's blocks to compute the values of the states that they define. Blocks that define discrete states compute the values of those states at each time step taken by the solver. Blocks that define continuous states, on the other hand, rely on the solver to compute the states. Continuous solvers perform this task. You should thus select a continuous solver if your model contains continuous states.

---

**Note** If you attempt to use the fixed-step discrete solver to update or simulate a model that has continuous states, Simulink displays an error message. Thus, updating or simulating a model is a quick way to determine whether it has continuous states.

---

### About Fixed-Step Continuous Solvers

Simulink provides a set of fixed-step continuous solvers that, like the fixed-step discrete solver, compute the simulation's next time by adding a fixed-size time step to the current time. In addition, the continuous solvers employ numerical integration to compute the values of a model's continuous

states at the current step from the values at the previous step and the values of the state derivatives. This allows the fixed-step continuous solvers to handle models that contain both continuous and discrete states.

---

**Note** In theory, a fixed-step continuous solver can handle models that contain no continuous states. However, that would impose an unnecessary computational burden on the simulation. Consequently, Simulink always uses the fixed-step discrete solver for a model that contains no states or only discrete states, even if you specify a fixed-step continuous solver for the model.

---

Simulink provides two distinct types of fixed-step continuous solvers: explicit and implicit solvers. Explicit solvers (see “Explicit Fixed-Step Continuous Solvers” on page 7-10) compute the value of a state at the next time step as an explicit function of the current value of the state and the state derivative, e.g.,

$$X(n+1) = X(n) + h * DX(n)$$

where  $X$  is the state,  $DX$  is the state derivative, and  $h$  is the step size. An implicit solver (see “Implicit Fixed-Step Continuous Solvers” on page 7-12) computes the state at the next time step as an implicit function of the state and the state derivative at the next time step, e.g.,

$$X(n+1) - X(n) - h * DX(n+1) = 0$$

This type of solver requires more computation per step than an explicit solver but is also more accurate for a given step size. This solver thus can be faster than explicit fixed-step solvers for certain types of stiff systems.

**Explicit Fixed-Step Continuous Solvers.** Simulink provides a set of explicit fixed-step continuous solvers. The solvers differ in the specific integration technique used to compute the model’s state derivatives. The following table lists the available solvers and the integration techniques they use.

Solver	Integration Technique
ode1	Euler’s Method
ode2	Heun’s Method

<b>Solver</b>	<b>Integration Technique</b>
ode3	Bogacki-Shampine Formula
ode4	Fourth-Order Runge-Kutta (RK4) Formula
ode5	Dormand-Prince Formula

The integration techniques used by the fixed-step continuous solvers trade accuracy for computational effort. The table lists the solvers in order of the computational complexity of the integration methods they use from least complex (ode1) to most complex (ode5).

As with the fixed-step discrete solver, the accuracy and length of time of a simulation driven by a fixed-step continuous solver depends on the size of the steps taken by the solver: the smaller the step size, the more accurate the results but the longer the simulation takes. For any given step size, the more computationally complex the solver, the more accurate the simulation.

If you specify a fixed-step solver type for a model, Simulink sets the solver's model to ode3, i.e., it chooses a solver capable of handling both continuous and discrete states with moderate computational effort. As with the discrete solver, Simulink by default sets the step size to the fundamental sample time of the model if the model has discrete states or to the result of dividing the difference between the simulation start and stop time by 50 if the model has no discrete states. This assures that the solver will take a step at every simulation time required to update the model's discrete states at the model's specified sample rates. However, it does not guarantee that the default solver will accurately compute a model's continuous states or that the model cannot be simulated in less time with a less complex solver. Depending on the dynamics of your model, you may need to choose another solver and/or sample time to achieve acceptable accuracy or to shorten the simulation time.

**Implicit Fixed-Step Continuous Solvers.** Simulink provides one solver in this category: `ode14x`. This solver uses a combination of Newton's method and extrapolation from the current value to compute the value of a model state at the next time step. Simulink allows you to specify the number of Newton's method iterations and the extrapolation order that the solver uses to compute the next value of a model state (see “Fixed-Step Solver Options” in the online Simulink documentation). The more iterations and the higher the extrapolation order that you select, the greater the accuracy but also the greater the computational burden per step size.

### Choosing a Fixed-Step Continuous Solver

Any of the fixed-step continuous solvers in Simulink can simulate a model to any desired level of accuracy, given enough time and a small enough step size. Unfortunately, in general, it is not possible, or at least not practical, to decide *a priori* which solver and step size combination will yield acceptable results for a model’s continuous states in the shortest time. Determining the best solver for a particular model thus generally requires experimentation.

Here is the most efficient way to choose the best fixed-step solver for your model experimentally. First, use one of the variable-step solvers to simulate your model to the level of accuracy that you desire. This will give you an idea of what the simulation results should be. Next, use `ode1` to simulate your model at the default step size for your model. Compare the results of simulating your model with `ode1` with the results of simulating with the variable-step solver. If the results are the same within the specified level of accuracy, you have found the best fixed-step solver for your model, namely `ode1`. That’s because `ode1` is the simplest of the Simulink fixed-step solvers and hence yields the shortest simulation time for the current step size.

If `ode1` does not give accurate results, repeat the preceding steps with the other fixed-step solvers until you find the one that gives accurate results with the least computational effort. The most efficient way to do this is to use a binary search technique. First, try `ode3`. If it gives accurate results, try `ode2`. If `ode2` gives accurate results, it is the best solver for your model; otherwise, `ode3` is the best. If `ode3` does not give accurate results, try `ode5`. If `ode5` gives accurate results, try `ode4`. If `ode4` gives accurate results, select it as the solver for your model; otherwise, select `ode5`.

If ode5 does not give accurate results, reduce the simulation step size and repeat the preceding process. Continue in this way until you find a solver that solves your model accurately with the least computational effort.

## Choosing a Variable-Step Solver

When the **Type** control of the **Solver** configuration pane is set to **variable-step**, the configuration pane's **Solver** control allows you to choose one of the set of variable-step solvers that Simulink provides. As with fixed-step solvers in Simulink, the set of variable-step solvers comprises a discrete solver and a subset of continuous solvers. Both types compute the time of the next time step by adding a step size to the time of the current time that varies depending on the rate of change of the model's states. The continuous solvers, in addition, use numerical integration to compute the values of the model's continuous states at the next time step. Both types of solvers rely on blocks that define the model's discrete states to compute the values of the discrete states that each defines.

The choice between the two types of solvers depends on whether the blocks in your model defines states and, if so, the kind of states that they define. If your model defines no states or defines only discrete states, you should select the discrete solver. In fact, if a model has no states or only discrete states, Simulink will use the discrete solver to simulate the model even if the model specifies a continuous solver.

### About Variable-Step Continuous Solvers

Simulink variable-step solvers vary the step size during the simulation, reducing the step size to increase accuracy when a model's states are changing rapidly and increasing the step size to avoid taking unnecessary steps when the model's states are changing slowly. Computing the step size adds to the computational overhead at each step but can reduce the total number of steps, and hence simulation time, required to maintain a specified level of accuracy for models with rapidly changing or piecewise continuous states.

Simulink provides the following variable-step continuous solvers:

- **ode45** is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. It is a *one-step* solver; that is, in computing  $y(t_n)$ , it needs only the solution at the immediately preceding time point,  $y(t_{n-1})$ . In

general, `ode45` is the best solver to apply as a first try for most problems. For this reason, `ode45` is the default solver used by Simulink for models with continuous states.

- `ode23` is also based on an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine. It can be more efficient than `ode45` at crude tolerances and in the presence of mild stiffness. `ode23` is a one-step solver.
- `ode113` is a variable-order Adams-Basforth-Moulton PECE solver. It can be more efficient than `ode45` at stringent tolerances. `ode113` is a *multistep* solver; that is, it normally needs the solutions at several preceding time points to compute the current solution.
- `ode15s` is a variable-order solver based on the numerical differentiation formulas (NDFs). These are related to but are more efficient than the backward differentiation formulas, BDFs (also known as Gear's method). Like `ode113`, `ode15s` is a multistep method solver. If you suspect that a problem is stiff, or if `ode45` failed or was very inefficient, try `ode15s`.
- `ode23s` is based on a modified Rosenbrock formula of order 2. Because it is a one-step solver, it can be more efficient than `ode15s` at crude tolerances. It can solve some kinds of stiff problems for which `ode15s` is not effective.
- `ode23t` is an implementation of the trapezoidal rule using a "free" interpolant. Use this solver if the problem is only moderately stiff and you need a solution without numerical damping.
- `ode23tb` is an implementation of TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order two. By construction, the same iteration matrix is used in evaluating both stages. Like `ode23s`, this solver can be more efficient than `ode15s` at crude tolerances.

---

**Note** For a *stiff* problem, solutions can change on a time scale that is very short compared to the interval of integration, but the solution of interest changes on a much longer time scale. Methods not designed for stiff problems are ineffective on intervals where the solution changes slowly because they use time steps small enough to resolve the fastest possible change. Jacobian matrices are generated numerically for `ode15s` and `ode23s`. For more information, see Shampine, L. F., *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, 1994.

---

## Specifying Variable-Step Solver Error Tolerances

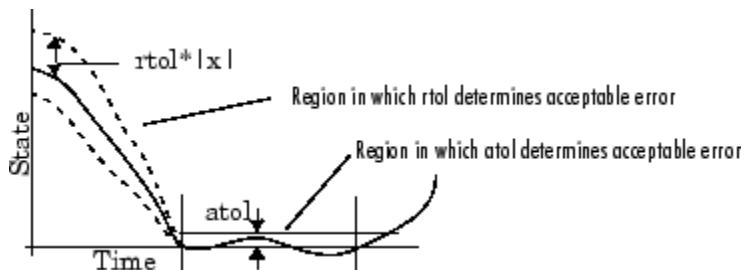
The solvers use standard local error control techniques to monitor the error at each time step. During each time step, the solvers compute the state values at the end of the step and also determine the *local error*, the estimated error of these state values. They then compare the local error to the *acceptable error*, which is a function of the relative tolerance (*rtol*) and absolute tolerance (*atol*). If the error is greater than the acceptable error for *any* state, the solver reduces the step size and tries again:

- *Relative tolerance* measures the error relative to the size of each state. The relative tolerance represents a percentage of the state's value. The default, 1e-3, means that the computed state is accurate to within 0.1%.
- *Absolute tolerance* is a threshold error value. This tolerance represents the acceptable error as the value of the measured state approaches zero.

The error for the *i*th state,  $e_i$ , is required to satisfy

$$e_i \leq \max(rtolf \times |x_i|, atol_i)$$

The following figure shows a plot of a state and the regions in which the acceptable error is determined by the relative tolerance and the absolute tolerance.



If you specify *auto* (the default), Simulink sets the absolute tolerance for each state initially to 1e-6. As the simulation progresses, Simulink resets the absolute tolerance for each state to the maximum value that the state has assumed thus far times the relative tolerance for that state. Thus, if a state goes from 0 to 1 and *reltol* is 1e-3, then by the end of the simulation the *abstol* is set to 1e-3 also. If a state goes from 0 to 1000, then the *abstol* is set to 1.

If the computed setting is not suitable, you can determine an appropriate setting yourself. You might have to run a simulation more than once to determine an appropriate value for the absolute tolerance.

The Integrator, Transfer Fcn, State-Space, and Zero-Pole blocks allow you to specify absolute tolerance values for solving the model states that they compute or that determine their output. The absolute tolerance values that you specify for these blocks override the global settings in the **Configuration Parameters** dialog box. You might want to override the global setting in this way, if the global setting does not provide sufficient error control for all of your model's states, for example, because they vary widely in magnitude.

# Importing and Exporting Simulation Data

Simulink allows you to import input signal and initial state data from the MATLAB workspace and export output signal and state data to the MATLAB workspace during simulation. This capability allows you to use standard or custom MATLAB functions to generate a simulated system's input signals and to graph, analyze, or otherwise postprocess the system's outputs. See the following sections for more information:

- “Importing Data from the MATLAB Workspace” on page 7-17
- “Exporting Data to the MATLAB Workspace” on page 7-22
- “Importing and Exporting States” on page 7-24

## Importing Data from the MATLAB Workspace

Simulink can apply input from a model's base workspace to the model's top-level input ports during a simulation run. To specify this option, select the **Input** box in the **Load from workspace** area of the **Data Import/Export** pane (see “Data Import/Export Pane” in the online Simulink documentation). Then, enter an external input specification (see “Importing Data Arrays” on page 7-17) in the adjacent edit box and click **Apply**.

The input data can take any of the following forms:

- array—see “Importing Data Arrays” on page 7-17
- time expression—see “Using a MATLAB Time Expression to Import Data” on page 7-18
- structure—see “Importing Data Structures” on page 7-19
- time series—see “Importing Time-Series Data” on page 7-21

Simulink linearly interpolates or extrapolates input values as necessary if the **Interpolate data** option is selected for the corresponding Import.

## Importing Data Arrays

This import format consists of a real (noncomplex) matrix of data type `double`. The first column of the matrix must be a vector of times in ascending order. The remaining columns specify input values. In particular, each column

represents the input for a different Import block signal (in sequential order) and each row is the input value for the corresponding time point.

The total number of columns of the input matrix must equal  $n + 1$ , where  $n$  is the total number of signals entering the model's input ports.

The default input expression for a model is  $[t, u]$  and the default input format is **Array**. So if you define  $t$  and  $u$  in the base workspace, you need only select the **Input** option to input data from the model's base workspace. For example, suppose that a model has two input ports, one of which accepts two signals and the other of which accepts one signal. Also, suppose that the base workspace defines  $u$  and  $t$  as follows:

```
t = (0:0.1:1)';
u = [sin(t), cos(t), 4*cos(t)];
```

---

**Note** The array input format allows you to load only real (noncomplex) scalar or vector data of type **double**. Use the structure format to input complex data, matrix (2-D) data, and/or data types other than **double**.

---

### Using a MATLAB Time Expression to Import Data

You can use a MATLAB time expression to import data from the MATLAB workspace. To use a time expression, enter the expression as a string (i.e., enclosed in single quotes) in the **Input** field of the **Data Import/Export** pane. The time expression can be any MATLAB expression that evaluates to a row vector equal in length to the number of signals entering the model's input ports. For example, suppose that a model has one vector Import that accepts two signals. Furthermore, suppose that `timefcn` is a user-defined function that returns a row vector two elements long. The following are valid input time expressions for such a model:

```
'[3*sin(t), cos(2*t)]'
'4*timefcn(w*t)+7'
```

Simulink evaluates the expression at each step of the simulation, applying the resulting values to the model's input ports. Note that Simulink defines the variable  $t$  when it runs the simulation. Also, you can omit the time variable

in expressions for functions of one variable. For example, Simulink interprets the expression `sin` as `sin(t)`.

## Importing Data Structures

Simulink can read data from the workspace in the form of a structure whose name is specified in the **Input** text field. You can import structures that include only signal data or both signal and time data.

**Importing signal-and-time data structures.** To import structures that include both signal and time data, select the **Structure with time** option on from the **Format** list on the **Data Import/Export** pane. The input structure must have two top-level fields: `time` and `signals`. The `time` field contains a column vector of the simulation times. The `signals` field contains an array of substructures, each of which corresponds to a model input port.

Each `signals` substructure must contain two fields named `values` and `dimensions`, respectively. The `values` field must contain an array of inputs for the corresponding input port where each input corresponds to a time point specified by the `time` field. The `dimensions` field specifies the dimensions of the input. If each input is a scalar or vector (1-D array) value, the `dimensions` field must be a scalar value that specifies the length of the vector (1 for a scalar). If each input is a matrix (2-D array), the `dimensions` field must be a two-element vector whose first element specifies the number of rows in the matrix and whose second element specifies the number of columns.

---

**Note** You must set the **Port dimensions** parameter of the Import to be the same value as the `dimensions` field of the corresponding input structure. If the values differ, Simulink stops and displays an error message when you try to simulate the model.

---

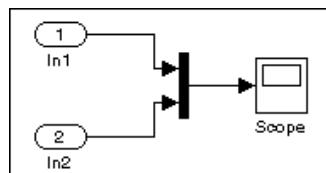
If the inputs for a port are scalar or vector values, the `values` field must be an  $M$ -by- $N$  array where  $M$  is the number of time points specified by the `time` field and  $N$  is the length of each vector value. For example, the following code creates an input structure for loading 11 time samples of a two-element signal vector of type `int8` into a model with a single input port:

```
a.time = (0:0.1:1)';
c1 = int8([0:1:10]');
c2 = int8([0:10:100]');
a.signals(1).values = [c1 c2];
a.signals(1).dimensions = 2;
```

To load this data into the model's input port, you would select the **Input** option on the **Data Import/Export** pane and enter *a* in the input expression field.

If the inputs for a port are matrices (2-D arrays), the **values** field must be an M-by-N-by-T array where M and N are the dimensions of each matrix input and T is the number of time points. For example, suppose that you want to input 51 time samples of a 4-by-5 matrix signal into one of your model's input ports. Then, the corresponding **dimensions** field of the workspace structure must equal [4 5] and the **values** array must have the dimensions 4-by-5-by-51.

As another example, consider the following model, which has two inputs.



Suppose that you want to input a sine wave into the first port and a cosine wave into the second port. To do this, define a vector, *a*, as follows, in the base workspace:

```
a.time = (0:0.1:1)';
a.signals(1).values = sin(a.time);
a.signals(1).dimensions = 1;
a.signals(2).values = cos(a.time);
a.signals(2).dimensions = 1;
```

Select the **Input** box for this model, enter *a* in the adjacent text field, and select **StructureWithTime** as the I/O format.

**Importing Signal-Only Structures.** The Structure format is the same as the Structure with time format except that the time field is empty. For example, in the preceding example, you could set the time field as follows:

```
a.time = []
```

In this case, Simulink reads the input for the first time step from the first element of an input port's value array, the value for the second time step from the second element of the value array, etc.

**Per-Port Structures.** This format consists of a separate structure-with-time or structure-without-time for each port. Each port's input data structure has only one signals field. To specify this option, enter the names of the structures in the **Input** text field as a comma-separated list, in1, in2, ..., inN, where in1 is the data for your model's first port, in2 for the second input port, and so on.

## Importing Time-Series Data

Any root-level Import block can import data specified by a time-series object (see `Simulink.Timeseries` in the online Simulink reference) residing in the MATLAB workspace. In addition, any root-level input port defined by a bus object (see `Simulink.Bus` in the online Simulink reference) can import data from a time-series array object (see `Simulink.TSArray` in the online Simulink reference) that has the same structure as the bus object.

This capability allows you to import data logged by a previous simulation run (see “Logging Signals” in the online Simulink reference). For example, suppose that you have a model that references several other models. You could use data logged from the inputs of the referenced models when simulating the top model as inputs for the referenced models simulated by themselves. This allows you to test the referenced models independently of the top model and each other.

To import data from time-series and time-series array objects, enter a comma-separated list of variables or expressions that evaluate to the objects in the **Input** edit field on the **Data Import/Export** pane of the Configuration Parameters dialog box (see “Configuration Parameters Dialog Box” on page 7-36). Each item in the **Input** list corresponds to one of the model's root-level

input ports, with the first item corresponding to the first root-level input port, the second to the second root-level input port, and so on.

## Exporting Data to the MATLAB Workspace

You can specify return variables by selecting the **Time**, **States**, and/or **Output** check boxes in the **Save to workspace** area of this dialog box pane. Specifying return variables causes Simulink to write values for the time, state, and output trajectories (as many as are selected) into the workspace.

To assign values to different variables, specify those variable names in the fields to the right of the check boxes. To write output to more than one variable, specify the variable names in a comma-separated list. Simulink saves the simulation times in the vector specified in the **Save to workspace** area.

---

**Note** Simulink saves the output to the workspace at the base sample rate of the model. Use a To Workspace block if you want to save output at a different sample rate.

---

The **Save options** area enables you to specify the format and restrict the amount of output saved.

Format options for model states and outputs are listed below.

### Format Options

**Array.** If you select this option, Simulink saves a model's states and outputs in a state and output array, respectively.

The state matrix has the name specified in the **Save to workspace** area (for example, `xout`). Each row of the state matrix corresponds to a time sample of the model's states. Each column corresponds to an element of a state. For example, suppose that your model has two continuous states, each of which is a two-element vector. Then the first two elements of each row of the state matrix contains a time sample of the first state vector. The last two elements of each row contain a time sample of the second state vector.

The model output matrix has the name specified in the **Save to workspace** area (for example, *yout*). Each column corresponds to a model output port, each row to the outputs at a specific time.

---

**Note** You can use array format to save your model's outputs and states only if the outputs are either all scalars or all vectors (or all matrices for states), are either all real or all complex, and are all of the same data type. Use the **Structure** or **StructureWithTime** output formats (see “Structure with time” on page 7-23) if your model's outputs and states do not meet these conditions.

---

**Structure with time.** If you select this format, Simulink saves the model's states and outputs in structures having the names specified in the **Save to workspace** area (for example, *xout* and *yout*).

The structure used to save outputs has two top-level fields: **time** and **signals**. The **time** field contains a vector of the simulation times. The **signals** field contains an array of substructures, each of which corresponds to a model output port. Each substructure has four fields: **values**, **dimensions**, **label**, and **blockName**. The **values** field contains the outputs for the corresponding output port. If the outputs are scalars or vectors, the **values** field is a matrix each of whose rows represents an output at the time specified by the corresponding element of the time vector. If the outputs are matrix (2-D) values, the **values** field is a 3-D array of dimensions  $M$ -by- $N$ -by- $T$  where  $M$ -by- $N$  is the dimensions of the output signal and  $T$  is the number of output samples. If  $T = 1$ , MATLAB drops the last dimension. Therefore, the **values** field is an  $M$ -by- $N$  matrix. The **dimensions** field specifies the dimensions of the output signal. The **label** field specifies the label of the signal connected to the output port or the type of state (continuous or discrete). The **blockName** field specifies the name of the corresponding output port or block with states.

The structure used to save states has a similar organization. The **states** structure has two top-level fields: **time** and **signals**. The **time** field contains a vector of the simulation times. The **signals** field contains an array of substructures, each of which corresponds to one of the model's states. Each **signals** structure has four fields: **values**, **dimensions**, **label**, and **blockName**. The **values** field contains time samples of a state of the block specified by the **blockName** field. The **label** field for built-in blocks indicates the type of state: either **CSTATE** (continuous state) or **DSTATE** (discrete state).

For S-Function blocks, the label contains whatever name is assigned to the state by the S-Function block.

The time samples of a state are stored in the values field as a matrix of values. Each row corresponds to a time sample. Each element of a row corresponds to an element of the state. If the state is a matrix, the matrix is stored in the values array in column-major order. For example, suppose that the model includes a 2-by-2 matrix state and that Simulink logs 51 samples of the state during a simulation run. The values field for this state would contain a 51-by-4 matrix where each row corresponds to a time sample of the state and where the first two elements of each row correspond to the first column of the sample and the last two elements correspond to the second column of the sample.

---

**Note** Simulink can read back simulation data saved to the workspace in the Structure with time output format. See “Importing signal-and-time data structures” on page 7-19 for more information.

---

**Structure.** This format is the same as the preceding except that Simulink does not store simulation times in the time field of the saved structure.

**Per-Port Structures.** This format consists of a separate structure-with-time or structure-without-time for each output port. Each output data structure has only one signals field. To specify this option, enter the names of the structures in the **Output** text field as a comma-separated list, out1, out2, . . . , outN, where out1 is the data for your model’s first port, out2 for the second input port, and so on.

## Importing and Exporting States

Simulink allows you to import the initial values of a system’s states, i.e., its initial conditions, at the beginning of a simulation and save the final values of the states at the end of the simulation. This feature allows you to save a steady-state solution and restart the simulation at that known state.

## Saving Final States

To save the final values of a model's states, check **Final states** in the **Save to workspace** area of the **Data Import/Export** pane and enter a name in the adjacent edit field. Simulink saves the states in a workspace variable having the specified name. The saved data has the format that you specify in the **Save options** area of the **Data Import/Export** pane.

When saving states from a referenced model in the structure-with-time format, Simulink adds a boolean subfield named `inReferencedModel` to the `signals` field of the saved data structure. This field's value is true (1) if the `signals` field records the final state of a block that resides in the submodel, e.g.,

```
>> xout.signals(1)

ans =

    values: [101x1 double]
    dimensions: 1
        label: 'DSTATE'
    blockName: [1x66 char]
inReferencedModel: 1
```

If the `signals` field records a submodel state, its `blockName` subfield contains a compound path comprising a top model path and a submodel path. The top model path is the path from the model root to the Model block that references the submodel. The submodel path is the path from the submodel root to the block whose state the `signals` field records. The compound path uses a | character to separate the top and submodel paths, e.g.,

```
>> xout.signals(1).blockName

ans =

sldemo_mdlref_basic/CounterA|sldemo_mdlref_counter/Previous Output
```

## Loading Initial States

To load states, check **Initial state** in the **Load from workspace** area of the **Data Import/Export** pane and specify the name of a variable that contains the initial state values, for example, a variable containing states saved from a previous simulation. The initial values specified by the workspace variable override the initial values specified by the model itself, i.e., the values specified by the initial condition parameters of those blocks in the model that have states.

---

**Note** You must use the structure or structure-with-time format to initialize the states of a top model and the models that it references.

---

## Limiting Output

Saving data to the workspace can slow down the simulation and consume memory. To avoid this, you can limit the number of samples saved to the most recent samples or you can skip samples by applying a decimation factor. To set a limit on the number of data samples saved, select the check box labeled **Limit data points to last** and specify the number of samples to save. To apply a decimation factor, enter a value in the field to the right of the **Decimation** label. For example, a value of 2 saves every other point generated.

## Specifying Output Options

The **Output options** list on the **Data Import/Export** configuration pane () enables you to control how much output the simulation generates. You can choose from three options:

- Refine output
- Produce additional output
- Produce specified output only

## Refining Output

The **Refine output** choice provides additional output points when the simulation output is too coarse. This parameter provides an integer number

of output points between time steps; for example, a refine factor of 2 provides output midway between the time steps, as well as at the steps. The default refine factor is 1.

To get smoother output, it is much faster to change the refine factor instead of reducing the step size. When the refine factor is changed, the solvers generate additional points by evaluating a continuous extension formula at those points. Changing the refine factor does not change the steps used by the solver.

The refine factor applies to variable-step solvers and is most useful when you are using `ode45`. The `ode45` solver is capable of taking large steps; when graphing simulation output, you might find that output from this solver is not sufficiently smooth. If this is the case, run the simulation again with a larger refine factor. A value of 4 should provide much smoother results.

---

**Note** This option helps the solver to locate zero crossings (see “Zero-Crossing Detection” on page 2-19).

---

## Producing Additional Output

The Produce additional output choice enables you to specify directly those additional times at which the solver generates output. When you select this option, Simulink displays an **Output times** field on the **Data Import/Export** pane. Enter a MATLAB expression in this field that evaluates to an additional time or a vector of additional times. The additional output is produced using a continuous extension formula at the additional times. Unlike the refine factor, this option changes the simulation step size so that time steps coincide with the times that you have specified for additional output.

## Producing Specified Output Only

The Produce specified output only choice provides simulation output *only* at the specified output times. This option changes the simulation step size so that time steps coincide with the times that you have specified for producing output. This choice is useful when you are comparing different simulations to ensure that the simulations produce output at the same times.

## Comparing Output Options

A sample simulation generates output at these times:

0, 2.5, 5, 8.5, 10

Choosing Refine output and specifying a refine factor of 2 generates output at these times:

0, 1.25, 2.5, 3.75, 5, 6.75, 8.5, 9.25, 10

Choosing the Produce additional output option and specifying [0:10] generates output at these times

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

and perhaps at additional times, depending on the step size chosen by the variable-step solver.

Choosing the Produce specified output only option and specifying [0:10] generates output at these times:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

# Configuration Sets

A configuration set is a named set of values for a model's parameters, such as solver type and simulation start or stop time. Every new model is created with a default configuration set, called Configuration, that initially specifies default values for the model's parameters. You can subsequently create and modify additional configuration sets and associate them with the model. The sets associated with a model can each specify different values for any given model parameter.

## Configuration Set Components

A configuration set comprises groups of related parameters called components. Every configuration set includes the following components:

- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing

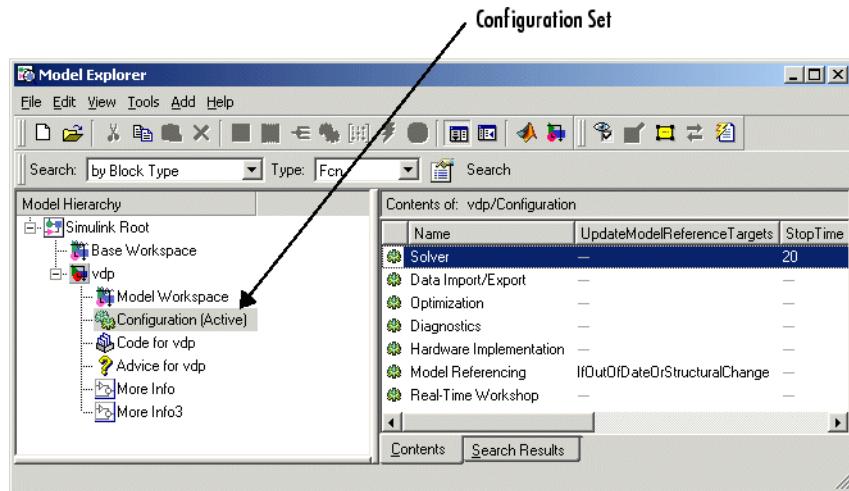
Some Simulink-based products, such as Real-Time Workshop, define additional components. If such a product is installed on your system, the configuration set also contains the components that it defines.

## The Active Set

Only one of the configuration sets associated with a model is active at any given time. The active set determines the current values of the model's model parameters. Changing the value of a parameter in the Model Explorer changes its value in the active set. Simulink allows you to change the active set at any time (except when executing the model). In this way, you can quickly reconfigure a model for different purposes, e.g., testing and production, or apply standard configuration settings to new models.

## Displaying Configuration Sets

To display the configuration sets associated with a model, open the Model Explorer (see “The Model Explorer” on page 8-2). The configuration sets associated with the model appear as gear-shaped nodes in the Model Explorer’s **Model Hierarchy** pane.



The Model Explorer’s **Contents** pane displays the components of the selected configuration set. The Model Explorer’s Dialog pane display a dialog for setting the parameters of the selected group (see “Configuration Parameters Dialog Box” on page 7-36).

## Activating a Configuration Set

To activate a configuration set, right-click the configuration set’s node to display the node’s context menu, then select **Activate** from the context menu.

## Copying, Deleting, and Moving Configuration Sets

You can use edit commands on the Model Explorer's **Edit** or context menus or object drag-and-drop operations to delete, copy, or move configuration sets among models displayed in the Model Explorer's **Model Hierarchy** pane.

For example, to copy a configuration set, using edit commands:

- 1 Select the configuration set that you want to copy in the **Model Hierarchy** pane.
- 2 Select **Copy** from the Model Explorer's **Edit** menu or the configuration set's context menu or press **Ctrl+C**.
- 3 Select the model in which you want to create the copy

---

**Note** You can create a copy in the same model as the original.

---

- 4 Select **Paste** from the Model Explorer's **Edit** menu or from the model's context menu or press **Ctrl+V**.

To copy the configuration set, using object drag-and-drop, hold the **Ctrl** key and the right mouse button down and drag the configuration set's node to the node of the model in which you want to create the copy. To move a configuration set from one model to another, using drag-and-drop, hold the **Ctrl** key and the left mouse button down and drag the configuration set's node to the node of the destination model.

---

**Note** You cannot move or delete a model's active configuration set.

---

## Copying Configuration Set Components

To copy a configuration set component from one configuration set to another:

- 1 Select the component in the Model Explorer's **Contents** pane.
- 2 Select **Copy** from the Model Explorer's **Edit** menu or the component's context menu or press **Ctrl+C**.

- 3** Select the configuration set into which you want to copy the component.
- 4** Select **Paste** from the Model Explorer's **Edit** menu or the component's context menu or press **Ctrl+C**.

---

**Note** The copy replaces the component of the same name in the destination configuration set. For example, if you copy the Solver component of configuration set A and paste it into configuration set B, the copy replaces B's existing Solver component.

---

## **Creating Configuration Sets**

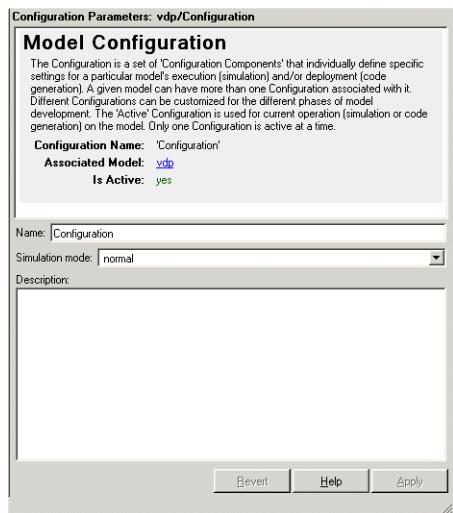
To create a new configuration set, copy an existing configuration set.

## **Setting Values in Configuration Sets**

To set the value of a parameter in a configuration set, select the configuration set in the Model Explorer and then edit the value of the parameter on the corresponding dialog in the Model Explorer's dialog view.

## **Model Configuration Dialog Box**

The Model Configuration dialog box appears when you select a model configuration in the Model Explorer.



The dialog box has the following fields.

### Name

Name of the configuration. You can change the name of the configuration by editing this field.

### Simulation mode

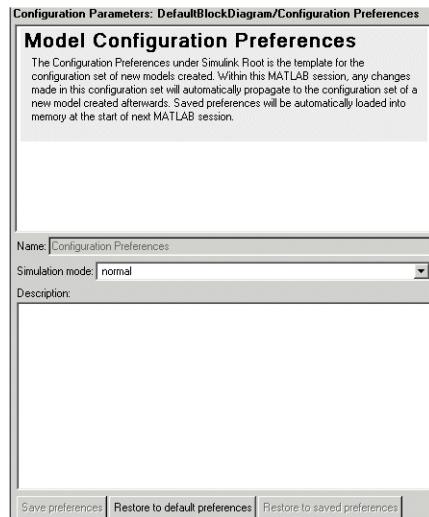
The simulation mode used to simulate the model in this configuration. The options are normal (see “Simulation Basics” on page 7-2), accelerator (see “The Simulink Accelerator” in the online Simulink documentation), or external mode. For information on external mode, see “External Mode” in the “Real-Time Workshop User’s Guide” (available on the MathWorks Web site if Real-Time Workshop is not installed on your system).

### Description

A description of this configuration. You can use this field to enter information pertinent to using this configuration.

## Model Configuration Preferences Dialog Box

The Model Configuration Preferences dialog box allows you to specify the settings for the configuration parameters of newly created models. The dialog box appears when you select Configuration Preferences under the Simulink Root node in the **Model Hierarchy** pane of the Model Explorer.



The dialog box has the following fields.

### Name

Name of the model preferences configuration. You can change the name of the configuration by editing this field.

### Simulation mode

The preferred mode used to simulate a model. The options are normal (“Simulation Basics” on page 7-2), accelerator (see “The Simulink Accelerator” in the online Simulink documentation), or external mode (see the “Real-Time Workshop User’s Guide” in the Real-Time Workshop documentation available on the MathWorks website).

**Description**

A description of the model configuration preferences. You can use this field to enter information pertinent to the preferences.

**Save Preferences**

Select this button to save the current configuration preferences.

**Restore to Default Preferences**

Select this button to restore the default configuration settings for creating new models.

**Restore to Saved Preferences**

Select this button to restore the preferences to the settings in effect the last time the preferences were saved. This option overrides any changes that you have made to the preferences since the beginning of the session or since the last time the preferences were restored.

## Configuration Parameters Dialog Box

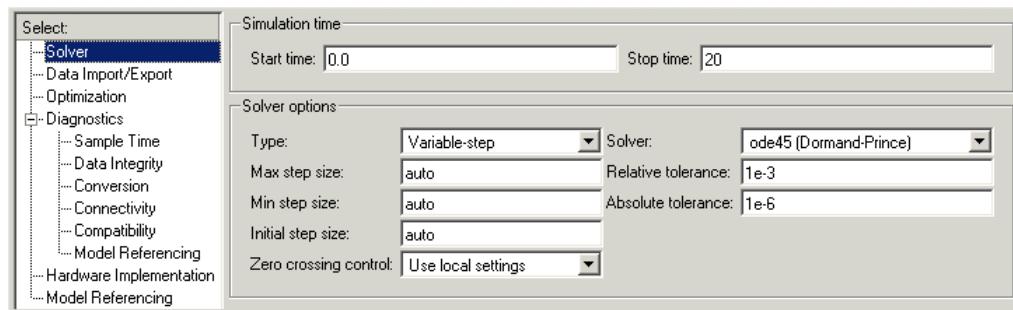
The **Configuration Parameters** dialog box allows you to modify settings for a model's active configuration set (see “Configuration Sets” on page 7-29).

---

**Note** You can also use the Model Explorer to modify settings for the active configuration set as well as for any other configuration set. See “The Model Explorer” on page 8-2 for more information.

---

To display the dialog box, select **Configuration Parameters** from the model editor's **Simulation** or context menu. The dialog box appears.



The dialog box groups the controls used to set the configuration parameters into various categories. To display the controls for a specific category, click the category in the **Select** tree on the left side of the dialog box.

See “The Configuration Parameters Dialog Box” in the online Simulink documentation for information on how to set specific configuration parameters:

In most cases, Simulink does not immediately apply a change that you have made on the dialog box. To apply a change, you must click either the **OK** or the **Apply** button at the bottom of the dialog box. The **OK** button applies all the changes you made and dismisses the dialog box. The **Apply** button applies the changes but leaves the dialog box open so that you can continue to make changes.

---

**Note** Each of the controls on the **Configuration Parameters** dialog box correspond to a configuration parameter that you can set via the `sim` and `simset` commands. See “Model Parameters” in the online Simulink reference for descriptions of these parameters. The description for each parameter specifies the **Configuration Parameters** dialog box prompt of the control that sets it. This allows you to determine the model parameter corresponding to a control on the **Configuration Parameters** dialog box.

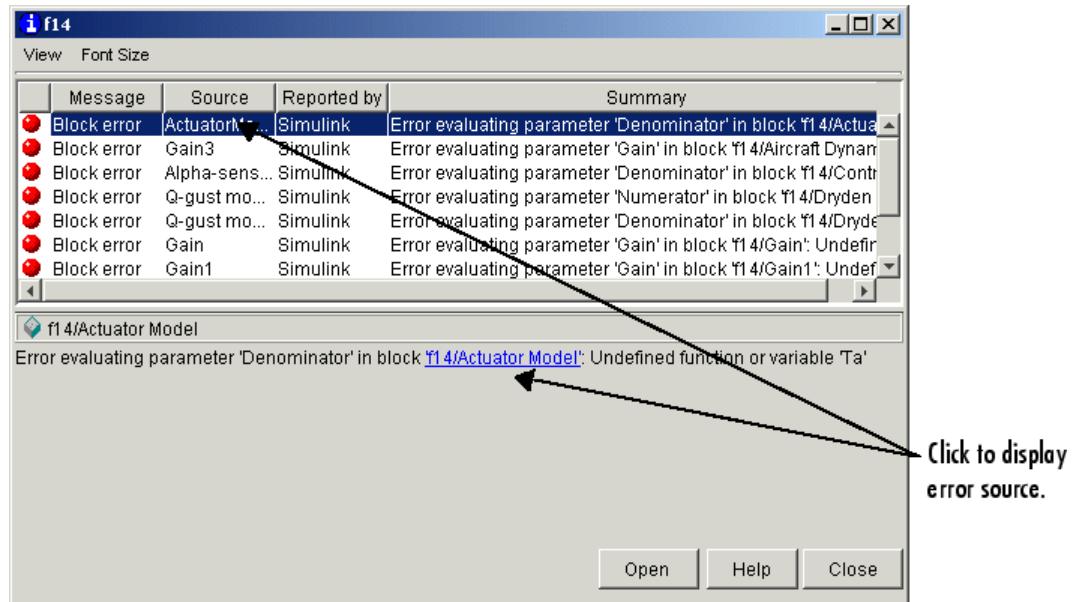
---

## Diagnosing Simulation Errors

If errors occur during a simulation, Simulink halts the simulation, opens the subsystems that caused the error (if necessary), and displays the errors in the Simulation Diagnostics Viewer. The following section explains how to use the viewer to determine the cause of the errors.

### Simulation Diagnostics Viewer

The viewer comprises an Error Summary pane and an Error Message pane.



### Error Summary Pane

The upper pane lists the errors that caused Simulink to terminate the simulation. The pane displays the following information for each error.

**Message.** Message type (for example, block error, warning, log)

**Source.** Name of the model element (for example, a block) that caused the error

**Reported by.** Component that reported the error (for example, Simulink, Stateflow, Real-Time Workshop, etc.)

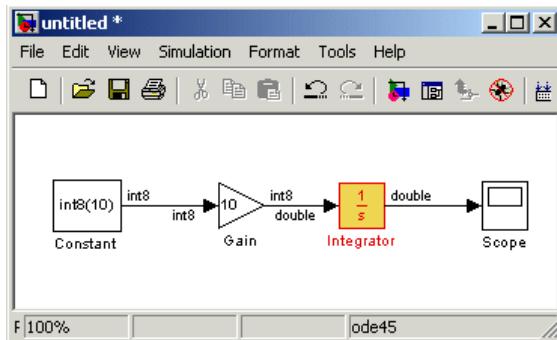
**Summary.** Error message, abbreviated to fit in the column

You can remove any of these columns of information to make more room for the others. To remove a column, select the viewer's **View** menu and uncheck the corresponding item.

### Error Message Pane

The lower pane initially contains the contents of the first error message listed in the top pane. You can display the contents of other messages by clicking their entries in the upper pane.

In addition to displaying the viewer, Simulink opens (if necessary) the subsystem that contains the first error source and highlights the source.



You can display the sources of other errors by clicking anywhere in the error message in the upper pane, by clicking the name of the error source in the error message (highlighted in blue), or by clicking the **Open** button on the viewer.

### Changing Font Size

To change the size of the font used to display errors, select **Font Size** from the viewer's menu bar. A menu of font sizes appears. Select the desired font size from the menu.

## Improving Simulation Performance and Accuracy

Simulation performance and accuracy can be affected by many things, including the model design and choice of configuration parameters.

The solvers handle most model simulations accurately and efficiently with their default parameter values. However, some models yield better results if you adjust solver parameters. Also, if you know information about your model's behavior, your simulation results can be improved if you provide this information to the solver.

### Speeding Up the Simulation

Slow simulation speed can have many causes. Here are a few:

- Your model includes a MATLAB Fcn block. When a model includes a MATLAB Fcn block, the MATLAB interpreter is called at each time step, drastically slowing down the simulation. Use the built-in Fcn block or Math Function block whenever possible.
- Your model includes an M-file S-function. M-file S-functions also cause the MATLAB interpreter to be called at each time step. Consider either converting the S-function to a subsystem or to a C-MEX file S-function.
- Your model includes a Memory block. Using a Memory block causes the variable-order solvers (`ode15s` and `ode113`) to be reset back to order 1 at each time step.
- The maximum step size is too small. If you changed the maximum step size, try running the simulation again with the default value (auto).
- Did you ask for too much accuracy? The default relative tolerance (0.1% accuracy) is usually sufficient. For models with states that go to zero, if the absolute tolerance parameter is too small, the simulation can take too many steps around the near-zero state values. See the discussion of error in “Maximum order” in the online Simulink documentation.
- The time scale might be too long. Reduce the time interval.
- The problem might be stiff, but you are using a nonstiff solver. Try using `ode15s`.

- The model uses sample times that are not multiples of each other. Mixing sample times that are not multiples of each other causes the solver to take small enough steps to ensure sample time hits for all sample times.
- The model contains an algebraic loop. The solutions to algebraic loops are iteratively computed at every time step. Therefore, they severely degrade performance. For more information, see “Algebraic Loops” on page 2-24.
- Your model feeds a Random Number block into an Integrator block. For continuous systems, use the Band-Limited White Noise block in the Sources library.

## Improving Simulation Accuracy

To check your simulation accuracy, run the simulation over a reasonable time span. Then, either reduce the relative tolerance to 1e-4 (the default is 1e-3) or reduce the absolute tolerance and run it again. Compare the results of both simulations. If the results are not significantly different, you can feel confident that the solution has converged.

If the simulation misses significant behavior at its start, reduce the initial step size to ensure that the simulation does not step over the significant behavior.

If the simulation results become unstable over time,

- Your system might be unstable.
- If you are using `ode15s`, you might need to restrict the maximum order to 2 (the maximum order for which the solver is A-stable) or try using the `ode23s` solver.

If the simulation results do not appear to be accurate,

- For a model that has states whose values approach zero, if the absolute tolerance parameter is too large, the simulation takes too few steps around areas of near-zero state values. Reduce this parameter value or adjust it for individual states in the Integrator dialog box.
- If reducing the absolute tolerances does not sufficiently improve the accuracy, reduce the size of the relative tolerance parameter to reduce the acceptable error and force smaller step sizes and more steps.



# Exploring, Searching, and Browsing Models

---

The following sections describe tools that enable you to quickly navigate to any point in a model and find and modify objects in a model.

## **The Model Explorer (p. 8-2)**

How to use the Model Explorer to find, display, and modify model contents.

## **The Finder (p. 8-16)**

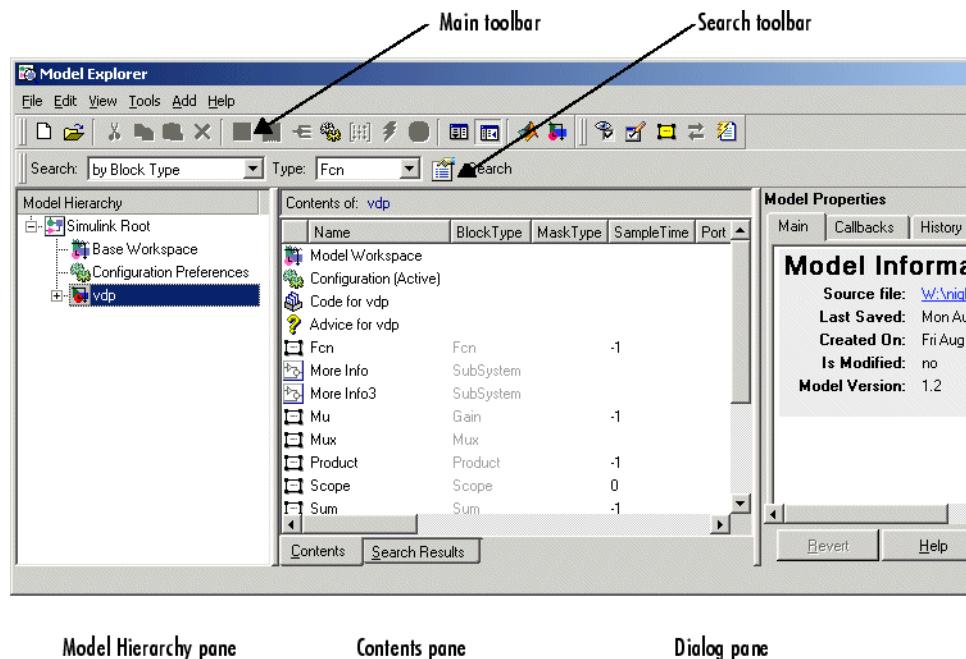
How to use the Simulink Finder to locate blocks, states, and other objects in a model, using search criteria that you specify.

## **The Model Browser (p. 8-22)**

How to navigate quickly to any point in a model's block hierarchy.

## The Model Explorer

The Model Explorer allows you to quickly locate, view, and change elements of a Simulink model or Stateflow chart. To display the Model Explorer, select **Model Explorer** from the Simulink **View** menu or select an object in the block diagram and select **Explore** from its context menu. The Model Explorer appears.



The Model Explorer includes the following components:

- **Model Hierarchy** pane (see “Model Hierarchy Pane” on page 8-3)
- **Contents** pane (see “Contents Pane” on page 8-5)
- **Dialog** pane (see “Dialog Pane” on page 8-9)
- **Main** toolbar (see “Main Toolbar” on page 8-9)
- **Search** bar (see “Search Bar” on page 8-12)

You can use the Model Explorer's **View** menu to hide the **Dialog** pane and the toolbars, thereby making more room for the other panes.

## Setting the Model Explorer's Font Size

To increase the size of the font used by the Model Explorer to display text, select **Increase Font Size** from the Model Explorer's **View** menu or type **Ctrl+**. To decrease the font size, select **Decrease Font Size** from the menu or type **Ctrl-**. The change remains in effect across Simulink sessions.

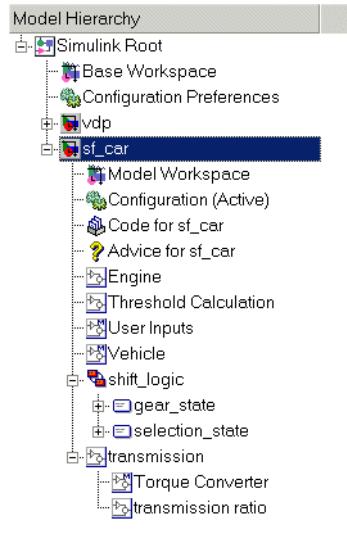
---

**Note** Increasing or decreasing the Model Explorer's font size also correspondingly increases or decreases the font size used by Simulink dialog boxes.

---

## Model Hierarchy Pane

The **Model Hierarchy** pane displays a tree-structured view of the Simulink model hierarchy.



## Simulink Root

The first node in the view represents the Simulink root. Expanding the root node displays nodes representing the MATLAB workspace (the Simulink base workspace) and each model and library loaded in the current session.

## Base Workspace

This node represents the MATLAB workspace. The MATLAB workspace is the base workspace for Simulink models. Variables defined in this workspace are visible to all open Simulink models, i.e., to all models whose nodes appear beneath the **Base Workspace** node in the **Model Hierarchy** pane.

## Configuration Preferences

If you check the **Show Configuration Preferences** option on the Model Explorer's **View** menu, the expanded Simulink Root node also displays a Configuration Preferences node. Selecting this node displays the preferred model configuration (see “Configuration Sets” on page 7-29) for new models in the adjacent panes. You can change the preferred configuration by editing the displayed settings and using the **Model Configuration Preferences** dialog box to save the settings (see “Model Configuration Preferences Dialog Box” on page 7-34).

## Model Nodes

Expanding a model node displays nodes representing the model's configuration sets (see “Configuration Sets” on page 7-29), top-level subsystems, model references, and Stateflow charts. Expanding a node representing a subsystem displays its subsystems, if any. Expanding a node representing a Stateflow chart displays the chart's top-level states. Expanding a node representing a state shows its substates.

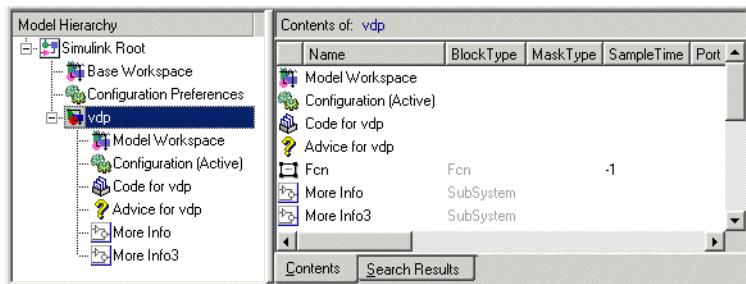
## Displaying Node Contents

To display the contents of an object displayed in the **Model Hierarchy** pane (e.g., a model or configuration set) in the adjacent **Contents** pane, select the object. To open a graphical object (e.g., a model, subsystem, or chart) in an editor window, right-click the object. A context menu appears. Select **Open** from the context menu. To open an object's properties dialog, select **Properties** from the object's context menu or from the **Edit** menu. See “Configuration Sets” on page 7-29 for information on using the **Model**

**Hierarchy** pane to delete, move, and copy configuration sets from one model to another.

## Contents Pane

The **Contents** pane displays either of two tabular views selectable by tabs. The **Contents** tab displays the contents of the object selected in the **Model Hierarchy** pane. The **Search Results** tab displays the results of a search operation (see “Search Bar” on page 8-12).

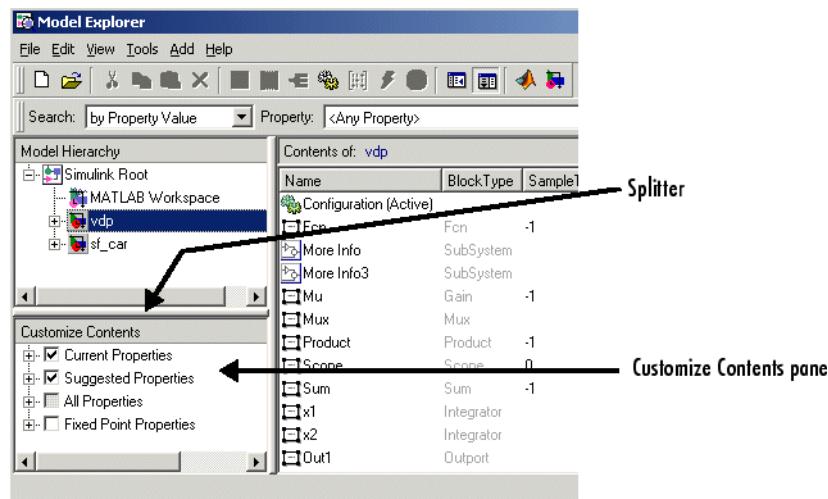


In both views, the table rows correspond to objects (e.g., blocks or states); the table columns, to object properties (e.g., name and type). The table cells display the values of the properties of the objects contained by the object selected in the **Model Hierarchy** pane or found by a search operation.

The objects and properties displayed in the **Contents** pane depend on the type of object (e.g., subsystem, chart, or configuration set) selected in the **Model Hierarchy** pane. For example, if the object selected in the **Model Hierarchy** pane is a model or subsystem, the **Contents** pane by default displays the name and type of the top-level blocks contained by that model or subsystem. If the selected object is a Stateflow chart or state, the **Contents** pane by default shows the name, scope, and other properties of the events and data that make up the chart or state.

## Customize Contents Pane

The **Customize Contents** pane allows you to select the properties that the **Contents** pane displays for the object selected in the **Model Hierarchy** pane. When visible, the pane appears in the lower-left corner of the Model Explorer window.



A splitter divides the **Customize Contents** pane from the **Model Hierarchy** pane above it. Drag the splitter up or down to adjust the relative size of the two panes.

The **Customize Contents** pane contains a tree-structured property list. The list's top-level nodes group object properties into the following categories:

- Current Properties

Properties that the **Contents** pane currently displays.

- All Properties

Properties of the contents of all models displayed in the Model Explorer thus far in this session.

- Fixed Point Properties

Fixed-point properties of blocks.

By default, the properties currently displayed in the **Contents** pane are the suggested properties for the currently selected model. The **Customize Contents** pane allows you to perform the following customizations:

- To display additional properties of the selected model, expand the **All Properties** node, if necessary, and check the desired properties.
- To delete some but not all properties from the **Contents** pane, expand the **Current Properties** node, if necessary, and uncheck the properties that you do not want to appear in the **Contents** pane.
- To delete all properties from the **Contents** pane (except the selected object's name), uncheck **Current Properties**.
- To add or remove fixed-point block properties from the **Contents** pane, check or uncheck **Fixed Point Properties**.

### Customizing the Contents Pane

The Model Explorer's **View** menu allows you to control the type of objects and properties displayed in the **Contents** pane.

- To display only object names in the **Contents** pane, uncheck the **Show Properties** item on the **View** menu.
- To customize the set of properties displayed in the **Contents** pane, select **Customize Contents** from the **View** menu or click the **Customize Contents** button on the Model Explorer's main toolbar (see "Main Toolbar" on page 8-9). The **Customize Contents** pane appears. Use the pane to select the properties you want the **Contents** pane to display.
- To specify the types of subsystem or chart contents displayed in the **Contents** pane, select **List View Options** from the **View** menu. A menu of object types appears. Check the types that you want to be displayed (e.g., **Blocks** and **Named Signals/Connections** or **All Simulink Objects** for models and subsystems).

### Reordering the Contents Pane

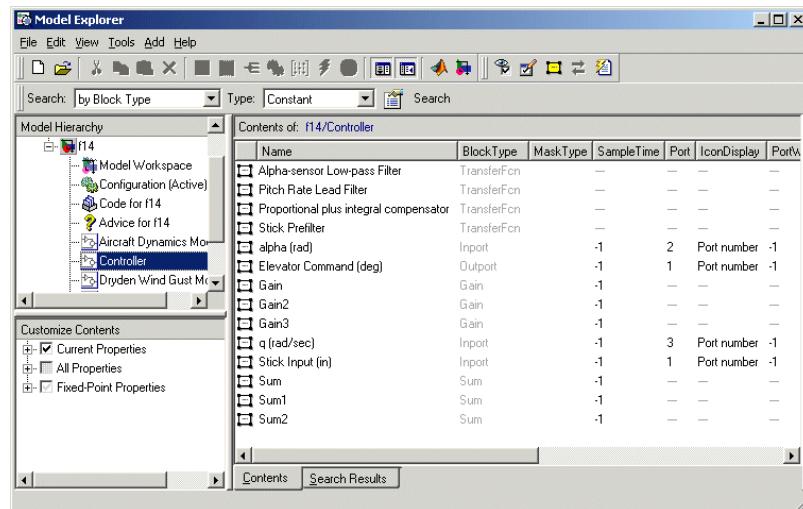
The **Contents** pane by default displays its contents in ascending order by name. To order the contents in ascending order by any other displayed property, click the head of the column that displays the property. To change

the order from ascending to descending, or vice versa, click the head of the property column that determines the current order.

## Marking Nonexistent Properties

Some of the properties that the Contents pane is configured to display may not apply to all the objects currently listed in the Contents pane. You can configure the Model Explorer to indicate the inapplicable properties.

To do this, select **Mark Nonexistent Properties** from the Model Explorer's **View** menu. The Model Explorer now displays dashes for the values of properties that do not apply to the objects displayed in the **Contents** pane.



## Changing Property Values

You can change modifiable properties displayed in the **Contents** pane (e.g., a block's name) by editing the displayed value. To edit a displayed value, first select the row that contains it. Then click the value. An edit control replaces the displayed value (e.g., an edit field for text values or a pull-down list for a range of values). Use the edit control to change the value of the selected property.

To assign the same property value to multiple objects displayed in the **Contents** pane, select the objects and then change one of the selected objects to have the new property value. The Model Explorer assigns the new property value to the other selected objects as well.

## Dialog Pane

The **Dialog** pane displays the dialog view of the object selected in the **Contents** pane, e.g., a block or a configuration subset. You can use the **Dialog** pane to view and change the selected object's properties. To show or hide this pane, select the **Dialog View** menu from the Model Explorer's **View** menu or the **Dialog View** button on the Model Explorer's main toolbar (see "Main Toolbar" on page 8-9).

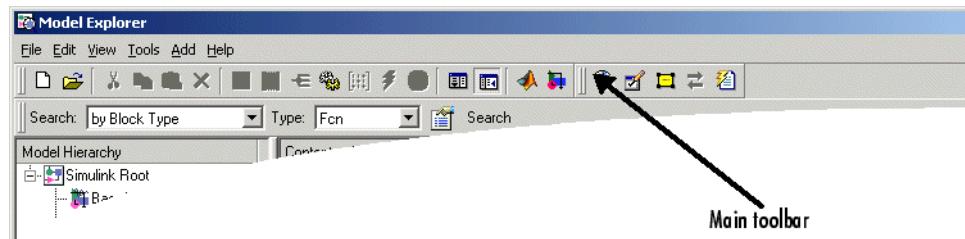
---

**Note** Unlike the **Contents** pane, which allows you to set the properties of multiple objects at a time, the **Dialog** pane allows you to set the properties of only one object at a time. For example, in the **Dialog** pane, you can edit the properties of the object currently selected in the **Contents** pane.

---

## Main Toolbar

The Model Explorer's main toolbar appears near the top of the Model Explorer window under the Model Explorer's menu.



The toolbar contains buttons that select commonly used Model Explorer commands:

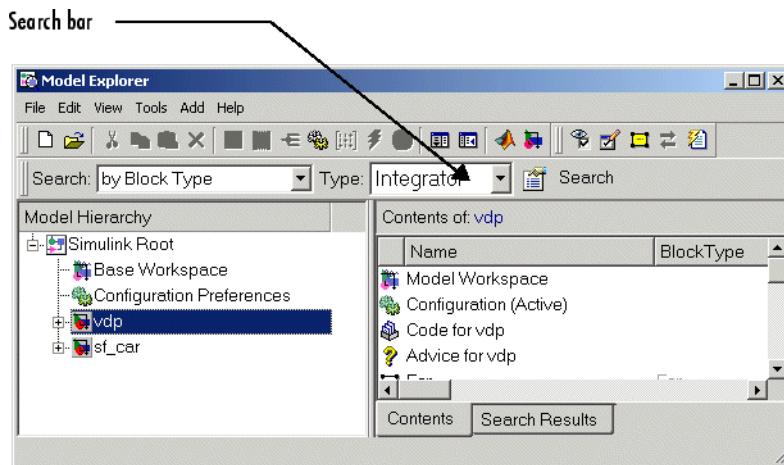
Button	Usage
	Create a new model.
	Open an existing model.
	Cut the objects (e.g., variables) selected in the <b>Contents</b> pane from the object (e.g., a workspace) selected in the <b>Model Hierarchy</b> pane. Save a copy of the object on the system clipboard.
	Copy the objects selected in the <b>Contents</b> pane to the system clipboard.
	Paste objects from the clipboard into the object selected in the Model Explorer's <b>Model Hierarchy</b> pane.
	Delete the objects selected in the <b>Contents</b> pane from the object selected in the <b>Model Hierarchy</b> pane.
	Add a MATLAB variable to the workspace selected in the <b>Model Hierarchy</b> pane.
	Add a Simulink.Parameter object to the workspace selected in the <b>Model Hierarchy</b> pane.
	Add a Simulink.Signal object to the workspace selected in the <b>Model Hierarchy</b> pane.
	Add a configuration set to the model selected in the <b>Model Hierarchy</b> pane.
	Add a Stateflow datum to the machine or chart selected in the <b>Model Hierarchy</b> pane.
	Add a Stateflow event to the machine or chart selected in the <b>Model Hierarchy</b> pane or to the state selected in the Model Explorer.

Button	Usage
	Add a code generation target to the model selected in the <b>Model Hierarchy</b> pane.
	Turn the Model Explorer's <b>Dialog</b> pane on or off.
	Customize the Model Explorer's <b>Contents</b> pane.
	Bring the MATLAB desktop to the front.
	Display the Simulink Library Browser.

To show or hide the main toolbar, select **Main Toolbar** from the Model Explorer's **View** menu.

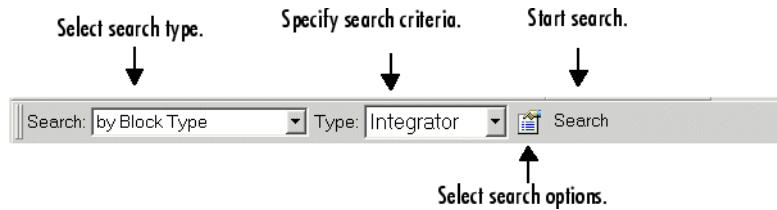
### Search Bar

The Model Explorer's search bar allows you to select, configure, and initiate searches of the object selected in the **Model Hierarchy** pane. It appears at the top of the Model Explorer window.



To show or hide the search bar, check or uncheck **Search Bar** in the Model Explorer's **View > Toolbars** menu.

The search bar includes the following controls:



#### Search Type

Specifies the type of search to be performed. Options include:

- by Block Type

Search for blocks of a specified block type. Selecting this search type causes the search bar to display a block type list control that allows you to select the target block type from the types contained by the currently selected model.

- **by Property Name**

Searches for objects that have a specified property. Selecting this search type causes the search bar to display a control that allows you to specify the target property's name by selecting from a list of properties that objects in the search domain can have.

- **by Property Value**

Searches for objects whose property matches a specified value. Selecting this search type causes the search bar to display controls that allow you to specify the name of the property, the value to be matched, and the type of match (equals, less than, greater than, etc.).

- **for Fixed Point**

Searches a model for all blocks that support fixed-point computations.

- **by Name**

Searches a model for all objects that have the specified string in the name of the object.

- **by Stateflow Type**

Searches for Stateflow objects of a specified type.

- **for Library Links**

Searches for library links in the current model.

- **by Class**

Searches for Simulink objects of a specified class.

- **for Model References**

Searches a model for references to other models.

- **by Dialog Prompt**

Searches a model for all objects whose dialogs contain a specified prompt.

- **by String**

Searches a model for all objects in which a specified string occurs.

## Search Options

Specifies options that apply to the current search. The options include:

- **Search Current System and Below**

Search the current system and the subsystems that it includes directly or indirectly.

- **Look Inside Masked Subsystems**

Search includes masked subsystems.

- **Look Inside Linked Subsystems**

Search includes linked subsystems.

- **Match Whole String**

Do not allow partial string matches, e.g., do not allow `sub` to match substring.

- **Match Case**

Consider case when matching strings, e.g., `Gain` does not match `gain`.

- **Regular Expression**

The Model Explorer considers a string to be matched as a regular expression.

- **Evaluate Property Values During Search**

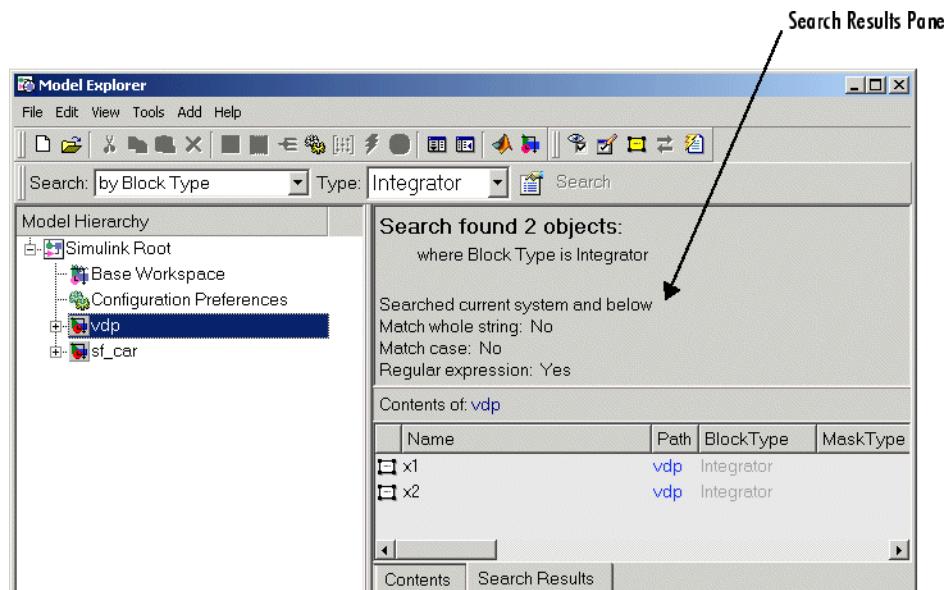
This option applies only for searches by property value. If enabled, the option causes the Model Explorer to evaluate the value of each property as a MATLAB expression and compare the result to the search value. If disabled (the default), the Model Explorer compares the unevaluated property value to the search value.

- **Refine Search**

Causes the next search operation to search for objects that meet both the original and new search criteria (see “Refining a Search” on page 8-15).

## Search Button

Initiates the search specified by the current settings of the search bar on the object selected in the Model Explorer's **Model Hierarchy** pane. The Model Explorer displays the results of the search in the tabbed **Search Results** pane.



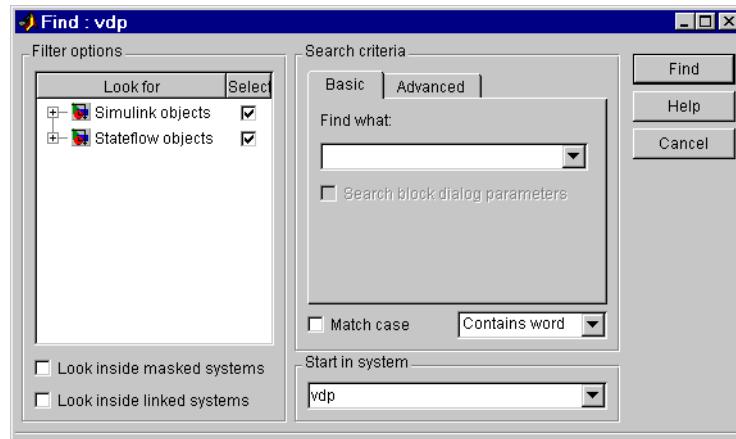
You can edit the results displayed in the **Search Results** pane. For example, to change all objects found by a search to have the same property value, select the objects in the **Search Results** pane and change one of them to have the new property value.

## Refining a Search

To refine the previous search, check the **Refine Search** option on the search bar's **Search Options** menu. A **Refine** button replaces the **Search** button on the search bar. Use the search bar to define new search criteria and then click the **Refine** button. The Model Explorer searches for objects that match the previous search criteria and the new criteria.

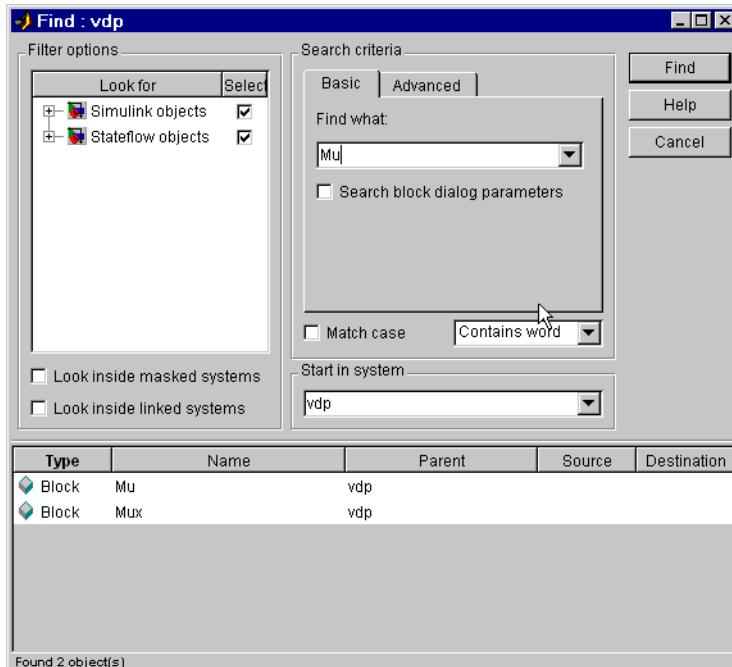
## The Finder

The Finder locates blocks, signals, states, or other objects in a model. To display the Finder, select **Find** from the **Edit** menu. The **Find** dialog box appears.



Use the **Filter options** (see “Filter Options” on page 8-18) and **Search criteria** (see “Search Criteria” on page 8-18) panels to specify the characteristics of the object you want to find. Next, if you have more than one system or subsystem open, select the system or subsystem where you want the search to begin from the **Start in system** list. Finally, click the **Find** button. Simulink searches the selected system for objects that meet the criteria you have specified.

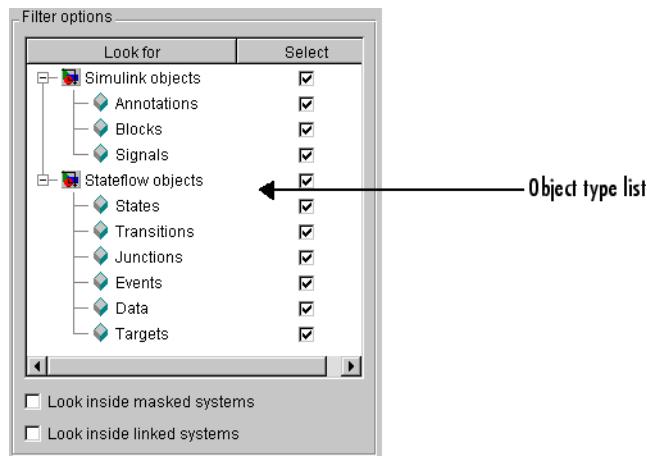
Any objects that satisfy the criteria appear in the results panel at the bottom of the dialog box.



You can display an object by double-clicking its entry in the search results list. Simulink opens the system or subsystem that contains the object (if necessary) and highlights and selects the object. To sort the results list, click any of the buttons at the top of each column. For example, to sort the results by object type, click the **Type** button. Clicking a button once sorts the list in ascending order, clicking it twice sorts it in descending order. To display an object's parameters or properties, select the object in the list. Then press the right mouse button and select **Parameter** or **Properties** from the resulting context menu.

## Filter Options

The **Filter options** panel allows you to specify the kinds of objects to look for and where to search for them.



### Object type list

The object type list lists the types of objects that Simulink can find. By clearing a type, you can exclude it from the Finder's search.

### Look inside masked subsystem

Selecting this option causes Simulink to look for objects inside masked subsystems.

### Look inside linked systems

Selecting this option causes Simulink to look for objects inside subsystems linked to libraries.

### Search Criteria

The **Search criteria** panel allows you to specify the criteria that objects must meet to satisfy your search request.

## Basic

The **Basic** panel allows you to search for an object whose name and, optionally, dialog parameters match a specified text string. Enter the search text in the panel's **Find what** field. To display previous search text, select the drop-down list button next to the **Find what** field. To reenter text, click it in the drop-down list. Select **Search block dialog parameters** if you want dialog parameters to be included in the search.

## Advanced

The **Advanced** panel allows you to specify a set of as many as seven properties that an object must have to satisfy your search request.

Select	Property	Value
<input type="checkbox"/> (none)	▼	

To specify a property, enter its name in one of the cells in the **Property** column of the **Advanced** pane or select the property from the cell's property list. To display the list, select the down arrow button next to the cell. Next enter the value of the property in the **Value** column next to the property name. When you enter a property name, the Finder checks the check box next to the property name in the **Select** column. This indicates that the property is to be included in the search. If you want to exclude the property, clear the check box.

## Match case

Select this option if you want Simulink to consider case when matching search text against the value of an object property.

## Other match options

Next to the **Match case** option is a list that specifies other match options that you can select.

- Match whole word

Specifies a match if the property value and the search text are identical except possibly for case.

- Contains word

Specifies a match if a property value includes the search text.

- Regular expression

Specifies that the search text should be treated as a regular expression when matched against property values. The following characters have special meanings when they appear in a regular expression.

Character	Meaning
^	Matches start of string.
\$	Matches end of string.
.	Matches any character.
\	Escape character. Causes the next character to have its ordinary meaning. For example, the regular expression \.. matches .a and .2 and any other two-character string that begins with a period.
*	Matches zero or more instances of the preceding character. For example, ba* matches b, ba, baa, etc.
+	Matches one or more instances of the preceding character. For example, ba+ matches ba, baa, etc.
[ ]	Indicates a set of characters that can match the current character. A hyphen can be used to indicate a range of characters. For example, [a-zA-Z0-9_]+ matches foo_bar1 but not foo\$bar. A ^ indicates a match when the current character is not one of the following characters. For example, [^0-9] matches any character that is not a digit.

Character	Meaning
\w	Matches a word character (same as [ a-zA-Z0-9]).
\W	Matches a nonword character (same as [^a-zA-Z0-9]).
\d	Matches a digit (same as [0-9]).
\D	Matches a nondigit (same as [^0-9]).
\s	Matches white space (same as [ \t\r\n\f]).
\S	Matches nonwhite space (same as [^ \t\r\n\f]).
\<WORD\>	Matches WORD where WORD is any string of word characters surrounded by white space.

## The Model Browser

The Model Browser enables you to

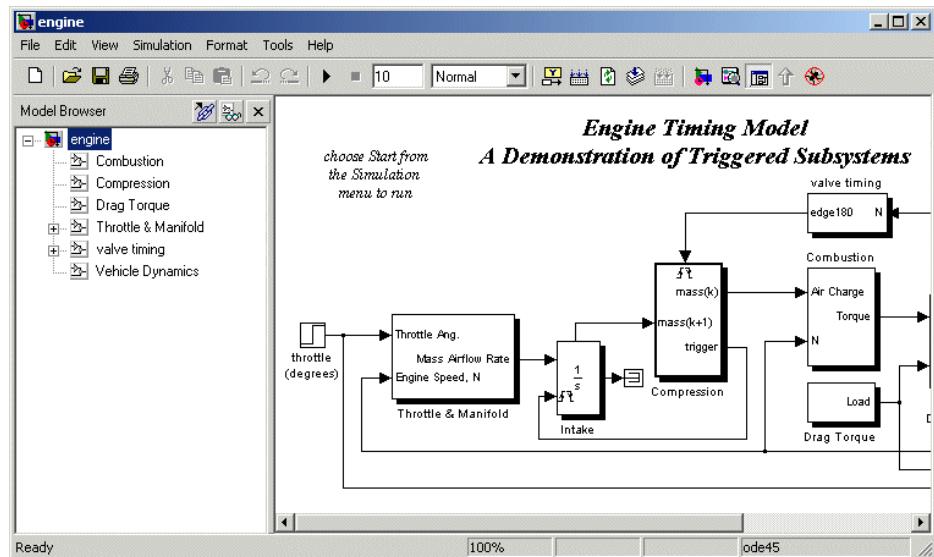
- Navigate a model hierarchically
- Open systems in a model
- Determine the blocks contained in a model

---

**Note** The browser is available only on Microsoft Windows platforms.

---

To display the Model Browser, select **Model Browser Options > Model Browser** from the Simulink **View** menu.



The model window splits into two panes. The left pane displays the browser, a tree-structured view of the block diagram displayed in the right pane.

---

**Note** The **Browser initially visible** preference causes Simulink to open models by default in the Model Browser. To set this preference, select **Preferences** from the Simulink **File** menu.

---

The top entry in the tree view corresponds to your model. A button next to the model name allows you to expand or contract the tree view. The expanded view shows the model's subsystems. A button next to a subsystem indicates that the subsystem itself contains subsystems. You can use the button to list the subsystem's children. To view the block diagram of the model or any subsystem displayed in the tree view, select the subsystem. You can use either the mouse or the keyboard to navigate quickly to any subsystem in the tree view.

## Navigating with the Mouse

Click any subsystem visible in the tree view to select it. Click the + button next to any subsystem to list the subsystems that it contains. Click the button again to contract the entry.

## Navigating with the Keyboard

Use the up/down arrows to move the current selection up or down the tree view. Use the left/right arrow or +/- keys on your numeric keypad to expand an entry that contains subsystems.

## Showing Library Links

The Model Browser can include or omit library links from the tree view of a model. Use the Simulink **Preferences** dialog box to specify whether to display library links by default. To toggle display of library links, select **Show Library Links** from the **Model browser Options** submenu of the Simulink **View** menu.

## Showing Masked Subsystems

The Model Browser can include or omit masked subsystems from the tree view. If the tree view includes masked subsystems, selecting a masked subsystem in the tree view displays its block diagram in the diagram view.

Use the Simulink **Preferences** dialog box to specify whether to display masked subsystems by default. To toggle display of masked subsystems, select **Look Under Masks** from the **Model browser Options** submenu of the Simulink **View** menu.

# Creating Masked Subsystems

---

This section explains how to create custom user interfaces (masks) for Simulink subsystems.

## About Masks (p. 9-2)

An overview of masked subsystems that introduces you to key concepts.

## Masked Subsystem Example (p. 9-6)

Introduces you to masking by taking you step-by-step through the creation of a simple masked subsystem.

## Masking a Subsystem (p. 9-12)

General procedure for masking subsystems.

## Mask Editor (p. 9-14)

Detailed description of the Mask Editor.

## Linking Mask Parameters to Block Parameters (p. 9-32)

How to link a mask's parameters to the parameters of blocks behind the mask.

## About Masks

A mask is a custom user interface for a subsystem that hides the subsystem's contents, making it appear to the user as an atomic block with its own icon and parameter dialog box. The Simulink Mask Editor enables you to create a mask for any subsystem. Masking a subsystem allows you to

- Replace the parameter dialogs of a subsystem and its contents with a single parameter dialog with its own block description, parameter prompts, and help text
- Replace a subsystem's standard icon with a custom icon that depicts its purpose
- Prevent unintended modification of subsystems by hiding their contents behind a mask
- Create a custom block by encapsulating a block diagram that defines the block's behavior in a masked subsystem and then placing the masked subsystem in a library

---

**Note** You can also mask S-Function and Model blocks. The instructions for masking Subsystem blocks apply to S-Function and Model blocks as well except where noted.

---

## Mask Features

Masks can include any of the following features.

### Mask Icon

The mask icon replaces a subsystem's standard icon, i.e., it appears in a block diagram in place of the standard icon for a subsystem block. Simulink uses MATLAB code that you supply to draw the custom icon. You can use any MATLAB drawing command in the icon code. This gives you great flexibility in designing an icon for a masked subsystem.

## Mask Parameters

Simulink allows you to define a set of user-settable parameters for a masked subsystem. Simulink stores the value of a parameter in the mask workspace (see “Mask Workspace” on page 9-4) as the value of a variable whose name you specify. These associated variables allow you to link mask parameters to specific parameters of blocks inside a masked subsystem (internal parameters) such that setting a mask parameter sets the associated block parameter (see “Linking Mask Parameters to Block Parameters” on page 9-32).

---

**Note** If you intend to allow the user to specify the model referenced by a masked Model block or a Model block in a masked subsystem, you must ensure that the mask requires that the user specify the model name as a literal value rather than as a workspace variable. This is because Simulink updates model reference targets before evaluating block parameters. The recommended way to force the user to specify the model name as a literal is to use a pop-up control on the mask to specify the model name. See “Pop-Up Control” on page 9-24 for more information.

---

## Mask Parameter Dialog Box

The mask parameter dialog box contains controls that enable a user to set the values of the masks parameters and hence the values of any internal parameters linked to the mask parameters.

The mask parameter dialog box replaces the subsystem’s standard parameter dialog box, i.e., clicking on the masked subsystem’s icon causes the mask dialog box to appear instead of the standard parameter dialog box for a Subsystem block

---

**Note** Use the ‘mask’ option of the `open_system` command to open a block’s mask dialog box at the MATLAB command line or in an M program.

---

You can customize every feature of the mask dialog box, including which parameters appear on the dialog box, the order in which they appear, parameter prompts, the controls used to edit the parameters, and the

parameter callbacks (code used to process parameter values entered by the user).

## Mask Initialization Code

The initialization code is MATLAB code that you specify and that Simulink runs to initialize the masked subsystem at the start of a simulation run. You can use the initialization code to set the initial values of the masked subsystem's mask parameters.

## Mask Workspace

Simulink associates a workspace with each masked subsystem that you create. Simulink stores the current values of the subsystem's parameters in the workspace as well as any variables created by the block's initialization code and parameter callbacks. You can use model and mask workspace variables to initialize a masked subsystem and to set the values of blocks inside the masked subsystem, subject to the following rules.

- The **Permit Hierarchical Resolution** option of the subsystem is set to All or ParametersOnly (see "Permit Hierarchical Resolution" in the online documentation for the Subsystem block).
- A block parameter expression can refer only to variables defined in the mask workspaces of the subsystem or nested subsystems that contain the block or in the model's workspace.
- A valid reference to a variable defined on more than one level in the model hierarchy resolves to the most local definition.

For example, suppose that model M contains masked subsystem A, which contains masked subsystem B. Further suppose that B refers to a variable *x* that exists in both A's and M's workspaces. In this case, the reference resolves to the value in A's workspace.

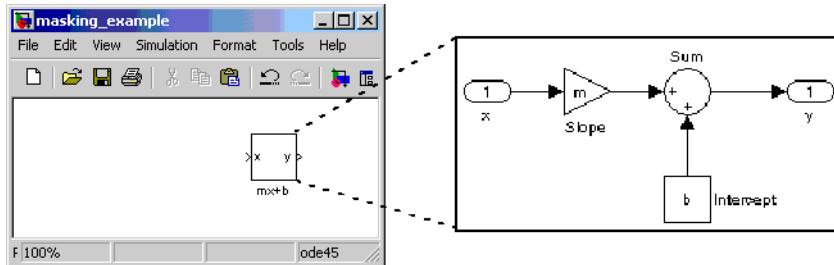
- A masked subsystem's initialization code can refer only to variables in its local workspace.
- The mask workspace of a Model block is not visible to the model that it references. Any variables used by the referenced model must resolve to workspaces defined in the referenced model or to the base (i.e., the MATLAB) workspace.

## **Creating Masks**

See “Masking a Subsystem” on page 9-12 for an overview of the process of creating a masked subsystem. See “Masked Subsystem Example” on page 9-6 for an example of the process.

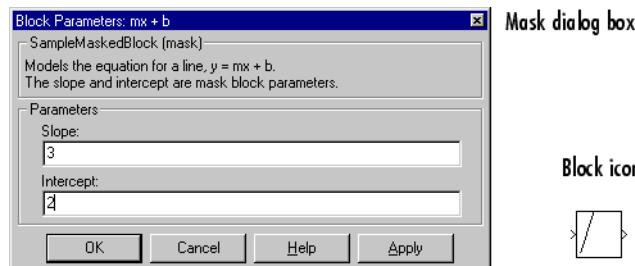
## Masked Subsystem Example

This simple subsystem models the equation for a line,  $y = mx + b$ .



Ordinarily, when you double-click a Subsystem block, the Subsystem block opens, displaying its blocks in a separate window. The  $mx + b$  subsystem contains a Gain block, named Slope, whose **Gain** parameter is specified as  $m$ , and a Constant block, named Intercept, whose **Constant value** parameter is specified as  $b$ . These parameters represent the slope and intercept of a line.

This example creates a custom dialog box and icon for the subsystem. One dialog box contains prompts for both the slope and the intercept. After you create the mask, double-click the Subsystem block to open the mask dialog box. The mask dialog box and icon look like this:



Block icon



A user enters values for **Slope** and **Intercept** in the mask dialog box. Simulink makes these values available to all the blocks in the underlying subsystem. Masking this subsystem creates a self-contained functional unit with its own application-specific parameters, **Slope** and **Intercept**. The mask maps these *mask parameters* to the generic parameters of the underlying

blocks. The complexity of the subsystem is encapsulated by a new interface that has the look and feel of a built-in Simulink block.

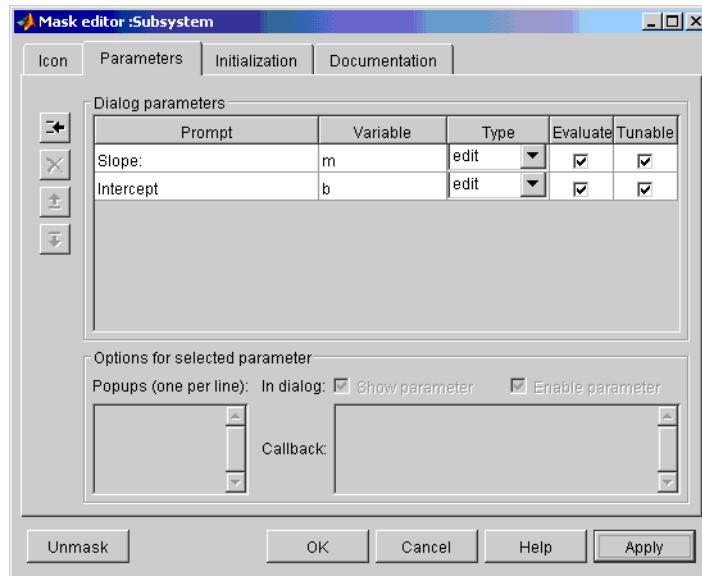
To create a mask for this subsystem, you need to

- Specify the prompts for the mask dialog box parameters. In this example, the mask dialog box has prompts for the slope and intercept.
- Specify the variable name used to store the value of each parameter.
- Enter the documentation of the block, consisting of the block description and the block help text.
- Specify the drawing command that creates the block icon.
- Specify the commands that provide the variables needed by the drawing command (there are none in this example).

## **Creating Mask Dialog Box Prompts**

To create the mask for this subsystem, select the Subsystem block and choose **Mask Subsystem** from the **Edit** menu.

This example primarily uses the Mask Editor's **Parameters** pane to create the masked subsystem's dialog box.



The Mask Editor enables you to specify these attributes of a mask parameter:

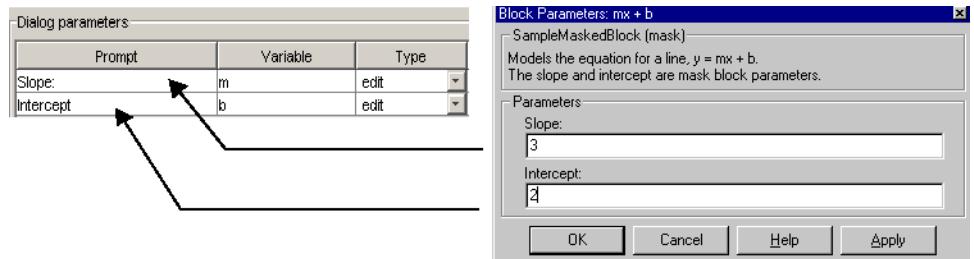
- Prompt, the text label that describes the parameter
- Control type, the style of user interface control that determines how parameter values are entered or selected
- Variable, the name of the variable that stores the parameter value

Generally, it is convenient to refer to masked parameters by their prompts. In this example, the parameter associated with slope is referred to as the **Slope** parameter, and the parameter associated with intercept is referred to as the **Intercept** parameter.

The slope and intercept are defined as edit controls. This means that the user types values into edit fields in the mask dialog box. These values are stored in variables in the *mask workspace*. Masked blocks can access variables only in the mask workspace. In this example, the value entered for the slope is

assigned to the variable  $m$ . The Slope block in the masked subsystem gets the value for the slope parameter from the mask workspace.

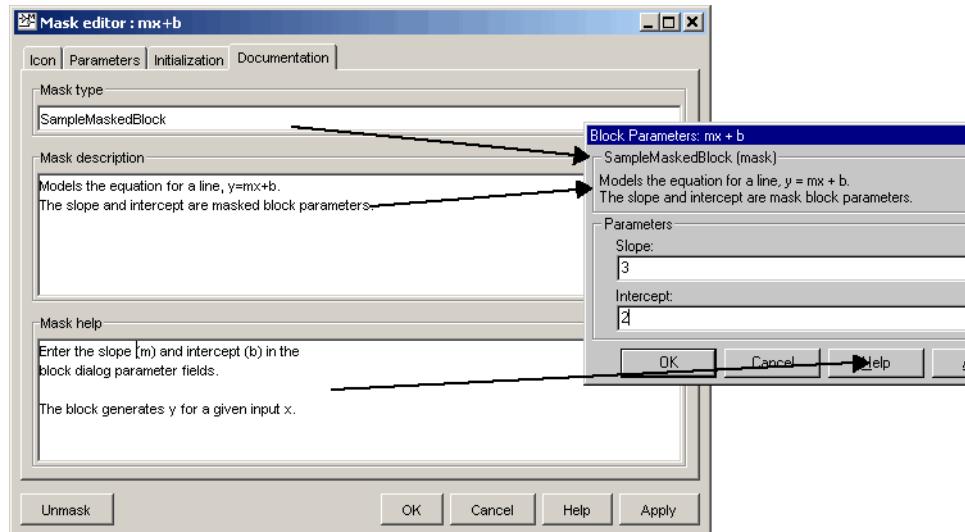
This figure shows how the slope parameter definitions in the Mask Editor map to the actual mask dialog box parameters.



After you create the mask parameters for slope and intercept, click **OK**. Then double-click the Subsystem block to open the newly constructed dialog box. Enter 3 for the **Slope** and 2 for the **Intercept** parameter.

### Creating the Block Description and Help Text

The mask type, block description, and help text are defined on the **Documentation** pane. For this sample masked block, the pane looks like this.

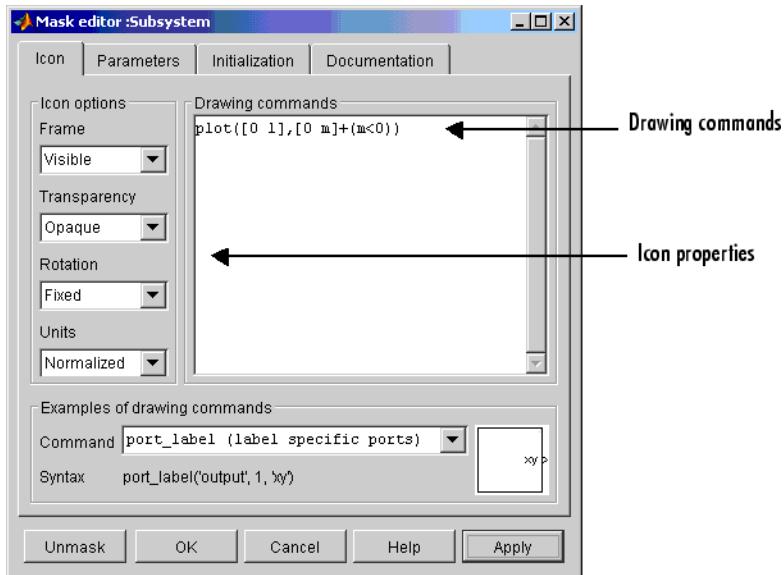


### Creating the Block Icon

So far, we have created a customized dialog box for the  $mx + b$  subsystem. However, the Subsystem block still displays the generic Simulink subsystem icon. An appropriate icon for this masked block is a plot that indicates the slope of the line. For a slope of 3, that icon looks like this.



The block icon is defined on the **Icon** pane. For this block, the **Icon** pane looks like this.



The drawing command

```
plot([0 1],[0 m]+(m<0))
```

plots a line from  $(0,0)$  to  $(1,m)$ . If the slope is negative, Simulink shifts the line up by 1 to keep it within the visible drawing area of the block.

The drawing commands have access to all the variables in the mask workspace. As you enter different values of slope, the icon updates the slope of the plotted line.

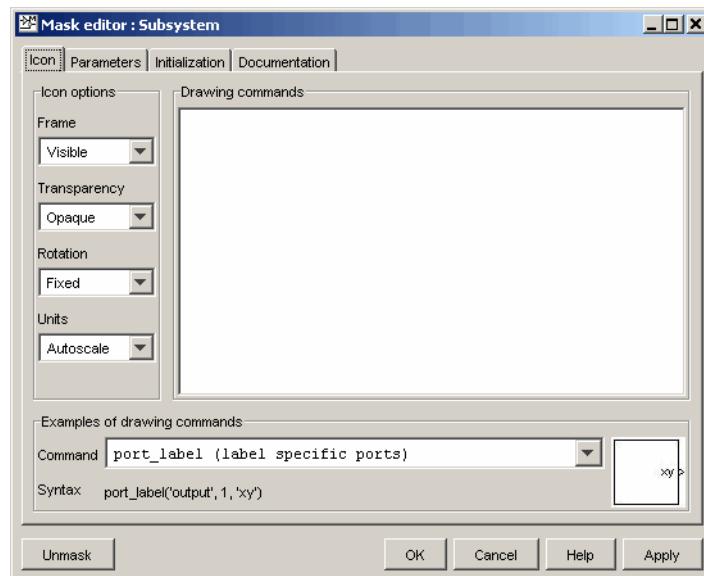
Select **Normalized** as the **Drawing coordinates** parameter, located at the bottom of the list of icon properties, to specify that the icon be drawn in a frame whose bottom-left corner is  $(0,0)$  and whose top-right corner is  $(1,1)$ . See “Icon Pane” on page 9-16 for more information.

## Masking a Subsystem

To mask a subsystem:

- 1 Select the subsystem.
- 2 Select **Edit Mask** from the **Edit** menu of the model window or from the block's context menu. (Right-click the subsystem block to display its context menu.)

The Mask Editor appears.



See “Mask Editor” on page 9-14 for a detailed description of the Mask Editor.

- 3 Use the Mask Editor’s tabbed panes to perform any of the following tasks.
  - Create a custom icon for the masked subsystem (see “Icon Pane” on page 9-16).
  - Create parameters that allow a user to set subsystem options (see “Mask Editor” on page 9-14).

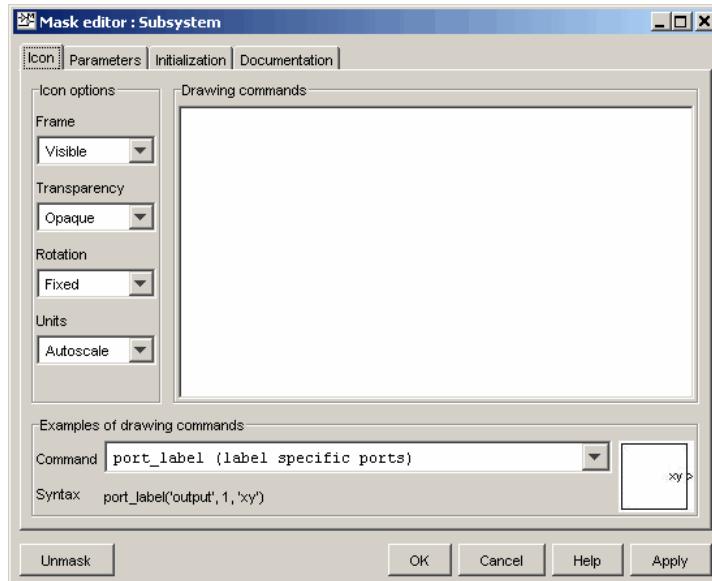
- Initialize the masked subsystem's parameters
  - Create online user documentation for the subsystem
- 4 Click **Apply** to apply the mask to the subsystem or **OK** to apply the mask and dismiss the Mask Editor.

## Mask Editor

The Mask Editor allows you to create or edit a subsystem's mask.

- To create a subsystem mask, select the subsystem block icon and then select **Mask Subsystem** from the **Edit** menu of the model window containing the subsystem's block.
- To edit an existing subsystem's mask, select the subsystem's block icon and then select **Edit Mask** from the **Edit** menu of the model window containing the subsystem's block.

A Mask Editor like the following appears in either case.



The Mask Editor contains a set of tabbed panes, each of which enables you to define a feature of the mask:

- The **Icon** pane enables you to define the block icon (see “Icon Pane” on page 9-16).

- The **Parameters** pane enables you to define and describe mask dialog box parameter prompts and name the variables associated with the parameters (see “Parameters Pane” on page 9-19).
- The **Initialization** pane enables you to specify initialization commands (see “Initialization Pane” on page 9-25).
- The **Documentation** pane enables you to define the mask type and specify the block description and the block help (see “Documentation Pane” on page 9-28).

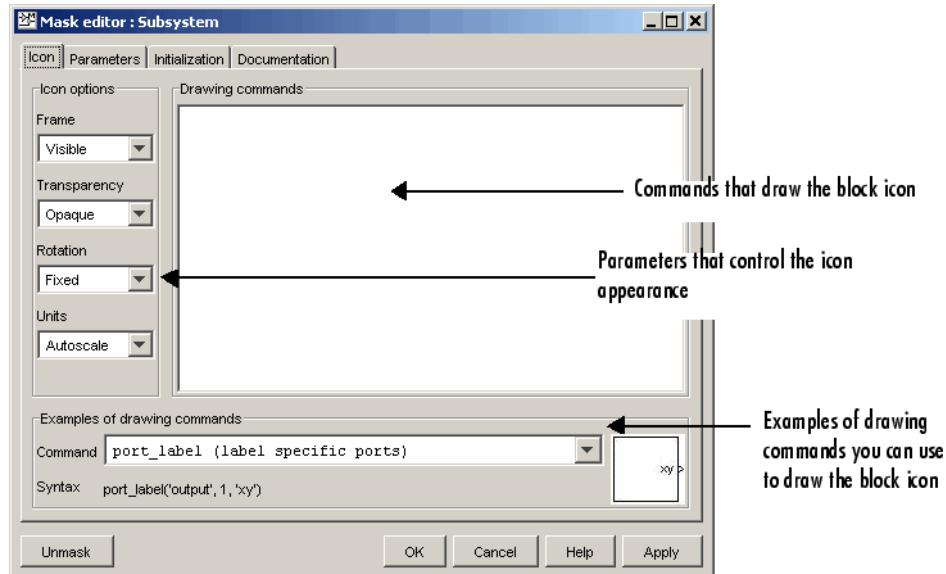
Five buttons appear along the bottom of the Mask Editor:

- The **Unmask** button deactivates the mask and closes the Mask Editor. While the model is still active, Simulink retains the mask information so that you can reactivate it. To reactivate the mask, select the block and choose **Mask Subsystem**. The Mask Editor opens, displaying the previous settings. When you close the model, Simulink discards the inactive mask information. If you want the mask information after this, you will need to recreate it the next time you open the model.
- The **OK** button applies the mask settings on all panes and closes the Mask Editor.
- The **Cancel** button closes the Mask Editor without applying any changes made since you last clicked the **Apply** button.
- The **Help** button displays the contents of this section.
- The **Apply** button creates or changes the mask using the information that appears on all masking panes. The Mask Editor remains open.

To see the system under the mask without unmasking it, select the Subsystem block, then select **Look Under Mask** from the **Edit** menu. This command opens the subsystem. The block’s mask is not affected.

## Icon Pane

The Mask Editor's **Icon** pane enables you to create icons that can contain descriptive text, state equations, images, and graphics.



The **Icon** pane contains the following controls.

### Drawing commands

This field allows you to enter commands that draw the block's icon. Simulink provides a set of commands that can display text, one or more plots, or show a transfer function (see "Mask Icon Drawing Commands" in the online Simulink reference). You must use these commands to draw your icon. Simulink executes the drawing commands in the order in which they appear in this field. Drawing commands have access to all variables in the mask workspace.

This example demonstrates how to create an improved icon for the  $mx + b$  sample masked subsystem discussed earlier in this section:

```

pos = get_param(gcb, 'Position');
width = pos(3) - pos(1); height = pos(4) - pos(2);
x = [0, width];
if (m >= 0), y = [0, (m*width)]; end
if (m < 0), y = [height, (height + (m*width))]; end

```

These initialization commands define the data that enables the drawing command to produce an accurate icon regardless of the shape of the block. The drawing command that generates this icon is `plot(x,y)`.

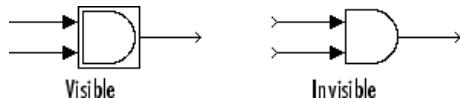
## Examples of drawing commands

This panel illustrates the usage of the various icon drawing commands supported by Simulink. To determine the syntax of a command, select the command from the **Command** list. Simulink displays an example of the selected command at the bottom of the panel and the icon produced by the command to the right of the list.

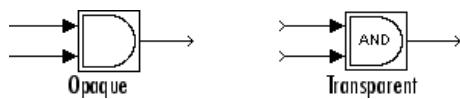
## Icon options

These controls allow you to specify the following attributes of the block icon.

**Frame.** The icon frame is the rectangle that encloses the block. You can choose to show or hide the frame by setting the **Frame** parameter to **Visible** or **Invisible**. The default is to make the icon frame visible. For example, this figure shows visible and invisible icon frames for an AND gate block.



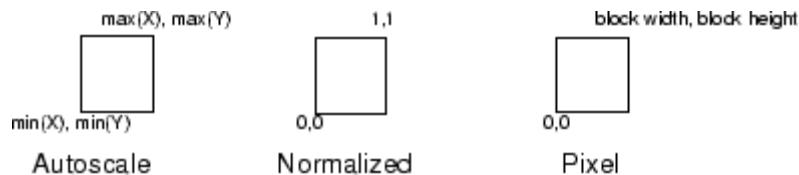
**Transparency.** The icon can be set to **Opaque** or **Transparent**, either hiding or showing what is underneath the icon. **Opaque**, the default, covers information Simulink draws, such as port labels. This figure shows opaque and transparent icons for an AND gate block. Notice the text on the transparent icon.



**Rotation.** When the block is rotated or flipped, you can choose whether to rotate or flip the icon or to have it remain fixed in its original orientation. The default is not to rotate the icon. The icon rotation is consistent with block port rotation. This figure shows the results of choosing Fixed and Rotates icon rotation when the AND gate block is rotated.



**Units.** This option controls the coordinate system used by the drawing commands. It applies only to plot and text drawing commands. You can select from among these choices: Autoscale, Normalized, and Pixel.



- Autoscale scales the icon to fit the block frame. When the block is resized, the icon is also resized. For example, this figure shows the icon drawn using these vectors:

```
X = [0 2 3 4 9]; Y = [4 6 3 5 8];
```



The lower-left corner of the block frame is (0,3) and the upper-right corner is (9,8). The range of the *x*-axis is 9 (from 0 to 9), while the range of the *y*-axis is 5 (from 3 to 8).

- Normalized draws the icon within a block frame whose bottom-left corner is (0,0) and whose top-right corner is (1,1). Only X and Y values between

0 and 1 appear. When the block is resized, the icon is also resized. For example, this figure shows the icon drawn using these vectors:

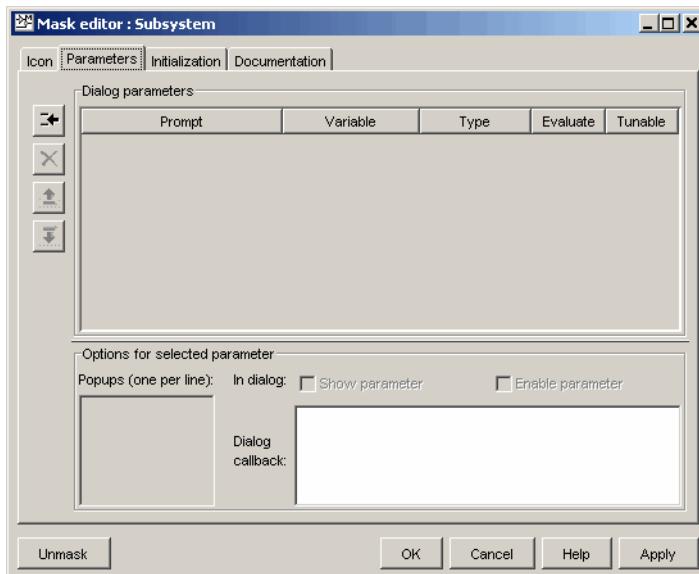
```
X = [.0 .2 .3 .4 .9]; Y = [.4 .6 .3 .5 .8];
```



- Pixel draws the icon with X and Y values expressed in pixels. The icon is not automatically resized when the block is resized. To force the icon to resize with the block, define the drawing commands in terms of the block size.

## Parameters Pane

The **Parameters** pane allows you to create and modify masked subsystem parameters (mask parameters, for short) that determine the behavior of the masked subsystem.



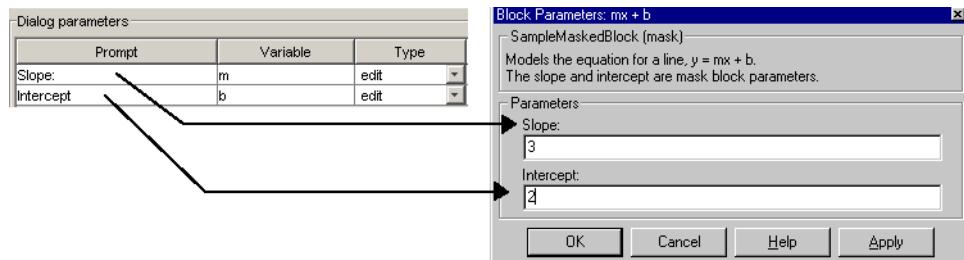
The **Parameters** pane contains the following elements:

- The **Dialog parameters** panel allows you to select and change the major properties of the mask's parameters (see “Dialog Parameters Panel” on page 9-20).
- The **Options for selected parameter** panel allows you to set additional options for the parameter selected in the **Dialog parameters** panel.
- The buttons on the left side of the **Parameters** pane allow you to add, delete, and change the order of appearance of parameters on the mask's parameter dialog box (see “Dialog Parameters Panel” on page 9-20).

### Dialog Parameters Panel

Lists the mask's parameters in tabular form. Each row displays the major attributes of one of the mask's parameters.

**Prompt.** Text that identifies the parameter on a masked subsystem's dialog box.



**Variable.** Name of the variable that stores the parameter's value in the mask's workspace (see “Mask Workspace” on page 9-4). You can use this variable as the value of parameters of blocks inside the masked subsystem, thereby allowing the user to set the parameters via the mask dialog box.

---

**Note** Simulink does not distinguish between uppercase and lowercase letters in mask variable names. For example, Simulink treats gain, GAIN, and Gain as the same name. Also, avoid prefacing mask variable names with L\_ and M\_ to prevent undesirable results. Simulink reserves these specific prefixes for use with its own internal variable names.

---

**Type.** Type of control used to edit the value of this parameter. The control appears on the mask's parameter dialog box following the parameter prompt. The button that follows the type name in the **Parameters** pane pops up a list of the controls supported by Simulink (see “Control Types” on page 9-23). To change the current control type, select another type from the list.

**Evaluate.** If checked, this option causes Simulink to evaluate the expression entered by the user before it is assigned to the variable. Otherwise, Simulink treats the expression itself as a string value and assigns it to the variable. For example, if the user enters the expression gain in an edit field and the **Evaluate** option is checked, Simulink evaluates gain and assigns the result to the variable. Otherwise, Simulink assigns the string 'gain' to the variable. See “Check Box Control” on page 9-24 and “Pop-Up Control” on page 9-24 for information on how this option affects evaluation of the parameters.

If you need both the string entered and the evaluated value, clear the **Evaluate** option. To get the value of a base workspace variable entered as the literal value of the mask parameter, use the MATLAB evalin command in the mask initialization code. For example, suppose the user enters the string 'gain' as the literal value of the mask parameter k where gain is the name of a base workspace variable. To obtain the value of the base workspace variable, use the following command in the mask's initialization code:

```
value = evalin('base', k)
```

**Tunable.** Selecting this option allows a user to change the value of the mask parameter while a simulation is running.

---

**Note** Simulink ignores this setting if the block being masked is a source block, i.e., the block has outputs but no input ports. In such a case, even if this option is selected, you cannot tune the parameter while a simulation is running. See “Changing Source Block Parameters” on page 5-9 for more information.

---

### Options for Selected Parameter Panel

This panel allows you to set additional options for the parameter selected in the **Dialog parameters** table.

**Show parameter.** The selected parameter appears on the masked block's parameter dialog box only if this option is checked (the default).

**Enable parameter.** Clearing this option grays the selected parameter's prompt and disables its edit control. This means that the user cannot set the value of the parameter.

**Popups.** This field is enabled only if the edit control for the selected parameter is a pop-up. Enter the values of the pop-up control in this field, each on a separate line.

**Callback.** Enter MATLAB code that you want Simulink to execute when a user applies a change to the selected parameter, i.e., selects the **Apply** or **OK** button on the mask dialog box.

The callback can create and reference variables only in the block's base workspace. If the callback needs the value of a mask parameter, it can use `get_param` to obtain the value, e.g.,

```
if str2num(get_param(gcb, 'g'))<0
    error('Gain is negative.')
end
```

## Parameter Buttons

The following sections explain the purpose of the buttons that appear on the **Parameters** pane in the order of their appearance from the top of the pane.

**Add Button.** Adds a parameter to the mask's parameter list. The newly created parameter appears in the adjacent **Dialog parameters** table.



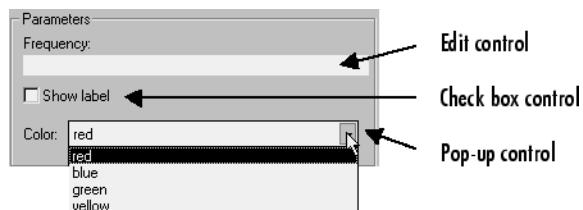
**Delete Button.** Deletes the parameter currently selected in the **Dialog parameters** table.

**Up Button.** Moves the currently selected parameter up one row in the **Dialog parameters** table. Dialog parameters appear in the mask's parameter dialog box (see “Mask Parameter Dialog Box” on page 9-3) in the same order in which they appear in the **Dialog parameters** table. This button (and the next) thus allows you to determine the order in which parameters appear on the dialog box.

**Down Button.** Moves the currently selected parameter down one row in the **Dialog parameters** table and hence down one position on the mask's parameter dialog box.

## Control Types

Simulink enables you to choose how parameter values are entered or selected. You can create three styles of controls: edit fields, check boxes, and pop-up controls. For example, this figure shows the parameter area of a mask dialog box that uses all three styles of controls (with the pop-up control open).



## Edit Control

An *edit field* enables the user to enter a parameter value by typing it into a field. This figure shows how the prompt for the sample edit control was defined.

Dialog parameters					
Prompt	Variable	Type	Evaluate	Tunable	
Slope:	m	edit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Intercept	b	edit	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

The value of the variable associated with the parameter is determined by the **Evaluate** option.

<b>Evaluate</b>	<b>Value</b>
On	The result of evaluating the expression entered in the field
Off	The actual string entered in the field

### Check Box Control

A *check box* enables the user to choose between two alternatives by selecting or deselecting a check box. This figure shows how the sample check box control is defined.

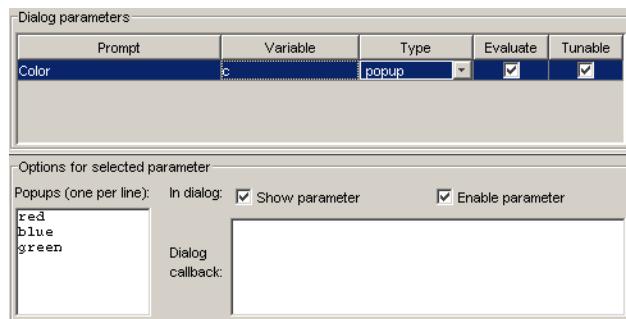


The value of the variable associated with the parameter depends on whether the **Evaluate** option is selected.

<b>Control State</b>	<b>Evaluated Value</b>	<b>Literal Value</b>
Selected	1	'on'
Unselected	0	'off'

### Pop-Up Control

A *pop-up* enables the user to choose a parameter value from a list of possible values. Specify the values in the **Popups** field on the **Options for selected parameter** pane (see “Popups” on page 9-22). The following example shows a pop-up parameter.

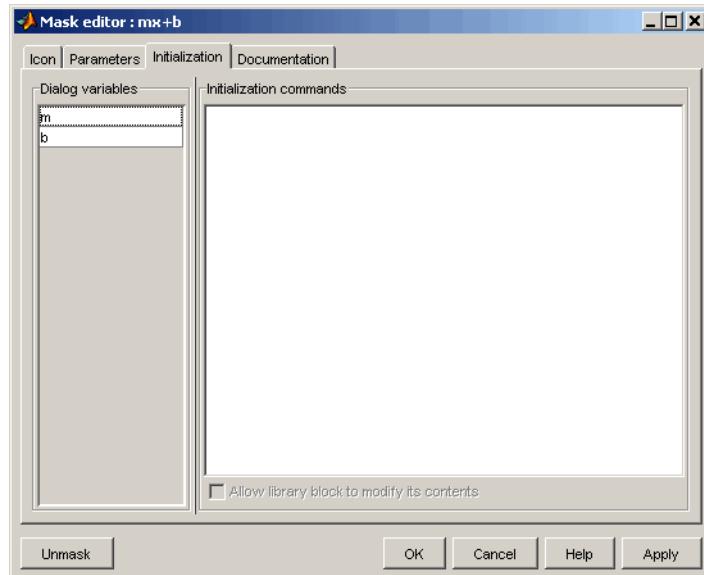


The value of the variable associated with the parameter (Color) depends on the item selected from the pop-up list and whether the **Evaluate** option is checked (on).

<b>Evaluate</b>	<b>Value</b>
On	Index of the value selected from the list, starting with 1. For example, if the third item is selected, the parameter value is 3.
Off	String that is the value selected. If the third item is selected, the parameter value is 'green'.

## Initialization Pane

The **Initialization** pane allows you to enter MATLAB commands that initialize the masked subsystem.



Simulink executes the initialization commands when you

- Load the model
- Start a simulation or update the model's block diagram
- Make changes to the block diagram that affect the appearance of the block, such as rotating the block
- Apply any changes to the block's dialog that affect the block's appearance or behavior, such as changing the value of a mask parameter on which the block's icon drawing code depends

The **Initialization** pane includes the following controls.

### Dialog variables

The **Dialog variables** list displays the names of the variables associated with the subsystem's mask parameters, i.e., the parameters defined in the **Parameters** pane. You can copy the name of a parameter from this list and paste it into the adjacent **Initialization commands** field, using the Simulink keyboard copy and paste commands. You can also use the list to change the

names of mask parameter variables. To change a name, double-click the name in the list. An edit field containing the existing name appears. Edit the existing name and press **Enter** or click outside the edit field to confirm your changes.

### Initialization commands

Enter the initialization commands in this field. You can enter any valid MATLAB expression, consisting of MATLAB functions, operators, and variables defined in the mask workspace. Initialization commands cannot access base workspace variables. Terminate initialization commands with a semicolon to avoid echoing results to the Command Window.

---

**Note** Avoid prefacing variable names in initialization commands with `L_` and `M_` to prevent undesirable results. Simulink reserves these specific prefixes for use with its own internal variable names.

---

### Allow library block to modify its contents

This check box is enabled only if the masked subsystem resides in a library. Checking this block allows the block's initialization code to modify the contents of the masked subsystem, i.e., it lets the code add or delete blocks and set the parameters of those blocks. Otherwise, Simulink generates an error when a masked library block tries to modify its contents in any way. To set this option at the MATLAB prompt, select the self-modifying block and enter the following command.

```
set_param(gcb, 'MaskSelfModifiable', 'on');
```

Then save the block.

### Debugging Initialization Commands

You can debug initialization commands in these ways:

- Specify an initialization command without a terminating semicolon to echo its results to the Command Window.

- Place a keyboard command in the initialization commands to stop execution and give control to the keyboard. For more information, see the help text for the keyboard command.
- Enter either of these commands in the MATLAB Command Window:

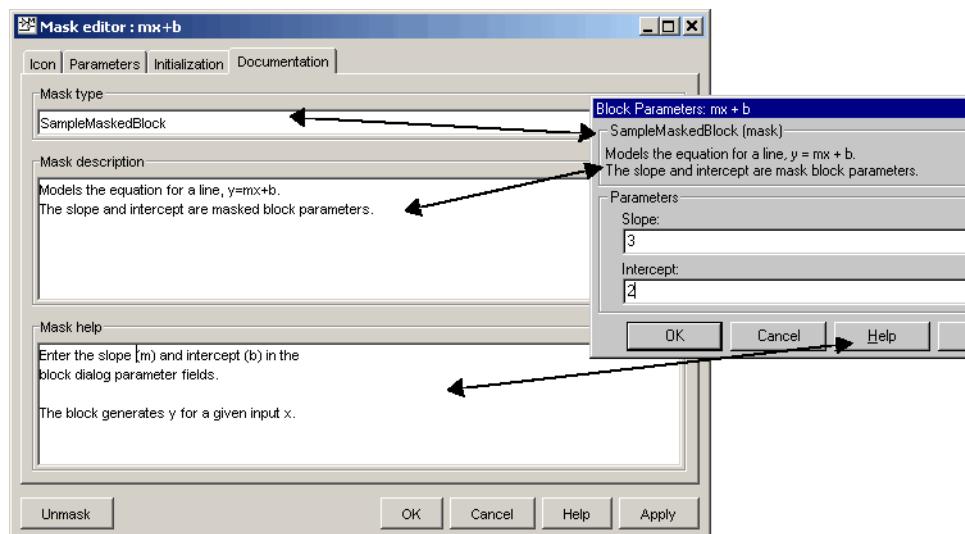
```
dbstop if error
dbstop if warning
```

If an error occurs in the initialization commands, execution stops and you can examine the mask workspace. For more information, see the help text for the dbstop command.

## Documentation Pane

The **Documentation** pane enables you to define or modify the type, description, and help text for a masked block.

This figure shows how fields on the **Documentation** pane correspond to the  $mx + b$  sample mask block's dialog box.



## Mask Type Field

The mask type is a block classification used only for purposes of documentation. It appears in the block's dialog box and on all Mask Editor panes for the block. You can choose any name you want for the mask type. When Simulink creates the block's dialog box, it adds "(mask)" after the mask type to differentiate masked blocks from built-in blocks.

## Mask Description Field

The block description is informative text that appears in the block's dialog box in the frame under the mask type. If you are designing a system for others to use, this is a good place to describe the block's purpose or function.

Simulink automatically wraps long lines of text. You can force line breaks by using the **Enter** or **Return** key.

## Block Help Field

You can provide help text that is displayed when the **Help** button is clicked on the masked block's dialog box. If you create models for others to use, this is a good place to explain how the block works and how to enter its parameters.

You can include user-written documentation for a masked block's help. You can specify any of the following for the masked block help text:

- URL specification (a string starting with `http:`, `www:`, `file:`, `ftp:`, or `mailto:`)
- web command (launches a browser)
- eval command (evaluates a MATLAB string)
- HTML-tagged text to be displayed in a Web browser

Simulink examines the first line of the masked block help text. If Simulink detects a URL specification, for example,

`http://www.mathworks.com`

or

`file:///c:/mydir/helpdoc.html`

Simulink displays the specified file in the browser. If Simulink detects a `web` command, for example,

```
web([docroot '/My Blockset Doc/' get_param(gcb,'MaskType')...  
'.html'])
```

or an `eval` command, for example,

```
eval('!Word My_Spec.doc')
```

Simulink executes the specified command. Otherwise, Simulink displays the contents of the **Block Help** field, which can include HTML tags, in the browser.

Note, if you enter HTML-tagged text, Simulink copies that text into a temporary directory and displays it from that temporary directory. If you want to include an image (for example, with the `img` tag) with that text, you need to place the image file in that temporary directory. (You can use `tempdir` to find the temporary directory for your system.) Alternatively, you can save the HTML-tagged text into an HTML file (such as `hello.html`) in the current directory and display that file directly (for example, `web('hello.html', '-helpbrowser')`). This method enables you to place referenced image files in the same directory as the HTML file.

## Changing Default Values for Mask Parameters in a Library

To change default parameter values in a masked library block:

- 1 Unlock the library.
- 2 Select the block, then select **Edit Mask** from the **Edit** menu to access the block dialog box.

Fill in the desired default values, then save the changes and close the dialog box.

- 3 Save the library.

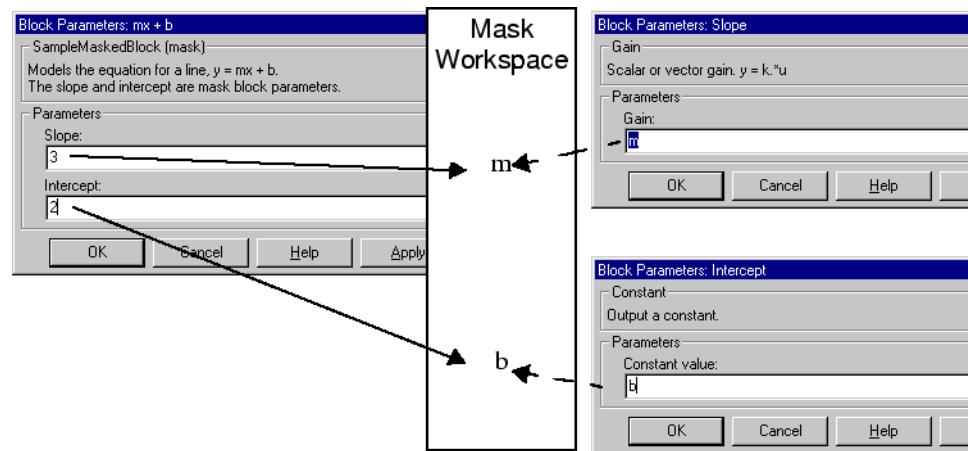
When the block is copied into a model and opened, the default values appear on the block's dialog box.

For more information, see “Working with Block Libraries” on page 5-21.

## Linking Mask Parameters to Block Parameters

The variables associated with mask parameters allow you to link mask parameters with block parameters. This in turn allows a user to use the mask to set the values of parameters of blocks inside the masked subsystem.

To link the parameters, open the block's parameter dialog box and enter an expression in the block parameter's value field that uses the mask parameter. The  $mx + b$  masked subsystem, described earlier in this chapter, uses this approach to link the Slope and Intercept mask parameters to corresponding parameters of a Gain and Constant block, respectively, that reside in the subsystem.



You can use a masked block's initialization code to link mask parameters indirectly to block parameters. In this approach, the initialization code creates variables in the mask workspace whose values are functions of the mask parameters and that appear in expressions that set the values of parameters of blocks concealed by the mask.

# Simulink Debugger

---

The following sections tell you how to use the Simulink debugger to pinpoint bugs in a model:

Introduction (p. 10-2)	Overview of the debugger.
Using the Debugger's Graphical User Interface (p. 10-3)	How to use the debugger's graphical user interface.
Using the Debugger's Command-Line Interface (p. 10-10)	How to debug from the MATLAB command line.
Getting Online Help (p. 10-12)	How to get help on debugger commands.
Starting the Debugger (p. 10-13)	How to start a simulation from the debugger.
Starting a Simulation (p. 10-14)	How to start a simulation in debug mode.
Running a Simulation Step by Step (p. 10-15)	How to run a simulation step by step.
Setting Breakpoints (p. 10-22)	How to set breakpoints at blocks and time steps.
Displaying Information About the Simulation (p. 10-28)	How to display information about the current simulation.
Displaying Information About the Model (p. 10-32)	How to display information about the model being debugged.

## Introduction

The Simulink debugger allows you to run a simulation method by method, stopping the simulation after each method, to examine the results of executing that method. This allows you to pinpoint problems in your model to specific blocks, parameters, or interconnections.

---

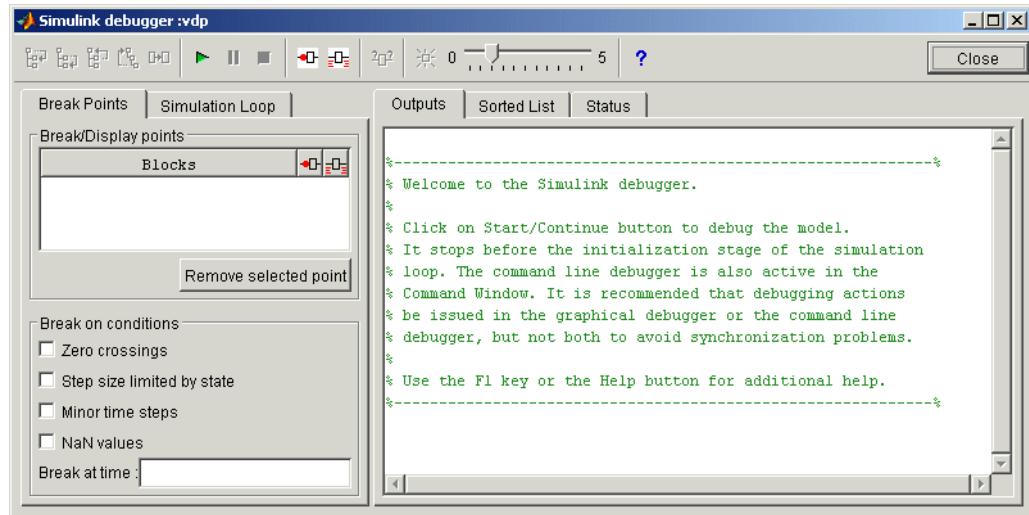
**Note** Methods are functions that Simulink uses to solve a model at each time step during the simulation. Blocks are made up of multiple methods. “Block execution” in this documentation is shorthand notation for “block methods execution.” Block diagram execution is a multi-step operation that requires execution of the different block methods in all the blocks in a diagram at various points during the process of solving a model at each time step during simulation, as specified by the simulation loop.

---

The Simulink debugger has both a graphical and a command-line user interface. The graphical interface allows you to access the debugger’s most commonly used features. The command-line interface gives you access to all the debugger’s capabilities. If both interfaces enable you to perform a task, the documentation shows you first how to use the graphical interface, then the command-line interface, to perform the task.

# Using the Debugger's Graphical User Interface

Select **Simulink Debugger** from a model window's **Tools** menu to display the Simulink debugger's graphical interface.

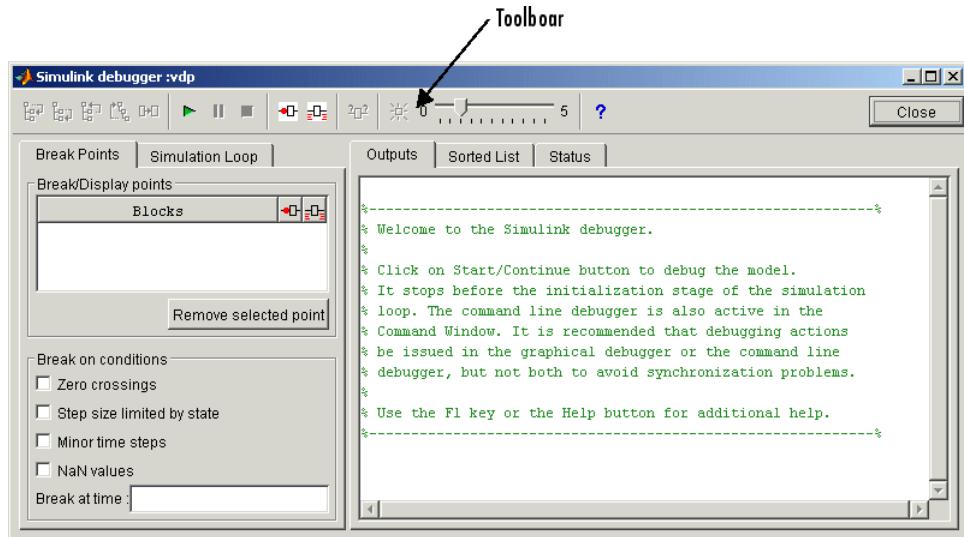


The following topics describe the major components of the debugger's graphical user interface:

- “Toolbar” on page 10-4
- “Breakpoints Pane” on page 10-5
- “Simulation Loop Pane” on page 10-6
- “Outputs Pane” on page 10-7
- “Sorted List Pane” on page 10-8
- “Status Pane” on page 10-9

## Toolbar

The debugger toolbar appears at the top of the debugger window.



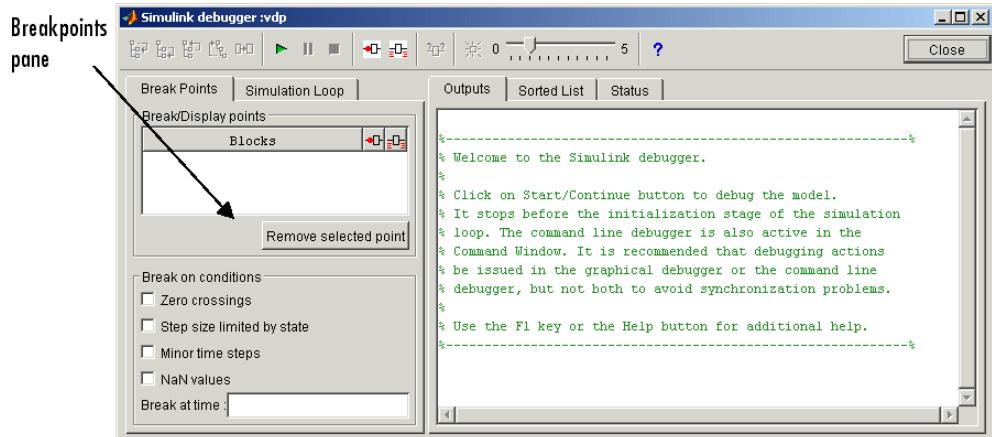
From left to right, the toolbar contains the following command buttons:

Button	Purpose
	Step into next method (see “Stepping Commands” on page 10-16 for more information on this and the following stepping commands).
	Step over next method.
	Step out of current method.
	Step to first method at start of next time step.
	Step to next block method.
	Start or continue the simulation.
	Pause the simulation.

Button	Purpose
	Stop the simulation.
	Break before the selected block.
	Display inputs and outputs of the selected block when executed.
	Display current inputs and outputs of selected block.
	Toggle animation mode on or off (see "Animation Mode" on page 10-18). The slider next to this button controls the animation rate.
	Display help for the debugger.
	Close the debugger.

## Breakpoints Pane

To display the **Breakpoints** pane, select the **Break Points** tab on the debugger window.



The **Breakpoints** pane allows you to specify block methods or conditions at which to stop a simulation. See “Setting Breakpoints” on page 10-22 for more information.

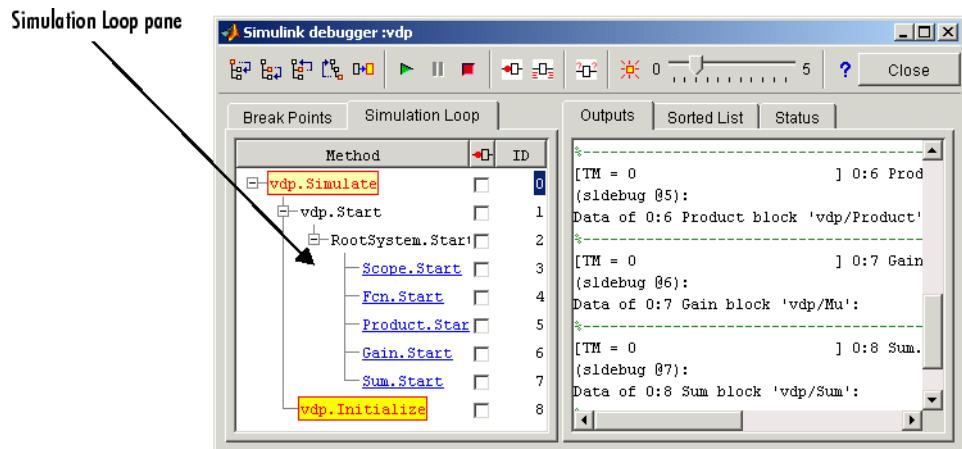
---

**Note** The debugger grays out and disables the **Breakpoints** pane when its animation mode is selected (see “Animation Mode” on page 10-18). This prevents you from setting breakpoints and indicates that animation mode ignores existing breakpoints.

---

## Simulation Loop Pane

To display the **Simulation Loop** pane, select the **Simulation Loop** tab on the debugger window.



The Simulation Loop pane contains three columns:

- Method
- Breakpoints
- ID

## Method Column

The **Method** column lists the methods that have been called thus far in the simulation as a method tree with expandable/collapsible nodes. Each node of the tree represents a method that calls other methods. Expanding a node shows the methods that the block method calls. Block method names are hyperlinks. Clicking a block method name highlights the corresponding block in the model diagram. Block method names are underlined to indicate that they are hyperlinks.

Whenever the simulation stops, the debugger highlights the name of the method where the simulation has stopped as well as the methods that directly or indirectly invoked it. The highlighted method names visually indicate the current state of the simulator's method call stack.

## Breakpoints Column

The breakpoints column consists of check boxes. Selecting a check box sets a breakpoint at the method whose name appears to the left of the check box. See “Setting Breakpoints from the Simulation Loop Pane” on page 10-24 for more information.

---

**Note** The debugger grays out and disables this column when its animation mode is selected (see “Animation Mode” on page 10-18). This prevents you from setting breakpoints and indicates that animation mode ignores existing breakpoints.

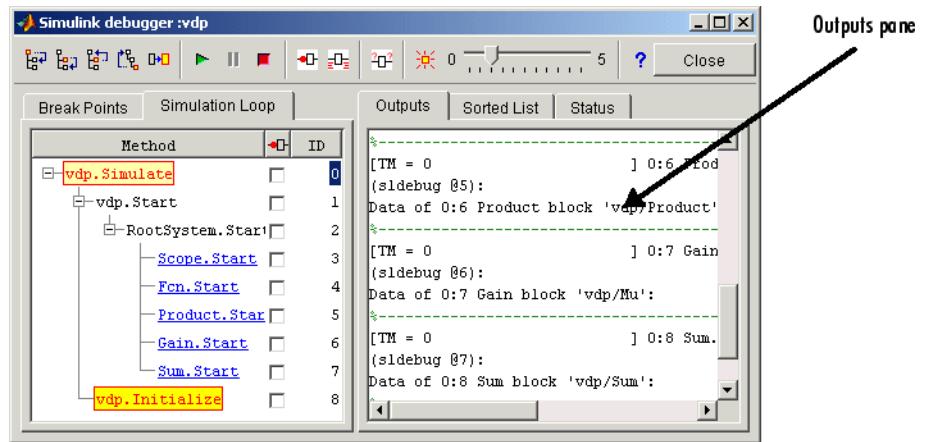
---

## ID Column

The ID column lists the IDs of the methods listed in the **Methods** column. See “Method ID” on page 10-10 for more information.

## Outputs Pane

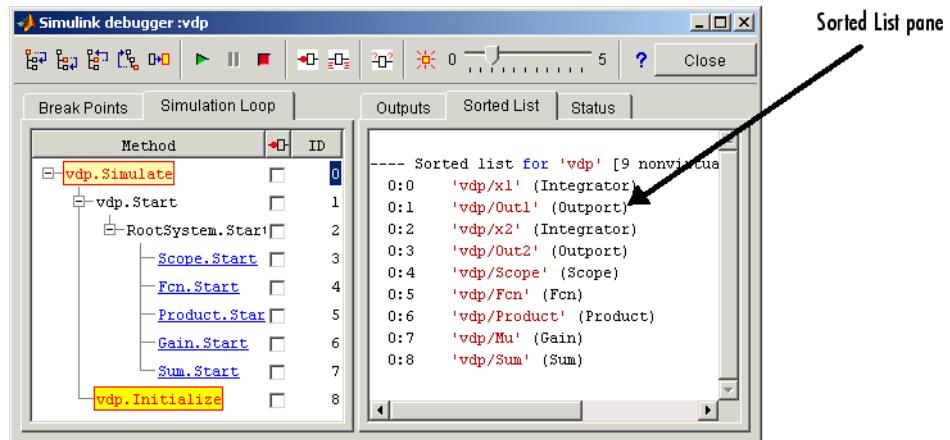
To display the **Outputs** pane, select the **Outputs** tab on the debugger window.



The Outputs pane displays the same debugger output that would appear in the MATLAB Command Window, if the debugger were running in command-line mode. The output includes the debugger command prompt and the inputs, outputs, and states of the block at whose method the simulation is currently paused (see “Block Data Output” on page 10-16). The command prompt displays current simulation time and the name and index of the method in which the debugger is currently stopped (see “Block ID” on page 10-10).

## Sorted List Pane

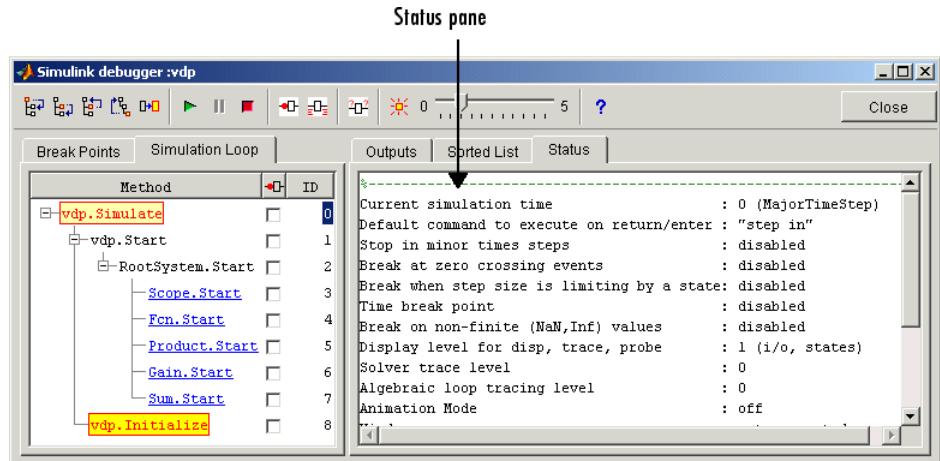
To display the **Sorted List** pane, select the **Sorted List** tab on the debugger window.



The **Sorted List** pane displays the sorted lists for the model being debugged. See “Displaying a Model’s Sorted Lists” on page 10-32 for more information.

## Status Pane

To display the **Status** pane, select the **Status** tab on the debugger window.



The **Status** pane displays the values of various debugger options and other status information.

## Using the Debugger’s Command-Line Interface

In command-line mode, you control the debugger by entering commands at the debugger command line in the MATLAB Command Window. The debugger accepts abbreviations for debugger commands. See “Simulink Debugger Commands—Alphabetical List” in the online Simulink reference for a list of command abbreviations and repeatable commands. You can repeat some commands by entering an empty command (i.e., by pressing the **Enter** key) at the MATLAB command line.

### Method ID

Some Simulink commands and messages use method IDs to refer to methods. A method ID is an integer assigned to a method the first time it is invoked in a simulation. The debugger assigns method indexes sequentially, starting with 0 for the first method invoked in a debugger session.

### Block ID

Some Simulink debugger commands and messages use block IDs to refer to blocks. Simulink assigns block IDs to blocks while generating the model’s sorted lists during the compilation phase of the simulation. A block ID has the form `sid:bid` where `sid` is an integer identifying the system that contains the block (either the root system or a nonvirtual subsystem) and `bid` is the position of the block in the system’s sorted list. For example, the block index `0:1` refers to the first block in the model’s root system. The `slist` command shows the block ID for each block in the model being debugged.

### Accessing the MATLAB Workspace

You can enter any MATLAB expression at the `sldebug` prompt. For example, suppose you are at a breakpoint and you are logging time and output of your model as `tout` and `yout`. Then the following command

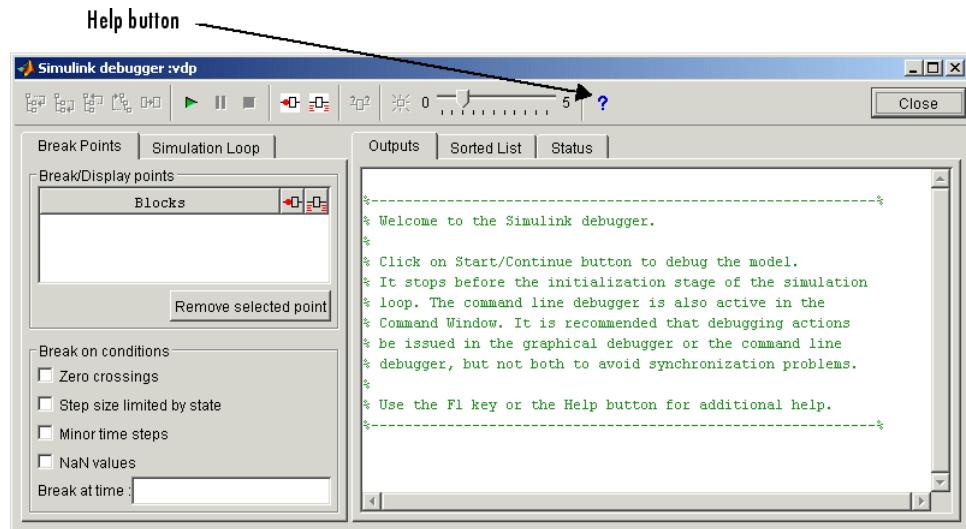
```
(sldebug ...) plot(tout, yout)
```

creates a plot. You cannot display the value of a workspace variable whose name is partially or entirely the same as that of a debugger command by entering it at the debugger command prompt. You can, however, use the

MATLAB eval command to work around this problem. For example, use eval('s') to determine the value of s rather than s(step) the simulation.

## Getting Online Help

You can get online help on using the debugger by clicking the **Help** button on the debugger's toolbar or by pressing the **F1** key when the text cursor is in a debugger panel or text field. Clicking the **Help** button displays help for the debugger in the MATLAB Help browser.



Pressing the **F1** key displays help for the debugger panel or text field that currently has the keyboard input focus. In command-line mode, you can get a brief description of the debugger commands by typing `help` at the debug prompt.

## Starting the Debugger

You can start the debugger either from a model window or from the MATLAB command line. To start the debugger from a model window, select **Simulink Debugger** from the model window's **Tools** menu. The debugger's graphical user interface appears (see “Using the Debugger’s Graphical User Interface” on page 10-3).

To start the debugger from the MATLAB Command Window, enter either a `sim` command or the `sldebug` command. For example, either the command

```
sim('vdp',[0,10],simset('debug','on'))
```

or the command

```
sldebug 'vdp'
```

loads the Simulink demo model `vdp` into memory, starts the simulation, and stops the simulation at the first block in the model's execution list.

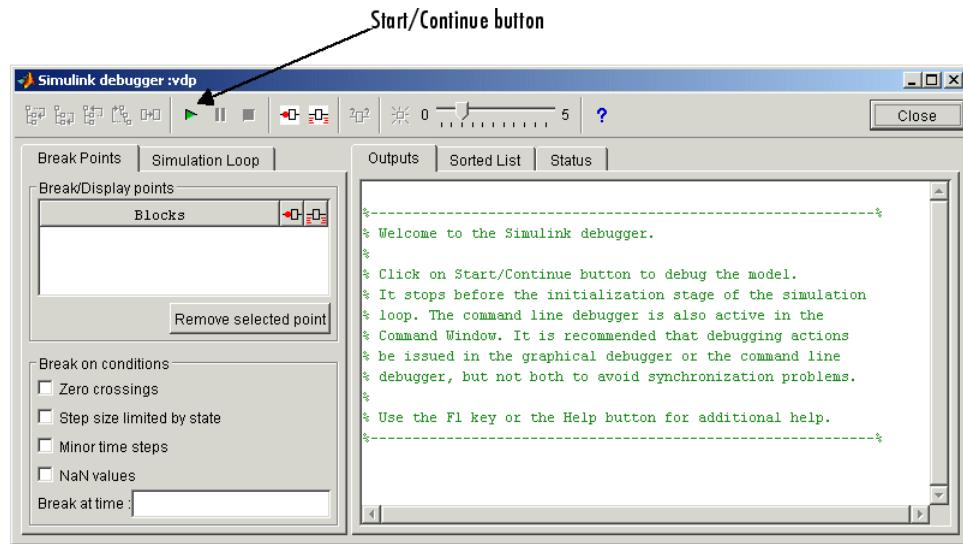
---

**Note** When running the debugger in graphical user interface (GUI) mode, you must explicitly start the simulation. See “Starting a Simulation” on page 10-14 for more information.

---

## Starting a Simulation

To start the simulation, click the **Start/Continue** button on the debugger's toolbar.



The simulation starts and stops at the first simulation method to be executed. It displays the name of the method in its **Simulation Loop** pane and in the current method annotation on the Simulink block diagram. At this point, you can set breakpoints, run the simulation step by step, continue the simulation to the next breakpoint or end, examine data, or perform other debugging tasks. The following sections explain how to use the debugger's graphical controls to perform these debugging tasks.

---

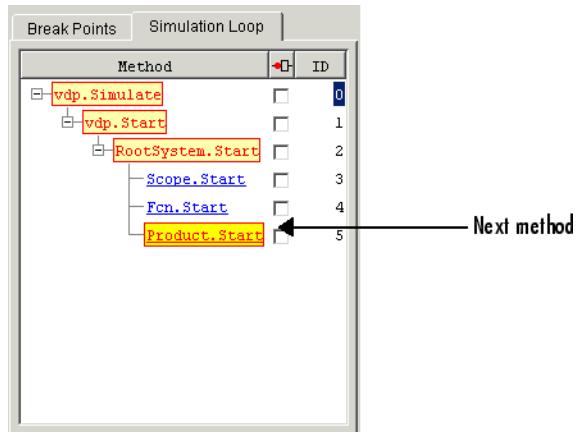
**Note** When you start the debugger in GUI mode, the debugger's command-line interface is also active in the MATLAB Command Window. However, you should avoid using the command-line interface, to prevent synchronization errors between the graphical and command-line interfaces.

---

## Running a Simulation Step by Step

The Simulink debugger provides various commands that let you advance a simulation from the method where it is currently suspended (the next method) by various increments (see “Stepping Commands” on page 10-16). For example, you can advance the simulation into or over the next method, or out of the current method, or to the top of the simulation loop. After each advance, the debugger displays information that enables you to determine the point to which the simulation has advanced and the results of advancing the simulation to that point.

For example, in GUI mode, after each step command, the debugger highlights the current method call stack in the **Simulation Loop** pane. The call stack comprises the next method and the methods that invoked the next method either directly or indirectly. The debugger highlights the call stack by highlighting the names of the methods that make up the call stack in the **Simulation Loop** pane.



In command-line mode, you can use the `where` command to display the method call stack. If the next method is a block method, the debugger points the debug pointer at the block corresponding to the method (see “Debug Pointer” on page 10-20 for more information). If the block of the next method to be executed resides in a subsystem, the debugger opens the subsystem and points to the block in the subsystem’s block diagram.

## Block Data Output

After executing a block method, the debugger prints any or all of the following block data in the debugger Output panel (in GUI mode) or in the MATLAB Command Window (in command-line mode):

- $U_n = v$   
where  $v$  is the current value of the block's  $n$ th input.
- $Y_n = v$   
where  $v$  is the current value of the block's  $n$ th output.
- $CSTATE = v$   
where  $v$  is the value of the block's continuous state vector.
- $DSTATE = v$   
where  $v$  is the value of the blocks discrete state vector.

The debugger also displays the current time, the ID and name of the next method to be executed, and the name of the block to which the method applies in the MATLAB Command Window. The following example illustrates typical debugger outputs after a step command.

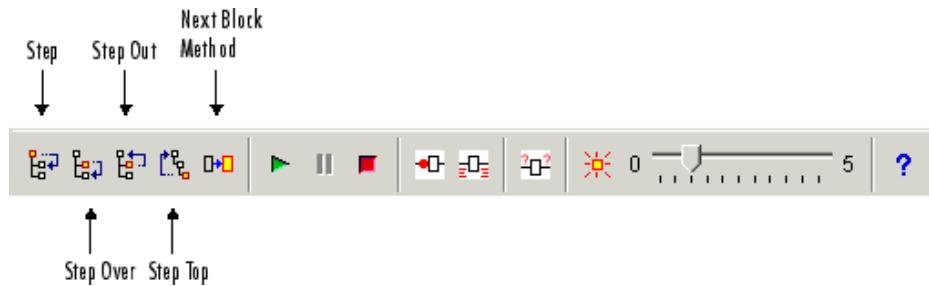
Current time	Next method
<pre>%-----% [Tm = 2.009509145207664e-005 ] 0:2 Integrator.Outputs 'vdp/x2' (sldebug @44): Data of 0:2 Integrator block 'vdp/x2': U1      = [-2] Y1      = [-4.0190182904153282e-005] CSTATE = [-4.0190182904153282e-005] %-----% [Tm = 2.009509145207664e-005 ] 0:3 Outport.Outputs 'vdp/Out2'</pre>	

## Stepping Commands

Command-line mode provides the following commands for advancing a simulation incrementally:

Command	Advances the simulation...
<code>step [in into]</code>	Into the next method, stopping at the first method in the next method or, if the next method does not contain any methods, at the end of the next method
<code>step over</code>	To the method that follows the next method, executing all methods invoked directly or indirectly by the next method
<code>step out</code>	To the end of the current method, executing any remaining methods invoked by the current method
<code>step top</code>	To the first method of the next time step (i.e., the top of the simulation loop)
<code>step blockmth</code>	To the next block method to be executed, executing all intervening model- and system-level methods
<code>next</code>	Same as <code>step over</code>

Buttons in the debugger toolbar allow you to access these commands in GUI mode.



Clicking a button has the same effect as entering the corresponding command at the debugger command line.

## Continuing a Simulation

In GUI mode, the **Stop** button turns red when the debugger suspends the simulation for any reason. To continue the simulation, click the **Start/Continue** button. In command-line mode, enter `continue` to continue the simulation. By default, the debugger runs the simulation to the next

breakpoint (see “Setting Breakpoints” on page 10-22) or to the end of the simulation, whichever comes first.

### Animation Mode

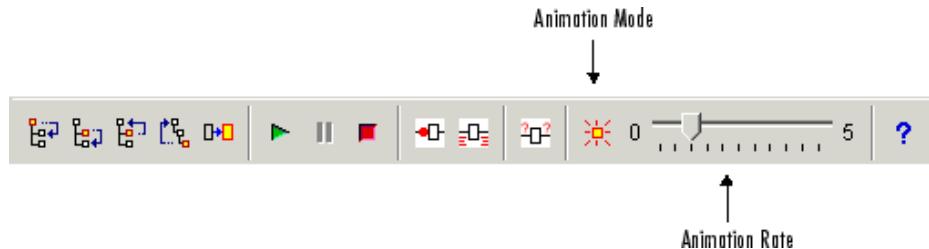
In *animation mode*, the **Start/Continue** button or the **continue** command advances the simulation method by method, pausing after each method, to the first method of the next major time step. While running the simulation in animation mode, the debugger uses its debug pointer (see “Debug Pointer” on page 10-20) to indicate on the block diagram which block method is being executed at each step. The moving pointer providing a visual indication of the progress of the simulation.

---

**Note** When animation mode is enabled, the debugger does not allow you to set breakpoints and ignores any breakpoints that you set when animating the simulation.

---

To enable animation when running the debugger in GUI mode, click the **Animation Mode** toggle button on the debugger’s toolbar.



The slider on the debugger toolbar allows you to increase or decrease the delay between method invocations and hence to slow down or speed up the animation rate. To disable animation mode when running the debugger in GUI mode, toggle the **Animation Mode** button on the toolbar.

To enable animation when running the debugger in command-line mode, enter the `animate` command at the MATLAB command line. The `animate` command’s optional delay parameter allows you to specify the length of

the pause between method invocations (1 second by default) and thereby accelerate or slow down the animation. For example, the command

```
animate 0.5
```

causes the animation to run at twice its default rate. To disable animation mode when running the debugger in command-line mode, enter

```
animate stop
```

at the MATLAB command line.

## Running a Simulation Nonstop

The `run` command lets you run a simulation to the end of the simulation, skipping any intervening breakpoints. At the end of the simulation, the debugger returns you to the MATLAB command line. To continue debugging a model, you must restart the debugger.

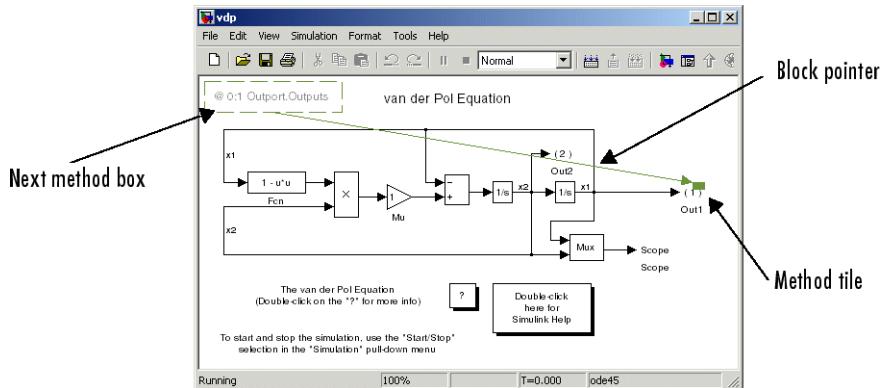
---

**Note** The GUI mode does not provide a graphical version of the `run` command. To run the simulation to the end, you must first clear all breakpoints and then click the **Start/Continue** button.

---

## Debug Pointer

Whenever the debugger stops the simulation at a method, it displays a debug pointer on the block diagram of the model being debugged.



The debug pointer is an annotation that indicates the next method to be executed when simulation resumes. It consists of the following elements:

- Next method box
- Block pointer
- Method tile

### Next Method Box

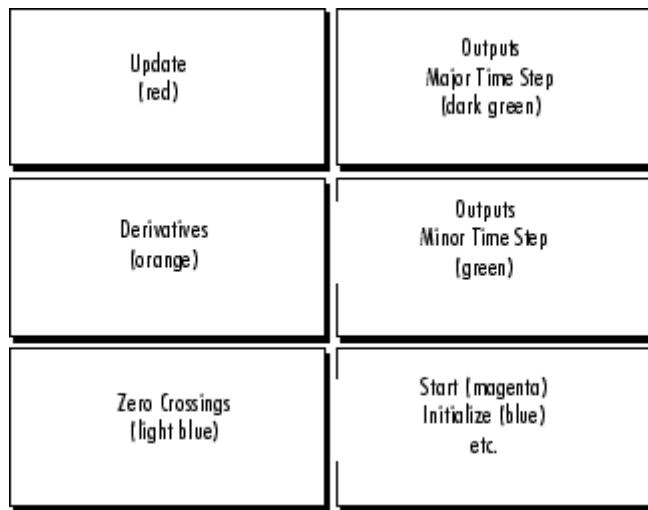
The next method box appears in the upper-left corner of the block diagram. It specifies the name and ID of the next method to be executed.

### Block Pointer

The block pointer appears when the next method is a block method. It indicates the block on which the next method operates.

## Method Tile

The method tile is a rectangular patch of color that appears when the next method is a block method. The tile overlays a portion of the block on which the next method executes. The color and position of the tile on the block indicate the type of the next block method as follows.



In animation mode, the tiles persist for the length of the current major time step and a number appears in each tile. The number specifies the number of times that the corresponding method has been invoked for the block thus far in the time step.

## Setting Breakpoints

The Simulink debugger allows you to define stopping points in a simulation called breakpoints. You can then run a simulation from breakpoint to breakpoint, using the debugger's continue command. The debugger lets you define two types of breakpoints: unconditional and conditional. An unconditional breakpoint occurs whenever a simulation reaches a method that you specified previously. A conditional breakpoint occurs when a condition that you specified in advance arises in the simulation.

Breakpoints are useful when you know that a problem occurs at a certain point in your program or when a certain condition occurs. By defining an appropriate breakpoint and running the simulation via the continue command, you can skip immediately to the point in the simulation where the problem occurs.

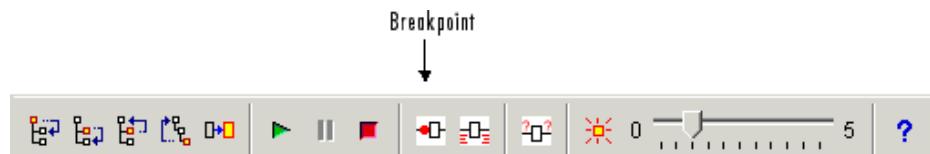
### Setting Unconditional Breakpoints

You can set unconditional breakpoints from the

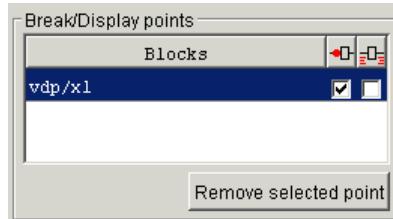
- Debugger toolbar
- **Simulation Loop** pane
- MATLAB Command Window (command-line mode only)

### Setting Breakpoints from the Debugger Toolbar

To set a breakpoint on a block's methods, select the block and then click the **Breakpoint** button on the debugger toolbar.



The debugger displays the name of the selected block in the **Break/Display points** panel of its **Breakpoints** pane.



---

**Note** Clicking the **Breakpoint** button on the toolbar sets breakpoints on the invocations of a block's methods in major time steps.

---

You can temporarily disable the breakpoints on a block by deselecting the check box in the breakpoints column of the panel. To clear the breakpoints on a block and remove its entry from the panel, select the entry and then click the **Remove selected point** button on the panel.

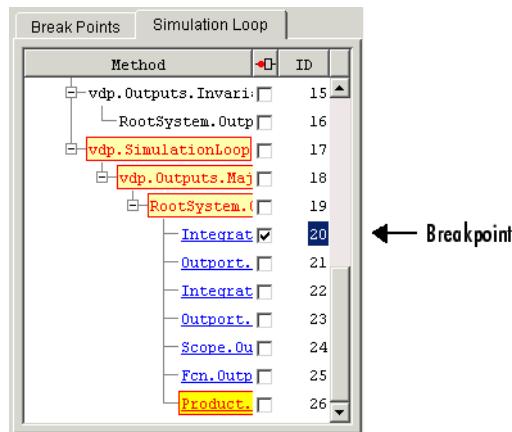
---

**Note** You cannot set a breakpoint on a virtual block. A virtual block is a block whose function is purely graphical: it indicates a grouping or relationship among a model's computational blocks. The debugger warns you if you attempt to set a breakpoint on a virtual block. You can obtain a listing of a model's nonvirtual blocks, using the `slist` command (see “Displaying a Model’s Nonvirtual Blocks” on page 10-34).

---

## Setting Breakpoints from the Simulation Loop Pane

To set a breakpoint at a particular invocation of a method displayed in the Simulation Loop pane, select the check box next to the method's name in the breakpoint column of the pane.



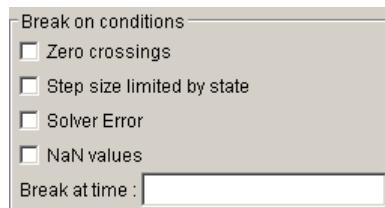
To clear the breakpoint, deselect the check box.

## Setting Breakpoints from the MATLAB Command Window

In command-line mode, use the `break` and `bafter` commands to set breakpoints before or after a specified method, respectively. Use the `clear` command to clear breakpoints.

## Setting Conditional Breakpoints

You can use either the **Break on conditions** panel of the debugger's **Breakpoints** pane



or the following commands (in command-line mode) to set conditional breakpoints.

Command	Causes Simulation to Stop
tbreak [t]	At a simulation time step
ebreak	At a recoverable error in the model
nanbreak	At the occurrence of an underflow or overflow (NaN) or infinite (Inf) value
xbreak	When the simulation reaches the state that determines the simulation step size
zcbreak	When a zero crossing occurs between simulation time steps

### Setting Breakpoints at Time Steps

To set a breakpoint at a time step, enter a time in the debugger's **Break at time** field (GUI mode) or enter the time using the **tbreak** command. This causes the debugger to stop the simulation at the **Outputs.Major** method of the model at the first time step that follows the specified time. For example, starting **vdp** in debug mode and entering the commands

```
tbreak 2
continue
```

causes the debugger to halt the simulation at the **vdp.Outputs.Major** method of time step 2.078 as indicated by the output of the **continue** command.

```
%-----%
[ TM = 2.078784598291364      ] vdp.Outputs.Major
(sldebug @18):
```

### Breaking on Nonfinite Values

Selecting the debugger's **Nan values** option or entering the **nanbreak** command causes the simulation to stop when a computed value is infinite or outside the range of values that can be represented by the machine running

the simulation. This option is useful for pinpointing computational errors in a Simulink model.

### **Breaking on Step-Size Limiting Steps**

Selecting the **Step size limited by state** option or entering the `xbreak` command causes the debugger to stop the simulation when the model uses a variable-step solver and the solver encounters a state that limits the size of the steps that it can take. This command is useful in debugging models that appear to require an excessive number of simulation time steps to solve.

### **Breaking at Zero Crossings**

Selecting the **Zero crossings** option or entering the `zcbreak` command causes the simulation to halt when Simulink detects a nonsampled zero crossing in a model that includes blocks where zero crossings can arise. After halting, Simulink displays the location in the model, the time, and the type (rising or falling) of the zero crossing. For example, setting a zero-crossing break at the start of execution of the `zeroxing` demo model,

```
sldebug zeroxing
%-----
%
[ TM = 0 ] zeroxing.Simulate
(sldebug @0): zcbreak
Break at zero crossing events : enabled
```

and continuing the simulation

```
(sldebug @0): continue
```

results in a rising zero-crossing break at

```
[ Tz = 0.2 ] [Hz = 0 ]
Detected 2 Zero Crossing Events 0:5:1R, 0:5:2R
%-----
%
[ Tm = 0.4 ] zeroxing.ZeroCrossingDetectionLoop
(sldebug @45):
```

If a model does not include blocks capable of producing nonsampled zero crossings, the command prints a message advising you of this fact.

### **Breaking on Solver Errors**

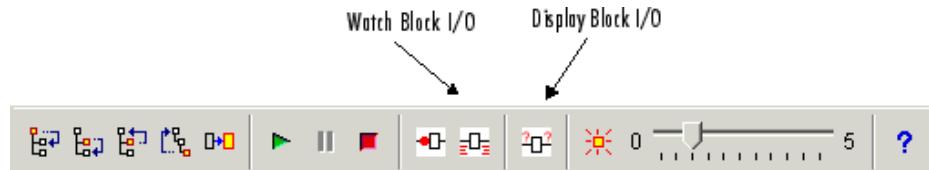
Selecting the debugger's **Solver Errors** option or entering the `ebreak` command causes the simulation to stop if the solver detects a recoverable error in the model. If you do not set or disable this breakpoint, the solver recovers from the error and proceeds with the simulation without notifying you.

## Displaying Information About the Simulation

The Simulink debugger provides a set of commands that allow you to display block states, block inputs and outputs, and other information while running a model.

### Displaying Block I/O

The debugger allows you to display block I/O by clicking the appropriate buttons on the debugger toolbar



or by entering the appropriate debugger command.

Command	Displays a Block's I/O
probe	Immediately
disp	At every breakpoint
trace	Whenever the block executes

### Displaying I/O of Selected Block

To display the I/O of a block, select the block and click in GUI mode or enter the `probe` command in command-line mode.

Command	Description
probe	Enter or exit probe mode. In probe mode, the debugger displays the current inputs and outputs of any block that you select in the model's block diagram. Typing any command causes the debugger to exit probe mode.

Command	Description
<code>probe gcb</code>	Display I/O of selected block.
<code>probe s:b</code>	Print the I/O of the block specified by system number s and block number b.

The debugger prints the current inputs, outputs, and states of the selected block in the debugger **Outputs** pane (GUI mode) or the MATLAB Command Window.

The `probe` command is useful when you need to examine the I/O of a block whose I/O is not otherwise displayed. For example, suppose you are using the `step` command to run a model method by method. Each time you step the simulation, the debugger displays the inputs and outputs of the current block. The `probe` command lets you examine the I/O of other blocks as well.

### Displaying Block I/O Automatically at Breakpoints

The `disp` command causes the debugger to display a specified block's inputs and outputs whenever it halts the simulation. You can specify a block either by entering its block index or by selecting it in the block diagram and entering `gcb` as the `disp` command argument. You can remove any block from the debugger's list of display points, using the `undisp` command. For example, to remove block `0:0`, either select the block in the model diagram and enter `undisp gcb` or simply enter `undisp 0:0`.

---

**Note** Automatic display of block I/O at breakpoints is not available in the debugger's GUI mode.

---

The `disp` command is useful when you need to monitor the I/O of a specific block or set of blocks as you step through a simulation. Using the `disp` command, you can specify the blocks you want to monitor and the debugger will then redisplay the I/O of those blocks on every step. Note that the debugger always displays the I/O of the current block when you step through a model block by block, using the `step` command. You do not need to use the `disp` command if you are interested in watching only the I/O of the current block.

## Watching Block I/O

To watch a block, select the block and click  in the debugger toolbar or enter the `trace` command. In GUI mode, if a breakpoint exists on the block, you can set a watch on it as well by selecting the check box for the block in the watch column  of the **Break/Display points** pane. In command-line mode, you can also specify the block by specifying its block index in the `trace` command. You can remove a block from the debugger's list of trace points using the `untrace` command.

The debugger displays a watched block's I/O whenever the block executes. Watching a block allows you obtain a complete record of the block's I/O without having to stop the simulation.

## Displaying Algebraic Loop Information

The `atrace` command causes the debugger to display information about a model's algebraic loops (see “Algebraic Loops” on page 2-24) each time they are solved. The command takes a single argument that specifies the amount of information to display.

Command	Displays for Each Algebraic Loop
<code>atrace 0</code>	No information
<code>atrace 1</code>	The loop variable solution, the number of iterations required to solve the loop, and the estimated solution error
<code>atrace 2</code>	Same as level 1
<code>atrace 3</code>	Level 2 plus the Jacobian matrix used to solve the loop
<code>atrace 4</code>	Level 3 plus intermediate solutions of the loop variable

## Displaying System States

The states debug command lists the current values of the system's states in the MATLAB Command Window. For example, the following sequence of commands shows the states of the Simulink bouncing ball demo (bounce) after its first and second time steps.

```
sldebug bounce
[Tm=0] **Start** of system 'bounce' outputs
(sldebug @0:0 'bounce/Position'): states
Continuous state vector (value,index,name):
    10          0 (0:0 'bounce/Position')
    15          1 (0:5 'bounce/Velocity')
(sldebug @0:0 'bounce/Position'): next
[Tm=0.01] **Start** of system 'bounce' outputs
(sldebug @0:0 'bounce/Position'): states
Continuous state vector (value,index,name):
    10.1495095  0 (0:0 'bounce/Position')
    14.9019     1 (0:5 'bounce/Velocity')
```

## Displaying Solver Information

The strace command allows you to pinpoint problems in solving a model's differential equations that can slow down simulation performance. Executing this command causes the debugger to display solver-related information at the MATLAB command line when you run or step through a simulation. The information includes the sizes of the steps taken by the solver, the estimated integration error resulting from the step size, whether a step size succeeded (i.e., met the accuracy requirements that the model specifies), the times at which solver resets occur, etc. If you are concerned about the time required to simulate your model, this information can help you to decide whether the solver you have chosen is the culprit and hence whether choosing another solver might shorten the time required to solve the model.

## Displaying Information About the Model

In addition to providing information about a simulation, the debugger can provide you with information about the model that underlies the simulation.

### Displaying a Model's Sorted Lists

In GUI mode, the debugger's **Sorted List** pane displays lists of blocks for a model's root system and each nonvirtual subsystem. Each list lists the blocks that the subsystems contains sorted according to their computational dependencies, alphabetical order, and other block sorting rules. In command-line mode, you can use the `slist` command to display a model's sorted lists.

```
---- Sorted list for 'vdp' [12 blocks, 9 nonvirtual blocks,
directFeed=0]
 0:0    'vdp/Integrator1' (Integrator)
 0:1    'vdp/Out1' (Outport)
 0:2    'vdp/Integrator2' (Integrator)
 0:3    'vdp/Out2' (Outport)
 0:4    'vdp/Fcn' (Fcn)
 0:5    'vdp/Product' (Product)
 0:6    'vdp/Mu' (Gain)
 0:7    'vdp/Scope' (Scope)
 0:8    'vdp/Sum' (Sum)
```

These displays include the block index for each command. You can thus use them to determine the block IDs of the model's blocks. Some debugger commands accept block IDs as arguments.

### Identifying Blocks in Algebraic Loops

If a block belongs to an algebraic list, the `slist` command displays an algebraic loop identifier in the entry for the block in the sorted list. The identifier has the form

```
algId=s#n
```

where `s` is the index of the subsystem containing the algebraic loop and `n` is the index of the algebraic loop in the subsystem. For example, the following

entry for an Integrator block indicates that it participates in the first algebraic loop at the root level of the model.

```
0:1 'test/ss/I1' (Integrator, tid=0) [algId=0#1, discontinuity]
```

You can use the debugger's `ashow` command to highlight the blocks and lines that make up an algebraic loop. See “[Displaying Algebraic Loops](#)” on page 10-35 for more information.

## Displaying a Block

To determine the block in a model’s diagram that corresponds to a particular index, enter `bshow s:b` at the command prompt, where `s:b` is the block index. The `bshow` command opens the system containing the block (if necessary) and selects the block in the system’s window.

## Displaying a Model’s Nonvirtual Systems

The `systems` command displays a list of the nonvirtual systems in the model being debugged. For example, the Simulink clutch demo (`clutch`) contains the following systems:

```
sldebug clutch
[Tm=0] **Start** of system 'clutch' outputs
(sldebug @0:0 'clutch/Clutch Pedal'): systems
 0  'clutch'
 1  'clutch/Locked'
 2  'clutch/Unlocked'
```

---

**Note** The `systems` command does not list subsystems that are purely graphical in nature, that is, subsystems that the model diagram represents as Subsystem blocks but that Simulink solves as part of a parent system. In Simulink models, the root system and triggered or enabled subsystems are true systems. All other subsystems are virtual (that is, graphical) and hence do not appear in the listing produced by the `systems` command.

---

## Displaying a Model's Nonvirtual Blocks

The `slist` command displays a list of the nonvirtual blocks in a model. The listing groups the blocks by system. For example, the following sequence of commands produces a list of the nonvirtual blocks in the Van der Pol (`vdp`) demo model.

```
sldebug vdp
[Tm=0] **Start** of system 'vdp' outputs
(sldebug @0:0 'vdp/Integrator1'): slist
---- Sorted list for 'vdp' [12 blocks, 9 nonvirtual blocks,
directFeed=0]
  0:0  'vdp/Integrator1' (Integrator)
  0:1  'vdp/Out1' (Outport)
  0:2  'vdp/Integrator2' (Integrator)
  0:3  'vdp/Out2' (Outport)
  0:4  'vdp/Fcn' (Fcn)
  0:5  'vdp/Product' (Product)
  0:6  'vdp/Mu' (Gain)
  0:7  'vdp/Scope' (Scope)
  0:8  'vdp/Sum' (Sum)
```

---

**Note** The `slist` command does not list blocks that are purely graphical in nature, that is, blocks that indicate relationships or groupings among computational blocks.

---

## Displaying Blocks with Potential Zero Crossings

The `zclist` command displays a list of blocks in which nonsampled zero crossings can occur during a simulation. For example, `zclist` displays the following list for the clutch sample model:

```
(sldebug @0:0 'clutch/Clutch Pedal'): zclist
  2:3  'clutch/Unlocked/Sign' (Signum)
  0:4  'clutch/Lockup Detection/Velocities Match' (HitCross)
  0:10  'clutch/Lockup Detection/Required Friction
        for Lockup/Abs' (Abs)
  0:11  'clutch/Lockup Detection/Required Friction for
        Lockup/ Relational Operator' (RelationalOperator)
```

```

0:18  'clutch/Break Apart Detection/Abs' (Abs)
0:20  'clutch/Break Apart Detection/Relational Operator'
      (RelationalOperator)
0:24  'clutch/Unlocked' (SubSystem)
0:27  'clutch/Locked' (SubSystem)

```

## Displaying Algebraic Loops

The `ashow` command highlights a specified algebraic loop or the algebraic loop that contains a specified block. To highlight a specified algebraic loop, enter `ashow s#n`, where `s` is the index of the system (see “Identifying Blocks in Algebraic Loops” on page 10-32) that contains the loop and `n` is the index of the loop in the system. To display the loop that contains the currently selected block, enter `ashow gcb`. To show a loop that contains a specified block, enter `ashow s:b`, where `s:b` is the block’s index. To clear algebraic-loop highlighting from the model diagram, enter `ashow clear`.

## Displaying Debugger Status

In GUI mode, the debugger displays the settings of various debug options, such as conditional breakpoints, in its **Status** panel. In command-line mode, the `status` command displays debugger settings. For example, the following sequence of commands displays the initial debug settings for the `vdp` model:

```

sim('vdp',[0,10],simset('debug','on'))
[Tm=0] **Start** of system 'vdp' outputs
(sldebug @0:0 'vdp/Integrator1'): status
  Current simulation time: 0 (MajorTimeStep)
  Last command: ""
  Stop in minor times steps is disabled.
  Break at zero crossing events is disabled.
  Break when step size is limiting by a state is disabled.
  Break on non-finite (NaN,Inf) values is disabled.
  Display of integration information is disabled.
  Algebraic loop tracing level is at 0.

```



# 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.

Commonly Used (p. A-3)	Blocks from other libraries that most models use.
Continuous (p. A-5)	Blocks that model linear functions.
Discontinuities (p. A-6)	Blocks whose outputs are discontinuous functions of their inputs.
Discrete (p. A-7)	Blocks that represent discrete-time functions.
Logic and Bit Operations (p. A-8)	Blocks that represent discrete-time functions.
Lookup Tables (p. A-10)	Blocks that use lookup tables to determine outputs from inputs.
Math Operations (p. A-11)	Blocks that model general mathematical functions.
Model Verification (p. A-13)	Blocks that enable you to create self-validating models.
Model-Wide Utilities (p. A-15)	Various utility blocks.
Ports & Subsystems (p. A-16)	Blocks for creating various types of subsystems.
Signal Attributes (p. A-18)	Blocks that modify or output attributes of signals.

Signal Routing (p. A-19)	Blocks that route signals from one point in a block diagram to another.
Sinks (p. A-20)	Blocks that display or write block output.
Sources (p. A-21)	Blocks that generate signals.
User-Defined Functions (p. A-23)	Blocks that allow you to define the function that relates inputs to outputs.
Additional Discrete (p. A-24)	Additional blocks for modeling discrete systems.
Additional Math (p. A-26)	Blocks that perform math operations useful for modeling digital systems.
Simulink Extras (p. A-27)	Blocks that perform specialized operations.

## Commonly Used

The Commonly Used library contains blocks from other libraries that most models use.

Block Name	Purpose
Bus Creator	Create a signal bus.
Bus Selector	Select signals from an incoming bus.
Constant	Generate a constant value.
Data Type Conversion	Convert an input signal to a specified data type.
Demux	Extract and output the elements of a bus or vector signal.
Discrete Time Integrator	Perform discrete-time integration or accumulation of a signal.
Gain	Multiply the input by a constant.
Ground	Ground an unconnected input port.
Import	Create an input port for a subsystem or an external input.
Integrator	Integrate a signal.
Logical Operator	Perform the specified logical operation on the input.
Mux	Combine several input signals into a vector or bus output signal.
Outport	Create an output port for a subsystem or an external output.
Product	Multiply or divide inputs.
Relational Operator	Perform the specified relational operation on the inputs.
Saturation	Limit the range of a signal.
Scope	Display signals generated during a simulation.
Subsystem	Represent a system within another system.

<b>Block Name</b>	<b>Purpose</b>
Sum	Add or subtract inputs.
Switch	Switch output between the first input and the third input based on the value of the second input.
Terminator	Terminate an unconnected output port.
Unit Delay	Delay a signal one sample period.

## Continuous

The Continuous library contains blocks that model linear functions.

Block Name	Purpose
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 Time 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.

Block Name	Purpose
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.
Dead Zone Dynamic	Set inputs within dynamically determined bounds to zero.
Hit Crossing	Detect crossing point.
Quantizer	Discretize input at a specified interval.
Rate Limiter	Limit the rate of change of a signal.
Rate Limiter Dynamic	Limit the rising and falling rates of the signal.
Relay	Switch output between two constants.
Saturation	Limit the range of a signal.
Saturation Dynamic	Bound the range of the input to limits that can change with time.
Wrap To Zero	Set output to zero if input is above threshold.

# Discrete

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

Block Name	Purpose
Difference	Calculate the change in a signal over one time step.
Discrete Derivative	Compute a discrete time derivative.
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.
Tapped Delay	Delay a scalar signal for multiple sample periods and output all the delayed versions.
Transfer Fcn First Order	Implement a discrete-time first-order transfer function.
Transfer Fcn Lead or Lag	Implement a discrete-time lead or lag compensator.
Transfer Fcn Real Zero	Implement a discrete-time transfer function that has a real zero and no pole.
Unit Delay	Delay a signal one sample period.
Weighted Moving Average	Implement a weighted moving average.
Zero-Order Hold	Implement zero-order hold of one sample period.

## Logic and Bit Operations

The Logic and Bit Operations library contains blocks that apply logic and bit operations to their inputs.

Block Name	Purpose
Bit Clear	Set the specified bit of the stored integer to zero.
Bitwise Operator	Perform the specified bitwise operation on the inputs.
Combinatorial Logic	Implement a truth table.
Compare To Constant	Determine how a signal compares to the specified constant.
Compare To Zero	Determine how a signal compares to zero.
Detect Change	Detect a change in a signal's value.
Detect Decrease	Detect a decrease in a signal's value.
Detect Fall Negative	Detect a falling edge when the signal's value decreases to a strictly negative value, and its previous value was nonnegative.
Detect Fall Nonpositive	Detect a falling edge when the signal's value decreases to a nonpositive value, and its previous value was strictly positive.
Detect Increase	Detect an increase in a signal's value.
Detect Rise Nonnegative	Detect a rising edge when a signal's value increases to a nonnegative value, and its previous value was strictly negative.
Detect Rise Positive	Detect a rising edge when a signal's value increases to a strictly positive value, and its previous value was nonpositive.
Extract Bits	Output a selection of contiguous bits from the input signal.
Interval Test	Determine if a signal is in a specified interval.
Interval Test Dynamic	Determine if a signal is in a specified interval whose limits can change.

Block Name	Purpose
Logical Operator	Perform the specified logical operation on the input.
Relational Operator	Perform the specified relational operation on the inputs.
Shift Arithmetic	Shift the bits and/or binary point of a signal.

## Lookup Tables

The Lookup Tables library contains blocks that use lookup tables to determine outputs from inputs.

Block Name	Purpose
Cosine	Implement a cosine function in fixed point using a lookup table approach that exploits quarter wave symmetry.
Direct Lookup Table (n-D)	Index into an N-dimensional table to retrieve a scalar, vector, or 2-D matrix.
Interpolation (n-D) Using PreLookup	Perform high-performance constant or linear interpolation.
Lookup Table	Perform piecewise linear mapping of the input.
Lookup Table (2-D)	Perform piecewise linear mapping of two inputs.
Lookup Table (n-D)	Perform piecewise linear or spline mapping of two or more inputs.
Lookup Table Dynamic	Approximate a one-dimensional function using a selected lookup method and a dynamically specified table.
PreLookup Index Search	Perform index search and interval fraction calculation for input on a breakpoint set.
Sine	Implement a sine wave in fixed point using a lookup table approach that exploits quarter wave symmetry.

# Math Operations

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

Block Name	Purpose
Abs	Output the absolute value of the input.
Add	Add or subtract inputs.
Algebraic Constraint	Constrain the input signal to zero.
Assignment	Assign values to specified elements of a signal.
Bias	Add a bias to the input.
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.
Divide	Multiply or divide inputs.
Dot Product	Generate the dot product.
Gain, Matrix Gain	Multiply block input by a specified value.
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.
MinMax Running Resettable	Determine the minimum or maximum of a signal over time.
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.
Reshape	Change the dimensionality of a signal.

<b>Block Name</b>	<b>Purpose</b>
Rounding Function	Perform a rounding function.
Sign	Indicate the sign of the input.
Sine Wave Function	Output a sine wave.
Slider Gain	Vary a scalar gain using a slider.
Subtract	Add or subtract inputs.
Sum	Generate the sum of inputs.
Sum of Elements	Add or subtract inputs.
Trigonometric Function	Perform a trigonometric function.
Unary Minus	Negate the input.
Weighted Sample Time	Support calculations involving sample time.
Math	

# Model Verification

## Acknowledgment

The Model Verification blocks were developed in conjunction with the Control System Design team of the Advanced Chassis System Development 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.

<b>Block Name</b>	<b>Purpose</b>
Check Static Range	Check that the input signal falls in a fixed range of amplitudes.
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.

Block Name	Purpose
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
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	Represent an enabled and triggered subsystem.
Enabled Subsystem	Represent an enabled subsystem.
For Iterator Subsystem	Implement a C-like <code>for</code> loop.
Function-Call Generator	Execute a function-call subsystem a specified number of times at a specified rate
Function-Call Subsystem	Represent a function-call subsystem.
If	Implement C-like <code>if-else</code> statement logic.
If Action Subsystem	Represent a subsystem whose execution is triggered by an If block.
Import	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.
Model	Include a model as a block in another model.
Outport	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.

Block Name	Purpose
Subsystem, Atomic Subsystem	Represent a system within another system.
Switch Case	Implement 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 that this block resides inside the Triggered Subsystem and the Enabled and Triggered Subsystem in the Subsystems library.
Triggered Subsystem	Represent a triggered 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.

Block Name	Purpose
Data Type Conversion	Convert a signal to another data type.
Data Type Conversion Inherited	Convert from one data type to another using inherited data type and scaling.
Data Type Duplicate	Force all inputs to the same data type.
Data Type Propagation	Set the data type and scaling of the propagated signal based on information from the reference signals.
Data Type Scaling Strip	Remove scaling and map to a built-in integer.
IC	Set the initial value of a signal.
Probe	Output a signal's attributes, including width, sample time, and/or signal type.
Rate Transition	Specify the data transfer mechanism between the data rates of a multirate system.
Signal Conversion	Convert a signal to a new type without altering signal values.
Signal Specification	Specify attributes of a signal.
Weighted Sample Time	Support calculations involving sample time.
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.

Block Name	Purpose
Bus Assignment	Assign values to specified elements of a bus.
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.
Environment Controller	Create branches of a block diagram that apply only to simulation or only to code generation.
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.
Index Vector	Switch output between different inputs based on the value of the first input.
Manual Switch	Switch between two inputs.
Merge	Combine several input lines into a scalar line.
Multiport 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.

Block Name	Purpose
Display	Show the value of the input.
Outport	Create an output port for a subsystem or an external output.
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.

Block Name	Purpose
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.
Counter Free-Running	Count up and overflow back to zero after the maximum value possible is reached for the specified number of bits.
Counter Limited	Count up and wrap back to zero after outputting the specified upper limit.
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.
Import	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.
Repeating Sequence Interpolated	Output discrete-time sequence and repeat, interpolating between data points.
Repeating Sequence Stair	Output and repeat the discrete time sequence.

<b>Block Name</b>	<b>Purpose</b>
Signal Builder	Generate an arbitrary piecewise linear signal.
Signal Generator	Generate various waveforms.
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.

Block Name	Purpose
Embedded MATLAB Function	Include MATLAB code in models that generate embeddable C code.
Fcn	Apply a specified expression to the input.
Level-2 M-File S-Function	Use a Level-2 M-file S-function in a model.
MATLAB Fcn	Apply a MATLAB function or expression to the input.
M-File S-Function	Use a Level-2 M-file S-function in a model.
S-Function	Access an S-function.
S-Function Builder	Build a C MEX S-function from specifications and code that you supply.

## Additional Discrete

The Additional Discrete library contains blocks for modeling discrete systems.

Block Name	Purpose
Fixed-Point State-Space	Implement discrete-time state space.
Transfer Fcn Direct Form II	Implement a Direct Form II realization of a transfer function.
Transfer Fcn Direct Form II Time Varying	Implement a time varying Direct Form II realization of a transfer function.
Unit Delay Enabled	Delay a signal one sample period, if the external enable signal is on.
Unit Delay Enabled External IC	Delay a signal one sample period, if the external enable signal is on, with an external initial condition.
Unit Delay Enabled Resettable	Delay a signal one sample period, if the external enable signal is on, with an external Boolean reset.
Unit Delay Enabled Resettable External IC	Delay a signal one sample period, if the external enable signal is on, with an external Boolean reset and initial condition.
Unit Delay External IC	Delay a signal one sample period, with an external initial condition.
Unit Delay Resettable	Delay a signal one sample period, with an external Boolean reset.
Unit Delay Resettable External IC	Delay a signal one sample period, with an external Boolean reset and initial condition.
Unit Delay With Preview Enabled	Output the signal and the signal delayed by one sample period, if the external enable signal is on.
Unit Delay With Preview Enabled Resettable	Output the signal and the signal delayed by one sample period, if the external enable signal is on, with an external Boolean reset.

Block Name	Purpose
Unit Delay With Preview Enabled Resettable External RV	Output the signal and the signal delayed by one sample period, if the external enable signal is on, with an external RV reset.
Unit Delay With Preview Resettable	Output the signal and the signal delayed by one sample period, with an external Boolean reset.
Unit Delay With Preview Resettable External RV	Output the signal and the signal delayed by one sample period, with an external RV reset.

## Additional Math

The Additional Math library contains math blocks useful for modeling digital systems.

Block Name	Purpose
Decrement Real World	Decrease the real world value of the signal by one.
Decrement Stored Integer	Decrease the stored integer value of a signal by one.
Decrement Time To Zero	Decrease the real-world value of the signal by the sample time, but only to zero.
Decrement To Zero	Decreases the real-world value of a signal by one, but only to zero.
Increment Real World	Increase the real world value of the signal by one.
Increment Stored Integer	Increase the stored integer value of a signal by one.

## Simulink Extras

The Extras block library contains specialized blocks.

## **A** Block Libraries

---

## A

Abs block  
    zero crossings 2-23  
absolute tolerance  
    definition 7-15  
    simulation accuracy 7-41  
Adams-Basforth-Moulton PECE solver 7-14  
algebraic loops  
    direct feedthrough blocks 2-24  
    displaying 2-26  
    highlighting 10-35  
    identifying blocks in 10-32  
    simulation speed 7-41  
aligning blocks 5-5  
annotations  
    changing font 4-20  
    creating 4-19  
    definition 4-19  
    deleting 4-19  
    editing 4-19  
    moving 4-19  
Apply button on Mask Editor 9-15  
Assignment mask parameter 9-21  
atomic subsystem 2-11  
attributes format string 4-18  
**AttributesFormatString** block  
    parameter 5-14  
Autoscale icon drawing coordinates 9-18

## B

Backlash block  
    zero crossings 2-23  
backpropagating sample time 2-38  
Backspace key  
    deleting annotations 4-19  
    deleting blocks 5-6  
    deleting labels 6-22  
Band-Limited White Noise block  
    simulation speed 7-41

block callback parameters 4-42  
Block data tips 5-2  
block descriptions  
    creating 9-10  
block diagram  
    updating 3-9  
block diagrams  
    panning 3-7  
    printing 3-15  
    zooming 3-7  
block libraries  
    Blocksets and Toolboxes A-27  
    definition 5-21  
    Extras A-27  
    searching 5-30  
block names  
    changing location 5-18  
    copied blocks 5-4  
    editing 5-17  
    flipping location 5-18  
    generated for copied blocks 5-5  
    hiding and showing 5-18  
    location 5-17  
    rules 5-17  
block parameters  
    about 5-7  
    modifying during simulation 7-5  
    scalar expansion 6-16  
    setting 5-8  
Block Properties dialog box 5-10  
block type of masked block 9-29  
blocks  
    aligning 5-5  
    autoconnecting 4-10  
    callback routines 4-41  
    changing font 5-17  
    changing font names 5-17  
    changing location of names 5-18  
    checking connections 2-14  
    connecting automatically 4-10

connecting manually 4-13  
copying from Library Browser 5-30  
copying into models 5-4  
copying to other applications 5-5  
deleting 5-6  
disconnecting 4-18  
drop shadows 5-16  
duplicating 5-6  
grouping to create subsystem 4-25  
hiding block names 5-18  
input ports with direct feedthrough 2-24  
library 5-21  
moving between windows 5-5  
moving in a model 5-6  
names  
    editing 5-17  
orientation 5-15  
reference 5-21  
resizing 5-15  
showing block names 5-18  
signal flow through 5-15  
under mask 9-15  
updating 2-14  
<>blocks 5-17  
    *See also* block names  
Blocksets and Toolboxes library A-27  
Bogacki-Shampine formula 7-14  
books  
    MATLAB related 1-8  
bounding box  
    grouping blocks for subsystem 4-25  
    selecting objects 4-4  
branch lines 4-14  
Break Library Link menu item 5-25  
breaking links to library block 5-25  
breakpoints  
    setting 10-22  
    setting at end of block 10-25  
    setting at timesteps 10-25  
    setting on nonfinite values 10-25

setting on step-size-limiting steps 10-26  
setting on zero crossings 10-26  
Browser 8-22

**C**

callback routines 4-41  
callback routines, referencing mask  
    parameters in 4-43  
callback tracing 4-41  
Cancel button on Mask Editor 9-15  
canvas, editor 3-6  
changing  
    signal labels font 6-22  
Clear menu item 5-6  
CloseFcn block callback parameter 4-43  
CloseFcn model callback parameter 4-42  
colors for sample times 2-38  
commands  
    undoing 3-6  
comp.soft-sys.matlab 1-9  
composite signals 6-8  
conditionally executed subsystem 2-11  
conditionally executed subsystems 4-29  
Configuration Parameters dialog box 7-36  
connecting blocks 4-13  
ConnectionCallback  
    port callback parameters 4-45  
constant sample time 2-39  
context menu 3-6  
continuous sample time 2-32  
control flow subsystem 4-29  
control input 4-29  
control signal 4-29 6-8  
copy  
    definition 5-21  
Copy menu item 5-4  
CopyFcn block callback parameter 4-43  
copying  
    blocks 5-4

- 
- signal labels 6-22  
 Create Mask menu item 9-15  
 Cut menu item 5-5
- D**
- dash-dot lines 6-8  
 dbstop if error command 9-28  
 dbstop if warning command 9-28  
 Dead Zone block  
     zero crossings 2-23  
 debugger  
     running incrementally 10-15  
     setting breakpoints 10-22  
     setting breakpoints at time steps 10-25  
     setting breakpoints at zero crossings 10-26  
     setting breakpoints on nonfinite  
         values 10-25  
     setting breakpoints on step-size-limiting  
         steps 10-26  
     skipping breakpoints 10-19  
     starting 10-10  
     stepping by time steps 10-17  
 debugging initialization commands 9-30  
 decimation factor  
     saving simulation output 7-26  
 Delete key  
     deleting blocks 5-6  
     deleting signal labels 6-22  
 DeleteFcn block callback parameter 4-43  
 demos  
     Simulink 1-5  
 description of masked blocks 9-29  
 DestroyFcn block callback parameter 4-43  
 diagnosing simulation errors 7-38  
 diagonal line segments 4-15  
 diagonal lines 4-13  
 direct feedthrough blocks 2-24  
 disabled subsystem  
     output 4-31
- disabling zero-crossing detection 2-22  
 disconnecting blocks 4-18  
 discrete blocks  
     in enabled subsystem 4-32  
     in triggered systems 4-37  
 discrete sample time 2-32  
 Discrete-Time Integrator block  
     sample time colors 2-38  
 discrete-time systems 2-31  
 Documentation pane of Mask Editor 9-15  
 Dormand-Prince  
     pair 7-13  
 drawing coordinates  
     Autoscale 9-18  
     normalized 9-18  
     Pixel 9-19  
 drop shadows 5-16  
 duplicating blocks 5-6
- E**
- editor 3-5  
     canvas 3-6  
     toolbar 3-5  
 either trigger event 4-34  
 Enable block  
     creating enabled subsystems 4-31  
     outputting enable signal 4-32  
     states when enabling 4-32  
 enabled subsystems 4-30  
     setting states 4-32  
 ending Simulink session 3-27  
 error tolerance 7-15  
     simulation accuracy 7-41  
     simulation speed 7-40  
 ErrorFcn block callback parameter 4-44  
 eval command  
     masked block help 9-29  
 examples  
     masking 9-6

multirate discrete model 2-34  
 Exit MATLAB menu item 3-27  
 Extras block library A-27

**F**

falling trigger event 4-34  
 Fcn block  
     simulation speed 7-40  
 files  
     writing to 7-4  
 Final State check box 7-25  
 fixed in minor time step 2-32  
 fixed-step solvers  
     definition 2-17  
 Flip Block menu item 5-15  
 Flip Name menu item 5-18  
 floating Display block 7-5  
 floating Scope block 7-5  
 font  
     annotations 4-20  
     block 5-17  
     block names 5-17  
     signal labels 6-22  
 Font menu item  
     changing block name font 5-17  
     changing the font of a signal label 6-22  
 font size, setting for Model Explorer 8-3  
 font size, setting for Simulink dialog boxes 8-3  
 From Workspace block  
     zero crossings 2-23  
 fundamental sample time 7-8

**G**

Gain block  
     algebraic loops 2-24  
 Go To Library Link menu item 5-26  
 grouping blocks 4-24

**H**

handles on selected object 4-4  
 held output of enabled subsystem 4-31  
 held states of enabled subsystem 4-32  
 help  
     sources of 1-7  
     via newsgroup 1-9  
 Help button on Mask Editor 9-15  
 help text for masked blocks 9-10  
 Hide Name menu item  
     hiding block names 5-18  
     hiding port labels 4-28  
 Hide Port Labels menu item 4-27  
 hiding block names 5-18  
 hierarchy of model  
     replacing virtual subsystems 2-14  
 Hit Crossing block  
     notification of zero crossings 2-21  
     zero crossings  
         and Disable zero crossing detection option 2-23  
 hybrid systems  
     integrating 2-41

**I**

Icon pane of Mask Editor 9-14  
 icons  
     creating for masked blocks 9-16  
 If block  
     zero crossings  
         and Disable zero crossing detection option 2-23  
 inherited sample time 2-32  
 InitFcn block callback parameter 4-43  
 InitFcn model callback parameter 4-42  
 initial conditions  
     specifying 7-24  
 Initial State check box 7-26  
 initial states

- loading 7-26
  - initial step size
    - simulation accuracy 7-41
  - initialization commands 9-25
    - debugging 9-30
  - Initialization pane of Mask Editor 9-15
  - Import block
    - in subsystem 4-24
    - supplying input to model 7-17
  - inputs
    - loading from base workspace 7-17
    - mixing vector and scalar 6-17
    - scalar expansion 6-16
  - Integrator block
    - algebraic loops 2-24
    - sample time colors 2-38
    - simulation speed 7-41
    - zero crossings 2-23
  - invariant constants 2-39
- J**
- Jacobian matrices 7-14
- K**
- keyboard actions summary 3-23
  - keyboard command 9-28
- L**
- labeling signals 6-21
  - labeling subsystem ports 4-27
  - learning Simulink 1-5
  - `libinfo` command 5-28
  - libraries, *see* block libraries
  - library blocks
    - breaking links to 5-25
    - definition 5-21
    - finding 5-26
    - getting information about 5-26
- Library Browser 5-28
    - copying blocks from 5-30
  - library links
    - creating 5-22
    - definition 5-21
    - disabling 5-23
    - displaying 5-27
    - modifying 5-23
    - propagating changes to 5-24
    - showing in Model Browser 8-23
    - status of 5-26
    - unresolved 5-22
  - line segments 4-14
    - diagonal 4-15
    - moving 4-15
  - line vertices
    - moving 4-16
  - lines
    - branch 4-14
    - connecting blocks 4-10
    - diagonal 4-13
    - moving 5-6
    - signals carried on 7-5
  - links
    - breaking 5-25
    - to library block 5-22
  - LinkStatus block parameter 5-26
  - LoadFcn block callback parameter 4-44
  - loading from base workspace 7-17
  - loading initial states 7-26
  - location of block names 5-17
  - Look Under Mask menu item 9-15
  - loops, algebraic, *see* algebraic loops
- M**
- M-file S-functions
    - simulation speed 7-40
  - Mask Editor 9-14
  - mask help text 9-10

- Mask Subsystem menu item 9-14  
mask type  
    defining 9-10  
mask workspace 9-27  
masked blocks  
    block descriptions 9-10  
    documentation 9-28  
    help text 9-10  
    icons  
        creating 9-10  
        Icon pane 9-16  
    initialization commands 9-25  
    looking under 9-15  
    parameters  
        assigning values to 9-21  
        default values 9-30  
        mapping 9-6  
        prompts for 9-20  
        referencing in callbacks 4-43  
    showing in Model Browser 8-23  
    type 9-29  
    unmasking 9-15  
masked subsystems  
    showing in Model Browser 8-23  
Math Function block  
    algebraic loops 2-24  
MathWorks Store  
    purchasing products from 1-8  
MathWorks Web site 1-8  
MATLAB  
    books 1-8  
    terminating 3-27  
MATLAB Central 1-9  
MATLAB Fcn block  
    simulation speed 7-40  
mdl files 3-11  
Memory block  
    simulation speed 7-40  
menus 3-5  
    context 3-6
- MinMax block  
    zero crossings 2-23  
mixed continuous and discrete systems 2-41  
model  
    editor 3-5  
Model Browser 8-22  
    showing library links in 8-23  
    showing masked subsystems in 8-23  
model callback parameters 4-41  
model configuration preferences 8-4  
Model Explorer  
    font size 8-3  
model file name, maximum size of 3-11  
model files  
    mdl file 3-11  
model navigation commands 4-26  
model verification A-13  
Model Verification block library A-13  
ModelCloseFcn block callback parameter 4-44  
models  
    callback routines 4-41  
    creating 4-2  
    creating templates 4-2  
    editing 3-3  
    navigating 4-26  
    printing 3-15  
    saving 3-11  
        selecting entire 4-5  
models, self-validating A-13  
mouse actions summary 3-23  
MoveFcn block callback parameter 4-44  
multirate systems  
    example 2-34  
Mux block A-3

## N

- NameChangeFcn block callback parameter 4-44  
names  
    blocks 5-17

copied blocks 5-4  
 New Library menu item 5-21  
 New menu item 4-2  
 newsgroup 1-9  
 normalized icon drawing coordinates 9-18  
 numerical differentiation formula 7-14  
 numerical integration 2-15

**O**

objects  
   selecting more than one 4-4  
   selecting one 4-4  
 ode113 solver  
   advantages 7-14  
   hybrid systems 2-41  
   Memory block  
     and simulation speed 7-40  
 ode15s solver  
   advantages 7-14  
   and stiff problems 7-40  
   hybrid systems 2-41  
   Memory block  
     and simulation speed 7-40  
   unstable simulation results 7-41  
 ode23 solver 7-14  
   hybrid systems 2-41  
 ode23s solver  
   advantages 7-14  
   simulation accuracy 7-41  
 ode45 solver  
   hybrid systems 2-41  
 Open menu item 3-3  
 OpenFcn block callback parameter  
   purpose 4-44  
 opening  
   Subsystem block 4-26  
 orientation of blocks 5-15  
 Outport block  
   in subsystem 4-24

Outport blocks A-3  
 output  
   additional 7-27  
   between trigger events 4-36  
   disabled subsystem 4-31  
   enable signal 4-32  
   options 7-26  
   saving to workspace 7-22  
   smoother 7-26  
   specifying for simulation 7-27  
   trigger signal 4-37  
   writing to file  
     when written 7-4  
   writing to workspace 7-22  
     when written 7-4  
 output ports  
   Enable block 4-32  
   Trigger block 4-37

**P**

panning block diagrams 3-7  
 PaperOrientation model parameter 3-18  
 PaperPosition model parameter 3-18  
 PaperPositionMode model parameter 3-18  
 PaperType model parameter 3-18  
 parameters  
   block 5-7  
   setting values of 5-8  
   tunable  
     definition 2-8  
 Parameters pane of Mask Editor 9-15  
 ParentCloseFcn block callback  
   parameter 4-44  
 Paste menu item 5-4  
 Pixel icon drawing coordinates 9-19  
 ports  
   block orientation 5-15  
   labeling in subsystem 4-27  
 PostLoadFcn model callback parameter 4-42

**PostSaveFcn** block callback parameter 4-45  
**PostSaveFcn** model callback parameter 4-42  
**PostScript** files  
  printing to 3-17  
**preferences**, model configuration 8-4  
**PreLoadFcn** model callback parameter 4-42  
**PreSaveFcn** block callback parameter 4-44  
**PreSaveFcn** model callback parameter 4-42  
**print** command 3-15  
**Print** menu item 3-15  
  printing to PostScript file 3-17  
**produce additional output** option 7-27  
**produce specified output only** option 7-27  
**Product** block  
  algebraic loops 2-24  
**professional version**  
  differences with Student Version 1-10  
**purely discrete systems** 2-34

**Q**

**Quit** MATLAB menu item 3-27

**R**

**Random Number** block  
  simulation speed 7-41  
**Redo** menu item 3-7  
**reference blocks**  
  definition 5-21  
**reference information**  
  obtaining 1-6  
**refine factor**  
  smoothing output 7-26  
**Relational Operator** block  
  zero crossings 2-23  
**relative tolerance**  
  definition 7-15  
  simulation accuracy 7-41  
**Relay** block

**zero crossings** 2-23  
**reset**  
  output of enabled subsystem 4-31  
  states of enabled subsystem 4-32  
**resizing blocks** 5-15  
**rising trigger event** 4-34  
**Rosenbrock formula** 7-14  
**Rotate Block** menu item 5-15  
**Runge-Kutta (2,3) pair** 7-14  
**Runge-Kutta (4,5) formula** 7-13

**S**

**sample time**  
  backpropagating 2-38  
  changing during simulation 2-33  
  colors 2-38  
  constant 2-39  
  continuous 2-32  
  discrete 2-32  
  fixed in minor time step 2-32  
  fundamental 7-8  
  inherited 2-32  
  simulation speed 7-41  
  variable 2-33  
**Sample Time Colors** menu item 2-39  
  updating coloring 4-9  
**sampled data systems** 2-31  
**Saturation** block A-3  
  zero crossings 2-23  
  how used 2-21  
**Save As** menu item 3-11  
**Save** menu item 3-11  
**Save options area** 7-22  
**save\_system** command  
  breaking links 5-25  
**scalar expansion** 6-16  
**Select All** menu item 4-5  
**Set Font** dialog box 5-17  
**set\_param** command

breaking link 5-25  
running a simulation 7-2  
setting breakpoints 10-22  
Shampine, L. F. 7-14  
Show Name menu item 5-18  
show output port  
    Enable block 4-32  
    Trigger block 4-37  
showing block names 5-18  
Sign block  
    zero crossings 2-23  
Signal Builder block  
    zero crossings 2-24  
signal buses 6-13  
signal flow through blocks 5-15  
signal labels  
    changing font 6-22  
    copying 6-22  
    creating 6-21  
    deleting 6-22  
    editing 6-22  
    moving 6-21  
signal propagation 6-5  
signals  
    composite 6-8  
    labeling 6-21  
    labels 6-21  
    names 6-21  
    virtual 6-5  
signals, creating 6-2  
simulation  
    accuracy 7-41  
    displaying information about  
        algebraic loops 10-30  
        block execution order 10-32  
        block I/O 10-28  
        debug settings 10-35  
        integration 10-31  
        nonvirtual blocks 10-34  
        nonvirtual systems 10-33  
        system states 10-31  
        zero crossings 10-34  
    execution phase 2-15  
    parameters  
        specifying 7-38  
    running incrementally 10-15  
    running nonstop 10-19  
    speed 7-40  
    status bar 3-6  
    stepping by breakpoints 10-22  
    stepping by time steps 10-17  
    unstable results 7-41  
Simulation Diagnostics Viewer 7-38  
simulation errors  
    diagnosing 7-38  
simulation time  
    compared to clock time 7-6  
    writing to workspace 7-22  
Simulink  
    demos 1-5  
    ending session 3-27  
    how to learn 1-5  
    icon 3-2  
    menus 3-5  
    starting 3-2  
    terminating 3-27  
Simulink block library, *see* block libraries  
simulink command  
    starting Simulink 3-2  
Simulink dialog boxes  
    font size 8-3  
Simulink Library Browser 3-2

size of block  
    changing 5-15  
sldebug command  
    starting the Simulink debugger 10-13  
solvers  
    fixed-step  
        definition 2-17  
    ode113  
        advantages 7-14  
        and simulation speed 7-40  
    ode15s  
        advantages 7-14  
        and simulation speed 7-40  
        and stiff problems 7-40  
        simulation accuracy 7-41  
    ode23 7-14  
    ode23s  
        advantages 7-14  
        simulation accuracy 7-41  
speed of simulation 7-40  
stairs function 2-35  
start time 7-6  
StartFcn block callback parameter 4-45  
StartFcn model callback parameter 4-42  
starting Simulink 3-2  
State-Space block  
    algebraic loops 2-24  
states  
    between trigger events 4-36  
    loading initial 7-26  
    when enabling 4-32  
    writing to workspace 7-22  
status bar 3-6  
Step block  
    zero crossings 2-24  
step size  
    simulation speed 7-40  
stiff problems 7-14  
stiff systems  
    simulation speed 7-40  
stop time 7-6  
StopFcn block callback parameter 4-45  
StopFcn model callback parameter 4-42  
Student Version  
    differences with professional version 1-10  
    Simulink differences 1-10  
subsystem  
    atomic 2-11  
    conditionally executed 2-11  
Subsystem block  
    adding to create subsystem 4-24  
    opening 4-26  
    zero crossings 2-24  
subsystem ports  
    labeling 4-27  
subsystems  
    creating 4-24  
    displaying parent of 4-27  
    labeling ports 4-27  
    opening 4-26  
    triggered and enabled 4-37  
    underlying blocks 4-26  
    undoing creation of 4-26  
Sum block  
    algebraic loops 2-24  
summary of mouse and keyboard actions 3-23  
support  
    sources of 1-7  
Switch block  
    zero crossings 2-24  
Switch Case block  
    zero crossings  
        and Disable zero crossing detection  
        option 2-24

**T**

technical support 1-9  
terminating MATLAB 3-27  
terminating Simulink 3-27

- terminating Simulink session 3-27  
 time interval  
     simulation speed 7-40  
 toolbar  
     editor 3-5  
 Transfer Fcn block  
     algebraic loops 2-24  
 Trigger block  
     creating triggered subsystem 4-36  
     outputting trigger signal 4-37  
     showing output port 4-37  
 triggered and enabled subsystems 4-37  
 triggered subsystems 4-34  
 triggers  
     control signal  
         outputting 4-37  
     either 4-34  
     events 4-34  
     falling 4-34  
     input 4-34  
     rising 4-34  
     type parameter 4-36  
 troubleshooting 1-7  
 tunable parameters  
     definition 2-8
- U**  
 Undo menu item 3-6  
 UndoDeleteFcn block callback parameter 4-45  
 undoing commands 3-6  
 Unmask button on Mask Editor 9-15  
 unstable simulation results 7-41  
 Update Diagram menu item  
     fixing bad link 5-23  
     out-of-date reference block 5-25  
     recoloring model 4-9  
 updating a block diagram 3-9
- URL specification in block help 9-29
- V**  
 vector length  
     checking 2-14  
 verification, model A-13  
 vertices  
     moving 4-16  
 virtual blocks 5-2  
 virtual signals 6-5
- W**  
 web command  
     masked block help 9-29  
 window reuse 4-27  
 workspace  
     loading from 7-17  
     mask 9-27  
     saving to 7-22  
     writing to  
         simulation terminated or  
         suspended 7-4  
 www.mathworks.com 1-8  
 www.mathworks.com/academia 1-8  
 www.mathworks.com/store 1-8  
 www.mathworks.com/support/books 1-8
- Z**  
 zero crossings  
     Saturation block 2-21  
 zero-crossing slope method 4-30  
 Zero-Pole block  
     algebraic loops 2-24  
 zooming block diagrams 3-7