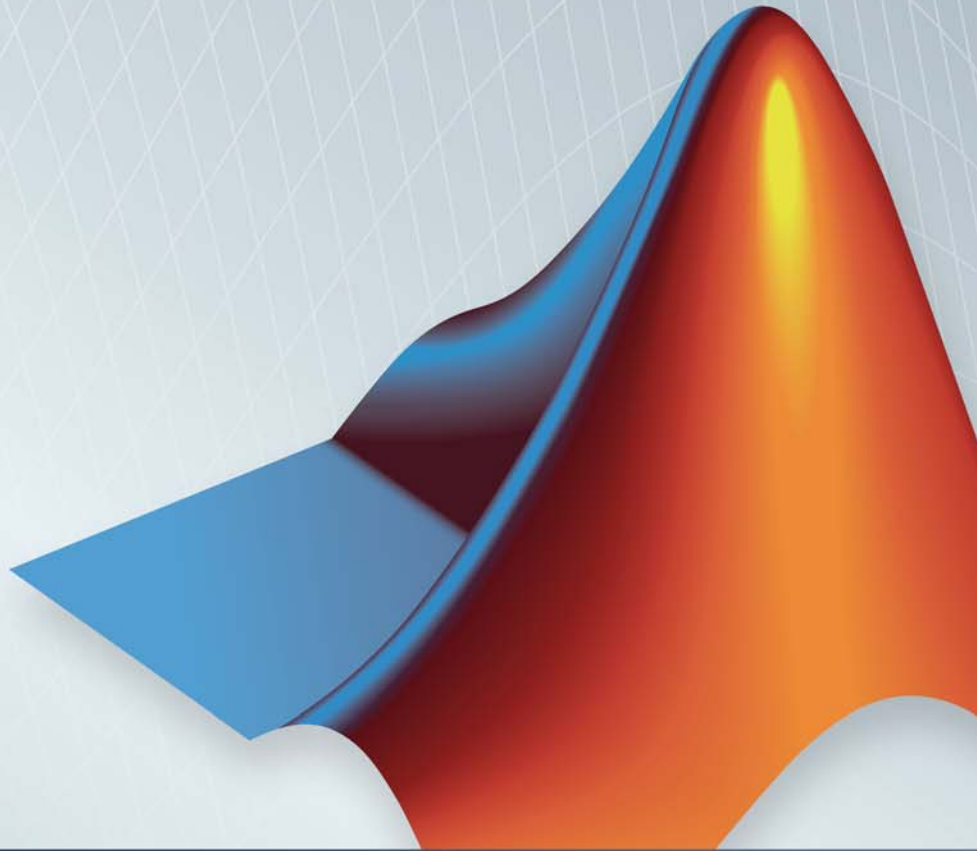


Simscape™

User's Guide

R2013b



MATLAB® & SIMULINK®



How to Contact MathWorks



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



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Simscape™ User's Guide

© COPYRIGHT 2007–2013 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 and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

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

Revision History

March 2007	Online only	New for Version 1.0 (Release 2007a)
September 2007	Online only	Revised for Version 2.0 (Release 2007b)
March 2008	Online only	Revised for Version 2.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.0 (Release 2008b)
March 2009	Online only	Revised for Version 3.1 (Release 2009a)
September 2009	Online only	Revised for Version 3.2 (Release 2009b)
March 2010	Online only	Revised for Version 3.3 (Release 2010a)
September 2010	Online only	Revised for Version 3.4 (Release 2010b)
April 2011	Online only	Revised for Version 3.5 (Release 2011a)
September 2011	Online only	Revised for Version 3.6 (Release 2011b)
March 2012	Online only	Revised for Version 3.7 (Release 2012a)
September 2012	Online only	Revised for Version 3.8 (Release 2012b)
March 2013	Online only	Revised for Version 3.9 (Release 2013a)
September 2013	Online only	Revised for Version 3.10 (Release 2013b)

Model Construction

Basic Principles of Modeling Physical Networks	1-2
Overview of the Physical Network Approach to Modeling	
Physical Systems	1-2
Variable Types	1-4
Building the Mathematical Model	1-5
Direction of Variables	1-6
Connector Ports and Connection Lines	1-8
Connecting Simscape Diagrams to Simulink Sources and Scopes	1-10
Simscape Block Libraries	1-11
Library Structure Overview	1-11
Using the Simulink Library Browser to Access the Block Libraries	1-13
Using the Command Prompt to Access the Block Libraries	1-14
Essential Physical Modeling Techniques	1-15
Building Your Model	1-15
Using the Conserving Ports	1-16
Using the Physical Signal Ports	1-17
Creating and Simulating a Simple Model	1-18
Building a Simscape Diagram	1-18
Modifying Initial Settings	1-26
Running the Simulation	1-27
Adjusting the Parameters	1-30
Modeling Best Practices	1-36
Grounding Rules	1-36
Avoiding Numerical Simulation Issues	1-40
Modeling Pneumatic Systems	1-44
Intended Applications	1-44

Assumptions and Limitations	1-44
Fundamental Equations	1-45
Network Variables	1-46
Connection Constraints	1-47
References	1-47

Thermal Liquid Models

2

Modeling Thermal Liquid Systems	2-2
When to Use Thermal Liquid Blocks	2-2
Modeling Workflow	2-3
Establish Model Requirements	2-3
Model Physical Components	2-4
Prepare Model for Analysis	2-5
Run Simulation	2-5
Thermal Liquid Library	2-7
Why Use Thermal Liquid Blocks?	2-7
Representing Thermal Liquid Components	2-7
Specifying Thermal Liquid Medium	2-9
Modeling Multidomain Systems	2-9
Thermal Liquid Modeling Framework	2-11
How Blocks Represent Components	2-11
How Ports Represent Interfaces	2-12
Thermal Flux Scheme	2-13
Heat Transfer in Insulated Oil Pipeline	2-15
Oil Pipelines	2-15
Modeling Considerations	2-16
Simscape Model	2-18
Run Simulation	2-20
Run Optimization Script	2-27

How Simscape Models Represent Physical Systems . . .	3-2
Representations of Physical Systems	3-2
Differential, Differential-Algebraic, and Algebraic Systems	3-2
Stiffness	3-3
Events and Zero Crossings	3-3
Working with Simscape Representation	3-3
How Simscape Simulation Works	3-5
Simscape Simulation Phases	3-5
Model Validation	3-7
Network Construction	3-7
Equation Construction	3-8
Initial Conditions Computation	3-8
Transient Initialization	3-9
Transient Solve	3-9
Setting Up Solvers for Physical Models	3-11
About Simulink and Simscape Solvers	3-11
Choosing Simulink and Simscape Solvers	3-11
Harmonizing Simulink and Simscape Solvers	3-13
Customizing Solvers for Physical Models	3-19
Important Concepts and Choices in Physical Simulation . .	3-19
Making Optimal Solver Choices for Physical Simulation . .	3-22
Troubleshooting Simulation Errors	3-28
Troubleshooting Tips and Techniques	3-28
System Configuration Errors	3-29
Numerical Simulation Issues	3-32
Initial Conditions Solve Failure	3-32
Transient Simulation Issues	3-33
Code Generation	3-35
About Code Generation from Simscape Models	3-35
Reasons for Generating Code	3-35
Using Code-Related Products and Features	3-36
How Simscape Code Generation Differs from Simulink . . .	3-36

Real-Time Simulation	3-39
What Is Real-Time Simulation?	3-39
Requirements for Real-Time Simulation	3-40
Simulating Physical Models in Real Time	3-41
Preparing a Model for Real-Time Simulation	3-42
Troubleshooting Real-Time Simulation Problems	3-45
Finding an Operating Point	3-48
What Is an Operating Point?	3-48
Some Operating Point Search Methods	3-49
Finding Operating Points in Physical Models	3-50
Linearizing at an Operating Point	3-55
What Is Linearization?	3-55
Some Linearization Methods	3-58
Linearizing a Physical Model	3-59
Linearize an Electronic Circuit	3-64
About the Nonlinear Bipolar Transistor Circuit	3-64
Finding Operating Points in a Transistor Circuit with the Simscape Solver	3-70
Linearizing a Transistor Circuit with Simulink and Related Software	3-71
Limitations	3-77
Sample Time and Solver Restrictions	3-77
Algebraic Loops	3-77
Restricted Simulink Tools	3-78
Unsupported Simulink Tools	3-80
Simulink Tools Not Compatible with Simscape Blocks ...	3-80
Code Generation	3-81
References	3-83

Data Logging

4

About Simulation Data Logging	4-2
--------------------------------------------	------------

Suggested Workflows	4-2
Limitations	4-2
How to Log Simulation Data	4-3
How to Enable Data Logging	4-3
Data Logging Options	4-4
Log and Plot Simulation Data	4-7
Log Simulation Statistics	4-13

Model Statistics

5

Simscape Model Statistics	5-2
1-D Physical System Statistics	5-4
3-D Multibody System Statistics	5-6
View Model Statistics	5-9

Physical Units

6

How to Work with Physical Units	6-2
Unit Definitions	6-4
How to Specify Units in Block Dialogs	6-10
Thermal Unit Conversions	6-12
About Affine Units	6-12

When to Apply Affine Conversion	6-12
How to Apply Affine Conversion	6-13
Angular Units	6-16
References	6-16
Units for Angular Velocity and Frequency	6-17

Add-On Product License Management

7

About the Simscape Editing Mode	7-2
Suggested Workflows	7-2
What You Can Do in Restricted Mode	7-3
What You Can Do in Full Mode	7-4
Switching Between Modes	7-4
Working with Block Libraries	7-7
Working with Restricted and Full Modes	7-9
Set the Model Loading Preference	7-9
Save a Model in Restricted Mode	7-10
Work with a Model in Restricted Mode	7-13
Switch from Restricted to Full Mode	7-22
Editing Mode Information	7-24
What Is the Current Mode?	7-24
Which Licenses Are Checked Out?	7-24

Index

Model Construction

- “Basic Principles of Modeling Physical Networks” on page 1-2
- “Simscape Block Libraries” on page 1-11
- “Essential Physical Modeling Techniques” on page 1-15
- “Creating and Simulating a Simple Model” on page 1-18
- “Modeling Best Practices” on page 1-36
- “Modeling Pneumatic Systems” on page 1-44

Basic Principles of Modeling Physical Networks

In this section...

“Overview of the Physical Network Approach to Modeling Physical Systems” on page 1-2

“Variable Types” on page 1-4

“Building the Mathematical Model” on page 1-5

“Direction of Variables ” on page 1-6

“Connector Ports and Connection Lines” on page 1-8

“Connecting Simscape Diagrams to Simulink Sources and Scopes” on page 1-10

Overview of the Physical Network Approach to Modeling Physical Systems

Simscape™ software is a set of block libraries and special simulation features for modeling physical systems in the Simulink® environment. It employs the Physical Network approach, which differs from the standard Simulink modeling approach and is particularly suited to simulating systems that consist of real physical components.

Simulink blocks represent basic mathematical operations. When you connect Simulink blocks together, the resulting diagram is equivalent to the mathematical model, or representation, of the system under design. Simscape technology lets you create a network representation of the system under design, based on the Physical Network approach. According to this approach, each system is represented as consisting of functional elements that interact with each other by exchanging energy through their ports.

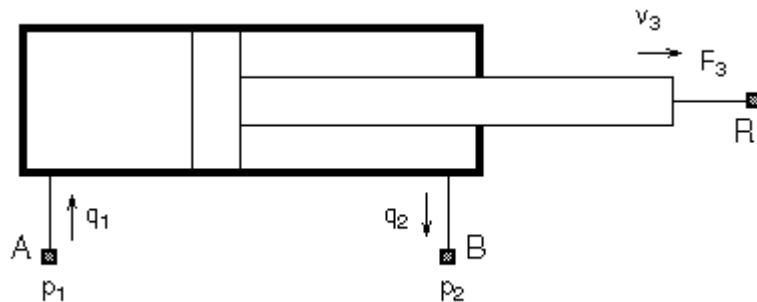
These connection ports are nondirectional. They mimic physical connections between elements. Connecting Simscape blocks together is analogous to connecting real components, such as pumps, valves, and so on. In other words, Simscape diagrams mimic the physical system layout. If physical components can be connected, their models can be connected, too. You do not have to specify flow directions and information flow when connecting Simscape blocks, just as you do not have to specify this information when you connect

real physical components. The Physical Network approach, with its Through and Across variables and nondirectional physical connections, automatically resolves all the traditional issues with variables, directionality, and so on.

The number of connection ports for each element is determined by the number of energy flows it exchanges with other elements in the system, and depends on the level of idealization. For example, a fixed-displacement hydraulic pump in its simplest form can be represented as a two-port element, with one energy flow associated with the inlet (suction) and the other with the outlet. In this representation, the angular velocity of the driving shaft is assumed constant, making it possible to neglect the energy exchange between the pump and the shaft. To account for a variable driving torque, you need a third port associated with the driving shaft.

An energy flow is characterized by its variables. Each energy flow is associated with two variables, one Through and one Across (see “Variable Types” on page 1-4 for more information). Usually, these are the variables whose product is the energy flow in watts. They are called the basic, or conjugate, variables. For example, the basic variables for mechanical translational systems are force and velocity, for mechanical rotational systems—torque and angular velocity, for hydraulic systems—flow rate and pressure, for electrical systems—current and voltage.

The following example illustrates a Physical Network representation of a double-acting hydraulic cylinder.



The element is represented with three energy flows: two flows of hydraulic energy through the inlet and outlet of the cylinder and a flow of mechanical

energy associated with the rod motion. It therefore has the following three connector ports:

- A — Hydraulic conserving port associated with pressure p_1 (an Across variable) and flow rate q_1 (a Through variable)
- B — Hydraulic conserving port associated with pressure p_2 (an Across variable) and flow rate q_2 (a Through variable)
- R — Mechanical translational conserving port associated with rod velocity v_3 (an Across variable) and force F_3 (a Through variable)

See “Connector Ports and Connection Lines” on page 1-8 for more information on connector port types.

Variable Types

Physical Network approach supports two types of variables:

- Through — Variables that are measured with a gauge connected in series to an element.
- Across — Variables that are measured with a gauge connected in parallel to an element.

The following table lists the Through and Across variables associated with each type of physical domain in Simscape software:

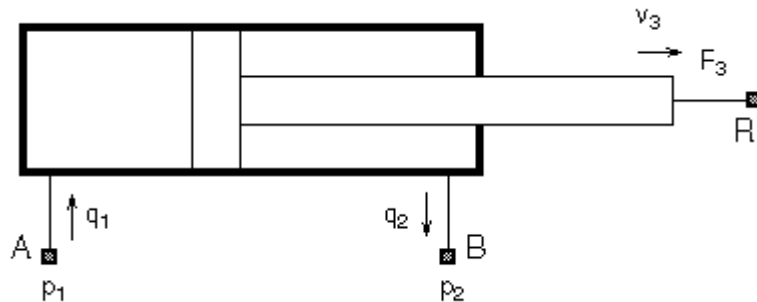
Physical Domain	Across Variable	Through Variable
Electrical	Voltage	Current
Hydraulic	Pressure	Flow rate
Magnetic	Magnetomotive force (mmf)	Flux
Mechanical rotational	Angular velocity	Torque
Mechanical translational	Translational velocity	Force
Pneumatic	Pressure and temperature	Mass flow rate and heat flow

Physical Domain	Across Variable	Through Variable
Thermal	Temperature	Heat flow
Thermal liquid	Pressure and temperature	Mass flow rate and thermal flux

Note Generally, the product of each pair of Across and Through variables associated with a domain is power (energy flow in watts). The exceptions are pneumatic domain, where the product of pressure and mass flow rate is not power, magnetic domain, where the product of mmf and flux is not power, but energy, and the thermal liquid domain, where products of both variable pairs are not power. These result in a pseudo-bond graph.

Building the Mathematical Model

Through and Across variables associated with all the energy flows form the basis of the mathematical model of the block.



For example, the model of a double-acting hydraulic cylinder shown in the previous illustration can be described with a simple set of equations:

$$F_3 = p_1 A_1 - p_2 A_2$$

$$q_1 = A_1 v_3$$

$$q_2 = A_2 v_3$$

where

q_1, q_2	Flow rates through ports A and B, respectively (Through variables)
p_1, p_2	Gauge pressures at ports A and B, respectively (Across variables)
A_1, A_2	Piston effective areas
F_3	Rod force (Through variable)
v_3	Rod velocity (Across variable)

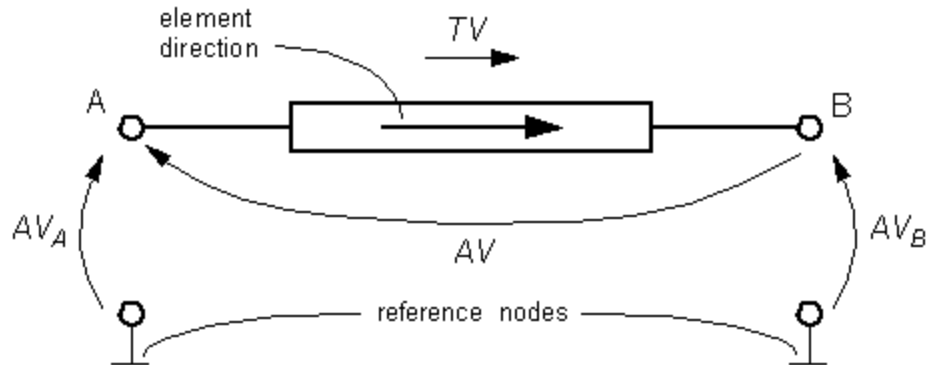
The model could be considerably more complex, for example, it could account for friction, fluid compressibility, inertia of the moving parts, and so on. For all these different mathematical models, however, the element configuration (that is, the number and type of ports and the associated Through and Across variables) would remain the same, meaning that the Physical Network approach lets you substitute models of different levels of complexity without introducing any changes to the schematic. For example, you can start developing your system by using the Resistive Tube block from the Foundation library, which accounts only for friction losses. At a later stage in development, you may want to account for fluid compressibility. You can then replace it with a Hydraulic Pipeline block, available with SimHydraulics® block libraries, or, depending on your application, even with a Segmented Pipeline block if you also need to account for fluid inertia. This modeling principle is called incremental modeling.

Direction of Variables

Each variable is characterized by its magnitude and sign. The sign is the result of measurement orientation. The same variable can be positive or negative, depending on the polarity of a measurement gauge.

Elements with only two ports are characterized with one pair of variables, a Through variable and an Across variable. Since these variables are closely related, their orientation is defined with one direction. For example, if an element is oriented from port A to port B, it implies that the Through variable (*TV*) is positive if it “flows” from A to B, and the Across variable is determined

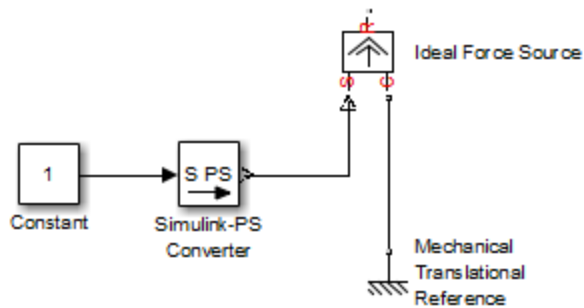
as $AV = AV_A - AV_B$, where AV_A and AV_B are the element node potentials or, in other words, the values of this Across variable at ports A and B, respectively.



This approach to the direction of variables has the following benefits:

- Provides a simple and consistent way to determine whether an element is active or passive. Energy is one of the most important characteristics to be determined during simulation. If the variables direction, or sign, is determined as described above, their product (that is, the energy) is positive if the element consumes energy, and is negative if it provides energy to a system. This rule is followed throughout the Simscape software.
- Simplifies the model description. Symbol A B is enough to specify variable polarity for both the Across and the Through variables.
- Lets you apply the oriented graph theory to network analysis and design.

As an example of variables direction rules, let us consider the Ideal Force Source block. In this block, as in many other mechanical blocks, port C is associated with the source reference point (case), and port R is associated with the rod.



The block positive direction is from port C to port R. This means that the force is positive if it acts in the direction from C to R, and causes bodies connected to port R to accelerate in the positive direction. The relative velocity is determined as $v = v_C - v_R$, where v_R , v_C are the absolute velocities at ports R and C, respectively, and it is negative if velocity at port R is greater than that at port C. The power generated by the source is computed as the product of force and velocity, and is negative if the source provides energy to the system.

Definition of positive direction is different for different blocks. Check the block source or the block reference page if in doubt about the block orientation and direction of variables.

All the elements in a network are divided into active and passive elements, depending on whether they deliver energy to the system or dissipate (or store) it. Active elements (force and velocity sources, flow rate and pressure sources, etc.) must be oriented strictly in accordance with the line of action or function that they are expected to perform in the system, while passive elements (dampers, resistors, springs, pipelines, etc.) can be oriented either way.

Connector Ports and Connection Lines


Simscape blocks may have the following types of ports:

- Physical Conserving ports — Nondirectional ports (for example, hydraulic or mechanical) that represent physical connections and relate physical variables based on the Physical Network approach.

- Physical Signal ports — Unidirectional ports transferring signals that use an internal Simscape engine for computations.

Each of these ports and connections between them are described in greater detail below.

Physical Conserving Ports

Simscape blocks have special Conserving ports . You connect Conserving ports with Physical connection lines, distinct from normal Simulink lines. Physical connection lines have no inherent directionality and represent the exchange of energy flows, according to the Physical Network approach.

- You can connect Conserving ports only to other Conserving ports of the same type.
- The Physical connection lines that connect Conserving ports together are nondirectional lines that carry physical variables (Across and Through variables, as described above) rather than signals. You cannot connect Physical lines to Simulink ports or to Physical Signal ports.
- Two directly connected Conserving ports must have the same values for all their Across variables (such as pressure or angular velocity).
- You can branch Physical connection lines. When you do so, components directly connected with one another continue to share the same Across variables. Any Through variable (such as flow rate or torque) transferred along the Physical connection line is divided among the multiple components connected by the branches. How the Through variable is divided is determined by the system dynamics.

For each Through variable, the sum of all its values flowing into a branch point equals the sum of all its values flowing out.

Each type of Physical Conserving ports used in Simscape blocks uniquely represents a physical modeling domain. For a list of port types, along with the Through and Across variables associated with each type, see the table in “Variable Types” on page 1-4.

Physical Signal Ports

Physical Signal ports **▷** carry signals between Simscape blocks. You connect them with regular connection lines, similar to Simulink signal connections. Physical Signal ports are used in Simscape block diagrams instead of Simulink input and output ports to increase computation speed and avoid issues with algebraic loops. Unlike Simulink signals, which are essentially unitless, physical signals can have units associated with them. You specify the units along with the parameter values in the block dialogs, and Simscape software performs the necessary unit conversion operations when solving a physical network.

Simscape Foundation library contains, among other sublibraries, a Physical Signals block library. These blocks perform math operations and other functions on physical signals, and allow you to graphically implement equations inside the Physical Network.

Connecting Simscape Diagrams to Simulink Sources and Scopes

Simscape block diagrams use physical signals instead of regular Simulink signals. Therefore, you need converter blocks to connect Simscape diagrams to Simulink sources and scopes.

Use the Simulink-PS Converter block to connect Simulink sources or other Simulink blocks to the inputs of a Physical Network diagram. You can also use it to specify the input signal units. For more information, see the Simulink-PS Converter block reference page.

Use the PS-Simulink Converter block to connect outputs of a Physical Network diagram to Simulink scopes or other Simulink blocks. You can also use it to specify the desired output signal units. For more information, see the PS-Simulink Converter block reference page.

For an example of using converter blocks to connect Simscape diagrams to Simulink sources and scopes, see “Creating and Simulating a Simple Model” on page 1-18.

Simscape Block Libraries

In this section...
“Library Structure Overview” on page 1-11
“Using the Simulink Library Browser to Access the Block Libraries” on page 1-13
“Using the Command Prompt to Access the Block Libraries” on page 1-14

Library Structure Overview

Simscape block library contains two libraries that belong to the Simscape product:

- Foundation library — Contains basic hydraulic, pneumatic, mechanical, electrical, magnetic, thermal, thermal liquid, and physical signal blocks, organized into sublibraries according to technical discipline and function performed
- Utilities library — Contains essential environment blocks for creating Physical Networks models

In addition, if you have installed any of the add-on products of the Physical Modeling family, you will see the corresponding libraries under the main Simscape library.

Simscape Foundation libraries contain a comprehensive set of basic elements and building blocks, such as:

- Mechanical building blocks for representing one-dimensional translational and rotational motion
- Electrical building blocks for representing electrical components and circuits
- Magnetic building blocks that represent electromagnetic components
- Hydraulic building blocks that model fundamental hydraulic effects and can be combined to create more complex hydraulic components

- Pneumatic building blocks that model fundamental pneumatic effects based on the ideal gas law
- Thermal building blocks that model fundamental thermal effects
- Thermal liquid building blocks that model fundamental thermodynamic effects in liquids
- Physical Signals block library that lets you perform math operations on physical signals, and graphically enter equations inside the physical network

Using the elements contained in these Foundation libraries, you can create more complex components that span different physical domains. You can then group this assembly of blocks into a subsystem and parameterize it to reuse and share these components.

In addition to Foundation libraries, there is also a Simscape Utilities library, which contains utility blocks, such as:

- Solver Configuration block, which contains parameters relevant to numerical algorithms for Simscape simulations. Each Simscape diagram (or each topologically distinct physical network in a diagram) must contain a Solver Configuration block.
- Simulink-PS Converter block and PS-Simulink Converter block, to connect Simscape and Simulink blocks. Use the Simulink-PS Converter block to connect Simulink outputs to Physical Signal inports. Use the PS-Simulink Converter block to connect Physical Signal outputs to Simulink inports.

For examples of using these blocks in a Simscape model, see the tutorial “Creating and Simulating a Simple Model” on page 1-18.

You can combine all these blocks in your Simscape diagrams to model physical systems. You can also use the basic Simulink blocks in your diagrams, such as sources or scopes. See “Connecting Simscape Diagrams to Simulink Sources and Scopes” on page 1-10 for more information on how to do this.

Simscape block libraries contain a comprehensive selection of blocks that represent engineering components such as valves, resistors, springs, and so on. These prebuilt blocks, however, may not be sufficient to address your particular engineering needs. When you need to extend the existing block

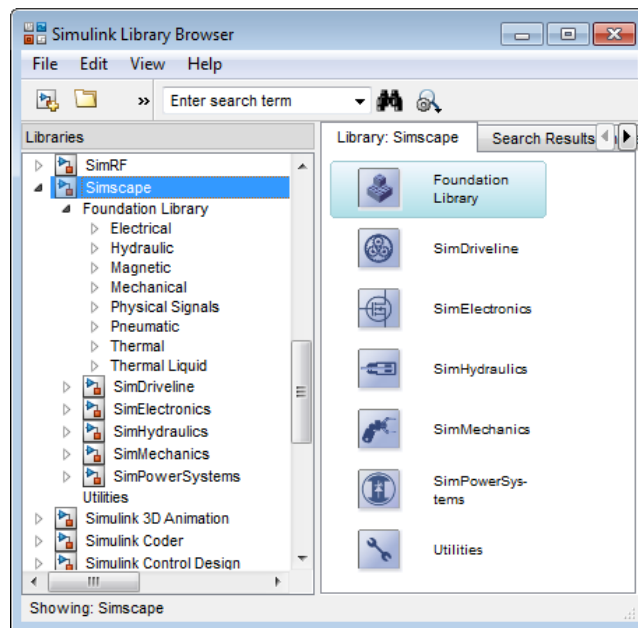
libraries, use the Simscape language to define customized components, or even new physical domains, as textual files. Then convert your textual components into libraries of additional Simscape blocks that you can use in your model diagrams. For more information on how to do this, see “Typical Simscape Language Tasks”.

Using the Simulink Library Browser to Access the Block Libraries

You can access the blocks through the Simulink Library Browser. To display the Library Browser, click the **Simulink Library** button in the toolbar of the MATLAB® desktop or Simulink model window:



Alternatively, you can type `simulink` in the MATLAB Command Window. Then expand the **Simscape** entry in the contents tree.

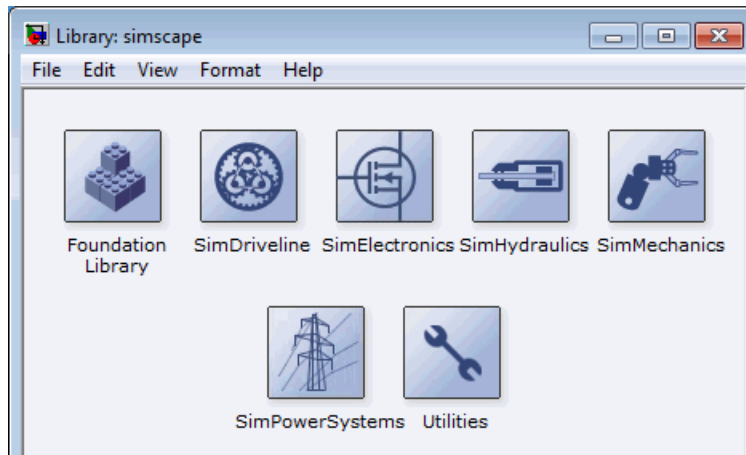


Using the Command Prompt to Access the Block Libraries

To access individual block libraries by using the command prompt:

- To open the Simscape library, type `simscape` in the MATLAB Command Window.
- To open the main Simulink library (to access generic Simulink blocks), type `simulink` in the MATLAB Command Window.

The Simscape library consists of two top-level libraries, Foundation and Utilities. In addition, if you have installed any of the add-on products of the Physical Modeling family, you will see the corresponding libraries under Simscape library, as shown in the following illustration. Some of these libraries contain second-level and third-level sublibraries. You can expand each library by double-clicking its icon.





Essential Physical Modeling Techniques

Building Your Model

The rules that you must follow when building a physical model with Simscape software are described in “Basic Principles of Modeling Physical Networks” on page 1-2. This section briefly reviews these rules.

- Build your physical model by using a combination of blocks from the Simscape Foundation and Utilities libraries. Simscape software lets you create a network representation of the system under design, based on the Physical Network approach. According to this approach, each system is represented as consisting of functional elements that interact with each other by exchanging energy through their ports.
- Each Simscape diagram (or each topologically distinct physical network in a diagram) must contain a Solver Configuration block from the Simscape Utilities library.
- If you have hydraulic elements in your model, the working fluid used in the hydraulic circuit defines their global parameters, such as fluid density, fluid kinematic viscosity, fluid bulk modulus, and so on. To specify the working fluid, attach a Custom Hydraulic Fluid block (or a Hydraulic Fluid block, available with SimHydraulics block libraries) to each topologically distinct hydraulic circuit. If no Hydraulic Fluid block or Custom Hydraulic Fluid block is attached to a circuit, the hydraulic blocks use the default fluid, which is equivalent to fluid defined by a Custom Hydraulic Fluid block with the default parameter values.
- If you have pneumatic elements in your model, default gas properties are for dry air and ambient conditions of 101325 Pa and 20 degrees Celsius. Attach a Gas Properties block to each topologically distinct pneumatic circuit to change gas properties and ambient conditions.
- To connect regular Simulink blocks (such as sources or scopes) to your physical network diagram, use the converter blocks, as described in “Using the Physical Signal Ports” on page 1-17.
- Use the incremental modeling approach. Start with a simple model, run and troubleshoot it, then add the desired special effects. For example, you can start developing your system by using the Resistive Tube block from the Foundation library, which accounts only for friction losses. At a later

stage in development, you may want to account for fluid compressibility. You can then replace it with a Hydraulic Pipeline block, available with SimHydraulics block libraries, or, depending on your application, even with a Segmented Pipeline block if you also need to account for fluid inertia. For all these different mathematical models, the element configuration (that is, the number and type of ports and the associated Through and Across variables) would remain the same, meaning that the Physical Network approach lets you substitute models of different levels of complexity without introducing any changes to the schematic.

Simscape blocks, in general, feature both Conserving ports  and Physical Signal inports and outputs .

Using the Conserving Ports

The following rules apply to Conserving ports:

- There are different types of Physical Conserving ports used in Simscape block diagrams, such as hydraulic, pneumatic, electrical, magnetic, thermal, mechanical translational, and mechanical rotational. Each type has specific Through and Across variables associated with it. For more information, see “Variable Types” on page 1-4.
- You can connect Conserving ports only to other Conserving ports of the same type.
- The Physical connection lines that connect Conserving ports together are nondirectional lines that carry physical variables (Across and Through variables, as described above) rather than signals. You cannot connect Physical lines to Simulink ports or to Physical Signal ports.
- Two directly connected Conserving ports must have the same values for all their Across variables (such as voltage or angular velocity).
- You can branch Physical connection lines. When you do so, components directly connected with one another continue to share the same Across variables. Any Through variable (such as current or torque) transferred along the Physical connection line is divided among the multiple components connected by the branches. How the Through variable is divided is determined by the system dynamics.

For each Through variable, the sum of all its values flowing into a branch point equals the sum of all its values flowing out.

Using the Physical Signal Ports

The following rules apply to Physical Signal ports:

- You can connect Physical Signal ports to other Physical Signal ports with regular connection lines, similar to Simulink signal connections. These connection lines carry physical signals between Simscape blocks.
- You can connect Physical Signal ports to Simulink ports through special converter blocks. Use the Simulink-PS Converter block to connect Simulink outputs to Physical Signal inports. Use the PS-Simulink Converter block to connect Physical Signal outputs to Simulink inports.
- Unlike Simulink signals, which are essentially unitless, Physical Signals can have units associated with them. Simscape block dialogs let you specify the units along with the parameter values, where appropriate. Use the converter blocks to associate units with an input signal and to specify the desired output signal units.

For examples of applying these rules when creating an actual physical model, see the tutorial “Creating and Simulating a Simple Model” on page 1-18.

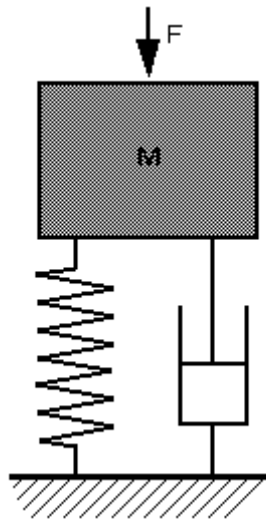
Creating and Simulating a Simple Model

In this section...
“Building a Simscape Diagram” on page 1-18
“Modifying Initial Settings” on page 1-26
“Running the Simulation” on page 1-27
“Adjusting the Parameters” on page 1-30

Building a Simscape Diagram

In this example, you are going to model a simple mechanical system and observe its behavior under various conditions. This tutorial illustrates the essential steps to building a physical model and makes you familiar with using the basic Simscape blocks.

The following schematic represents a simple model of a car suspension. It consists of a spring and damper connected to a body (represented as a mass), which is agitated by a force. You can vary the model parameters, such as the stiffness of the spring, the mass of the body, or the force profile, and view the resulting changes to the velocity and position of the body.



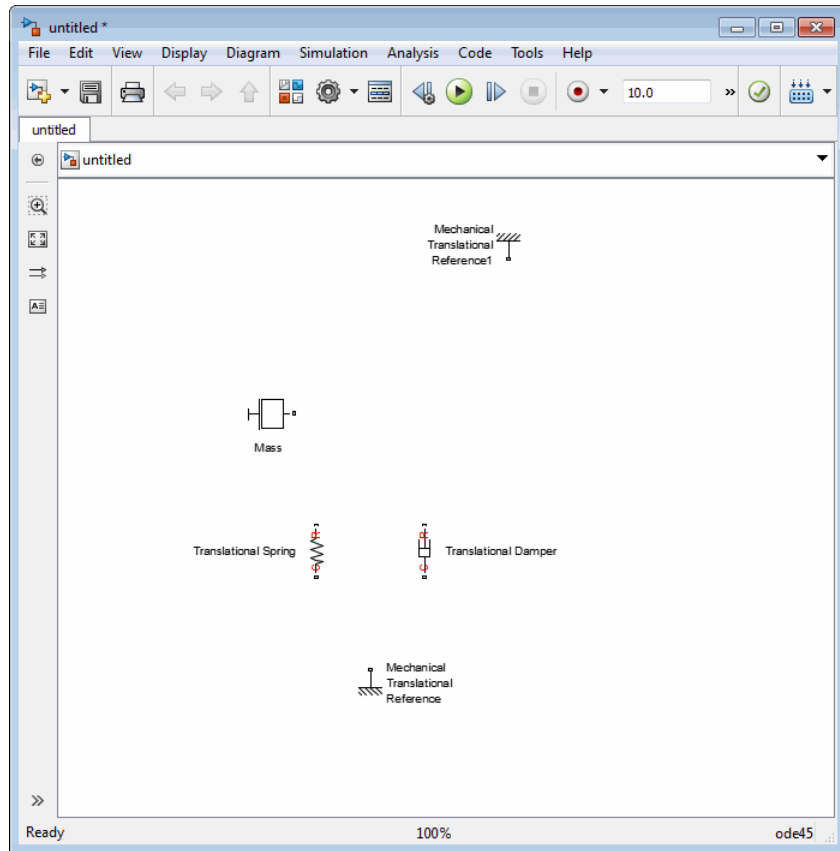
To create an equivalent Simscape diagram, follow these steps:

- 1** Open the Simulink Library Browser, as described in “Simscape Block Libraries” on page 1-11.
- 2** Create a new model. To do this, from the top menu bar of the Library Browser, select **File > New > Model**. The software creates an empty model in memory and displays it in a new model editor window.

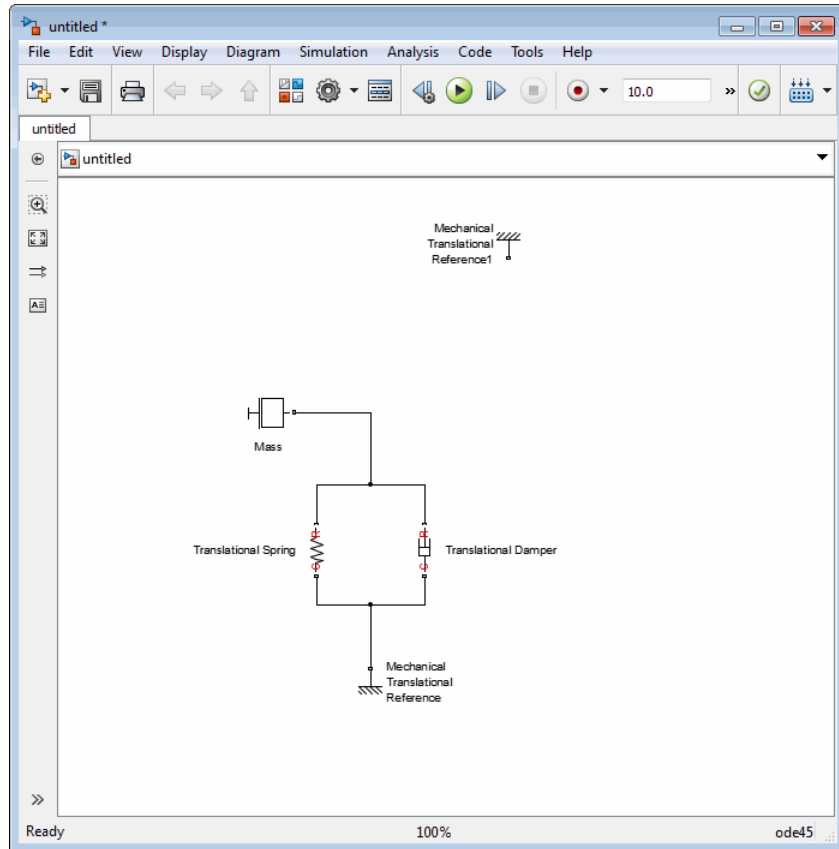
Note Alternately, you can type `ssc_new` at the MATLAB Command prompt, to create a new model prepopulated with certain required and commonly-used blocks. For more information, see “Creating a New Simscape Model”.

- 3** Open the Simscape > Foundation Library > Mechanical > Translational Elements library.
- 4** Drag the Mass, Translational Spring, Translational Damper, and two Mechanical Translational Reference blocks into the model window.

- 5 Orient the blocks as shown in the following illustration. To rotate a block, select it and press **Ctrl+R**.

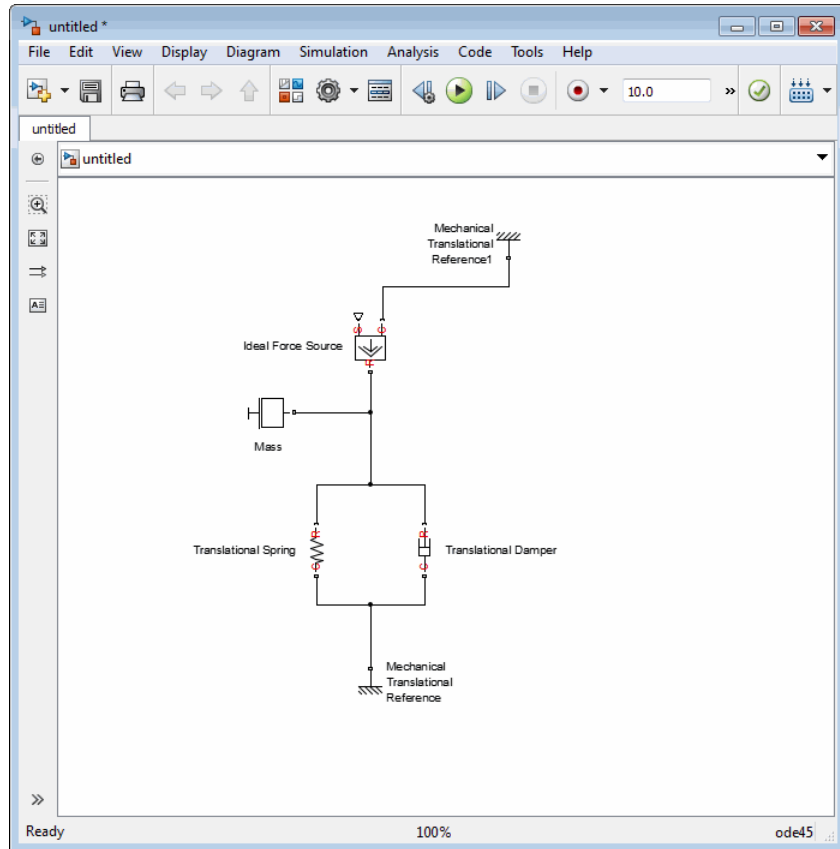


- 6 Connect the Translational Spring, Translational Damper, and Mass blocks to one of the Mechanical Translational Reference blocks as shown in the next illustration.

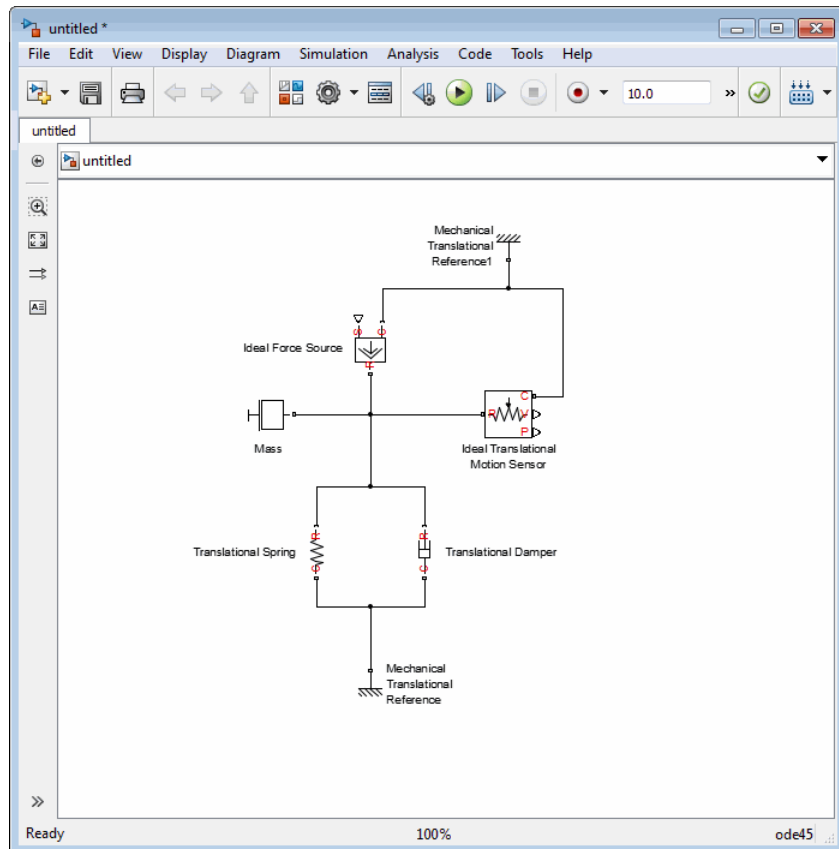


- 7 To add the representation of the force acting on the mass, open the Simscape > Foundation Library > Mechanical > Mechanical Sources library and add the Ideal Force Source block to your diagram.

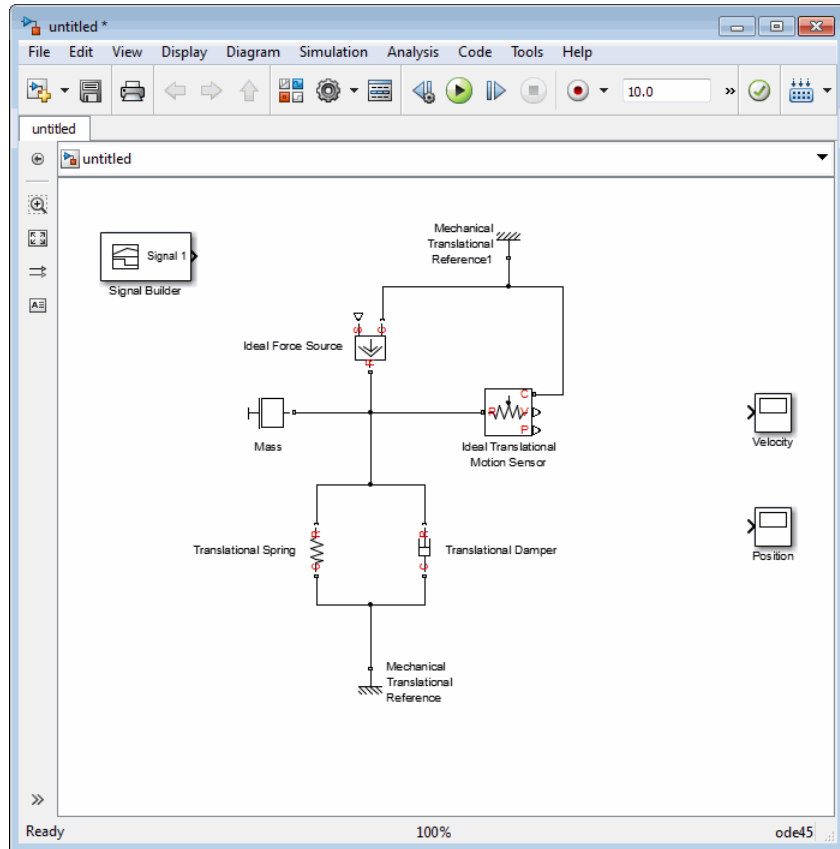
To reflect the correct direction of the force shown in the original schematic, flip the block by selecting **Diagram > Rotate & Flip > Flip Block > Up-Down** from the top menu bar of the model window. Connect the block's port C (for “case”) to the second Mechanical Translational Reference block, and its port R (for “rod”) to the Mass block, as shown below.



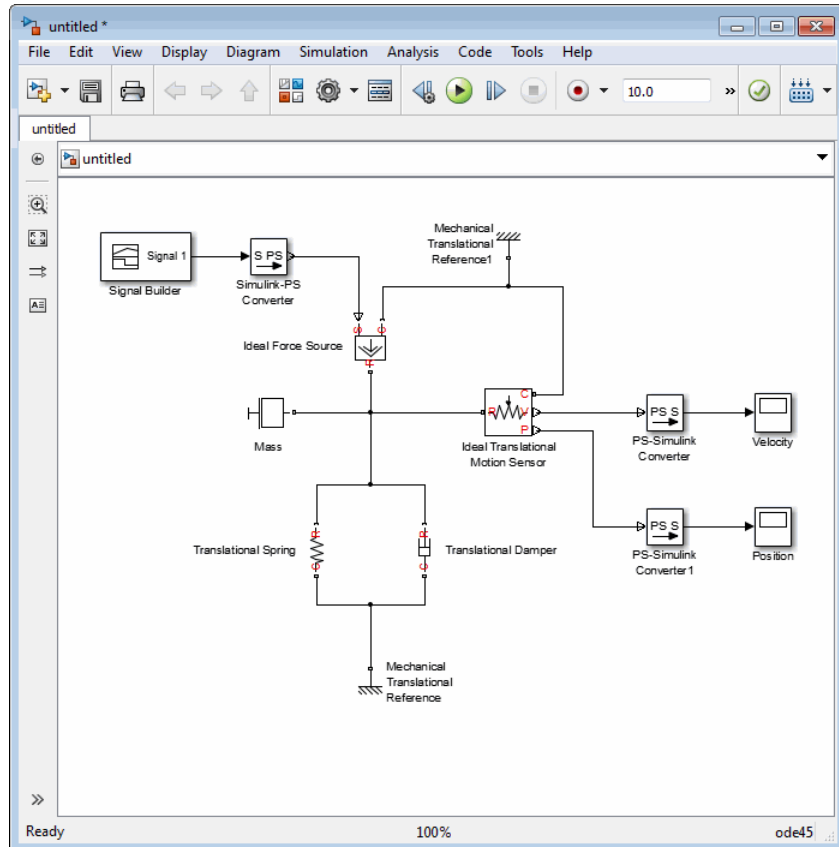
- 8 Add the sensor to measure speed and position of the mass. Place the Ideal Translational Motion Sensor block from the Mechanical Sensors library into your diagram and connect it as shown below.



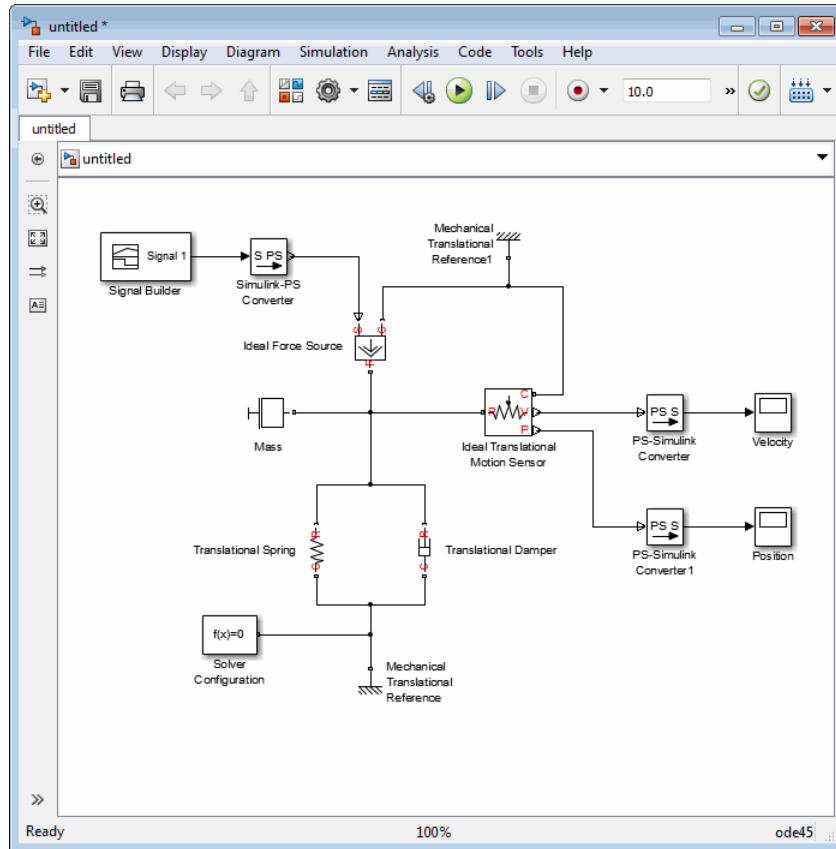
- Now you need to add the sources and scopes. They are found in the regular Simulink libraries. Open the Simulink > Sources library and copy the Signal Builder block into the model. Then open the Simulink > Sinks library and copy two Scope blocks. Rename one of the Scope blocks to **Velocity** and the other to **Position**.



- 10 Every time you connect a Simulink source or scope to a Simscape diagram, you have to use an appropriate converter block, to convert Simulink signals into physical signals and vice versa. Open the Simscape > Utilities library and copy a Simulink-PS Converter block and two PS-Simulink Converter blocks into the model. Connect the blocks as shown below.



- 11** Each topologically distinct physical network in a diagram requires exactly one Solver Configuration block, found in the Simscape > Utilities library. Copy this block into your model and connect it to the circuit by creating a branching point and connecting it to the only port of the Solver Configuration block. Your diagram now should look like this.



12 Your block diagram is now complete. Save it as `mech_simple`.

Modifying Initial Settings

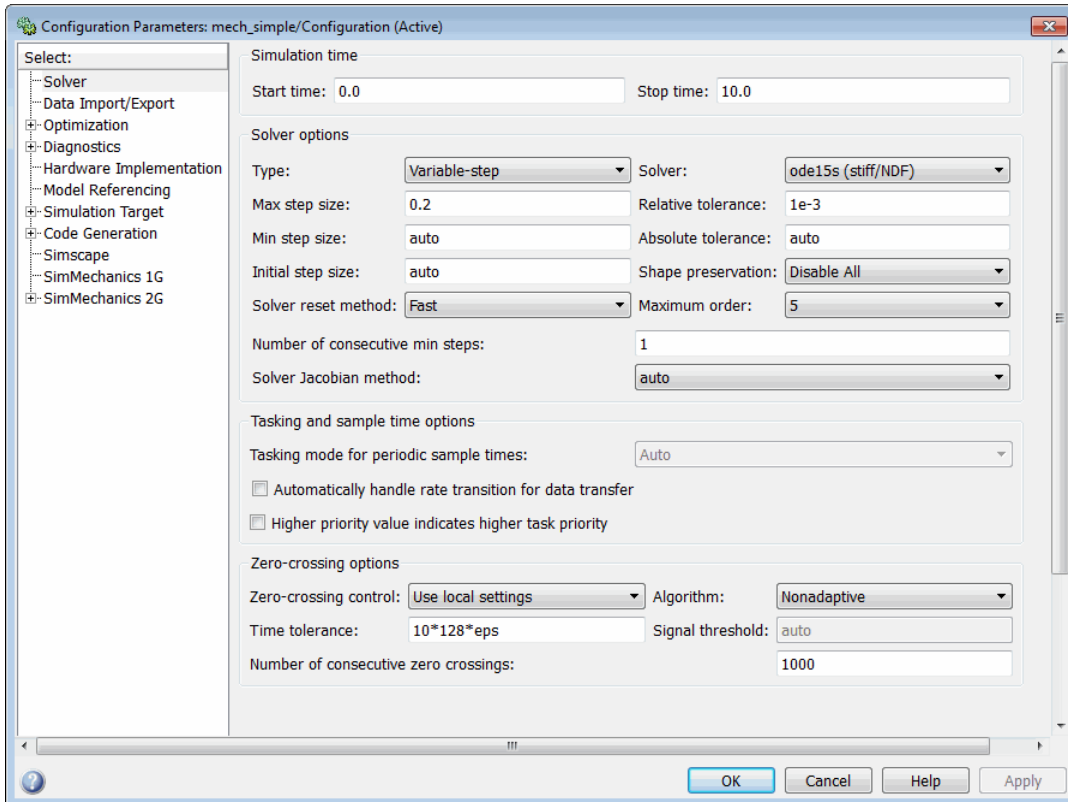
After you have put together a block diagram of your model, as described in the previous section, you need to select a solver and provide the correct values for configuration parameters.

To prepare for simulating the model, follow these steps:

- 1 Select a Simulink solver. On the top menu bar of the model window, select **Simulation > Model Configuration Parameters**. The Configuration Parameters dialog box opens, showing the **Solver** node.

Under **Solver options**, set **Solver** to `ode15s` (Stiff/NDF) and **Max step size** to `0.2`.

Also note that **Simulation time** is specified to be between 0 and 10 seconds. You can adjust this setting later, if needed.



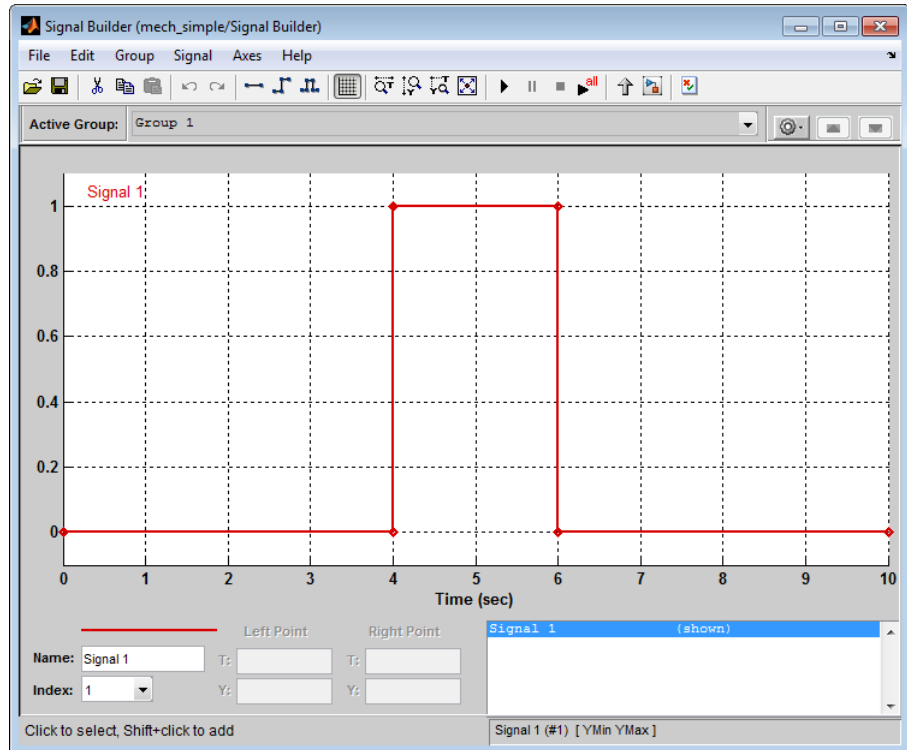
Click **OK** to close the Configuration Parameters dialog box.

2 Save the model.


Running the Simulation

After you've put together a block diagram and specified the initial settings for your model, you can run the simulation.

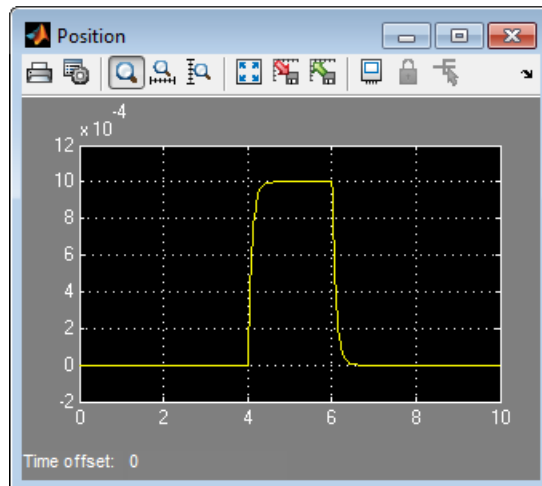
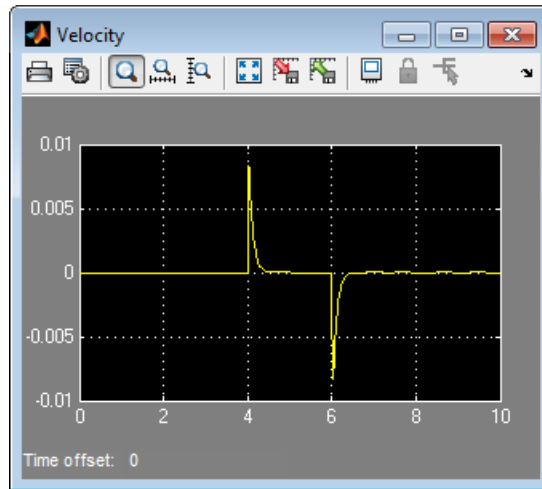
- 1 The input signal for the force is provided by the Signal Builder block. The signal profile is shown in the illustration below. It starts with a value of 0, then at 4 seconds there is a step change to 1, and then it changes back to 0 at 6 seconds. This is the default profile.



The Velocity scope outputs the mass velocity, and the Position scope outputs the mass displacement as a function of time. Double-click both scopes to open them.

- 2 To run the simulation, click  in the model window toolbar. The Simscape solver evaluates the model, calculates the initial conditions, and runs the simulation. For a detailed description of this process, see “How Simscape Simulation Works” on page 3-5. Completion of this step may take a few seconds. The message in the bottom-left corner of the model window provides the status update.

- 3 Once the simulation starts running, the Velocity and Position scope windows display the simulation results, as shown in the next illustration.



In the beginning, the mass is at rest. Then at 4 seconds, as the input signal changes abruptly, the mass velocity spikes in the positive direction and gradually returns to zero. The mass position at the same time changes more gradually, on account of inertia and damping, and stays at the new

value as long as the force is acting upon it. At 6 seconds, when the input signal changes back to zero, the velocity gets a mirror spike, and the mass gradually returns to its initial position.

You can now adjust various inputs and block parameters and see their effect on the mass velocity and displacement.

Adjusting the Parameters

After running the initial simulation, you can experiment with adjusting various inputs and block parameters.

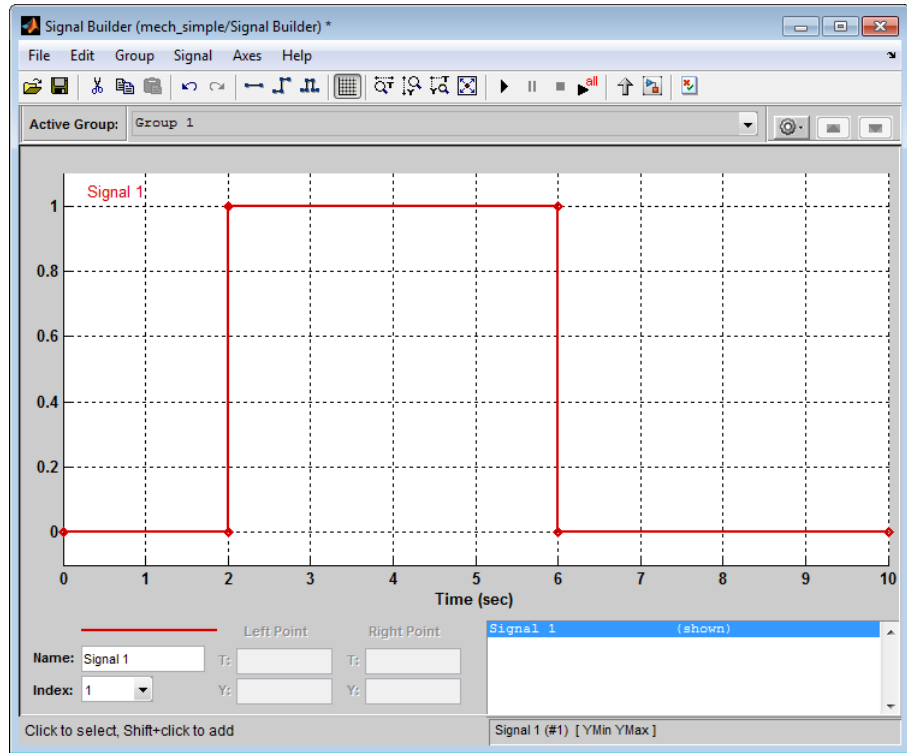
Try the following adjustments:

- 1** Change the force profile.
- 2** Change the model parameters.
- 3** Change the mass position output units.

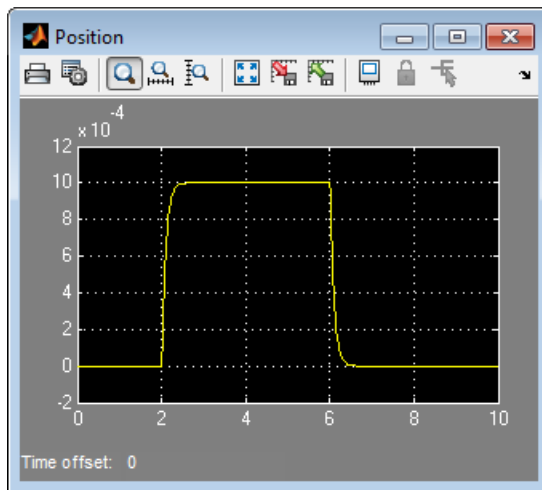
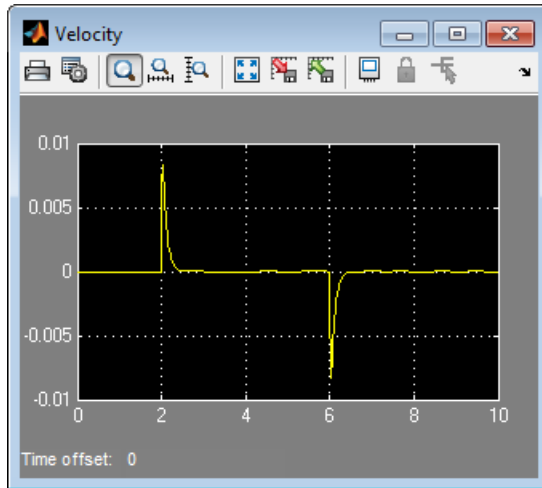
Changing the Force Profile

This example shows how a change in the input signal affects the force profile, and therefore the mass displacement.

- 1** Double-click the Signal Builder block to open it.
- 2** Click the first vertical segment of the signal profile and drag it from 4 to 2 seconds, as shown below. Close the block dialog.



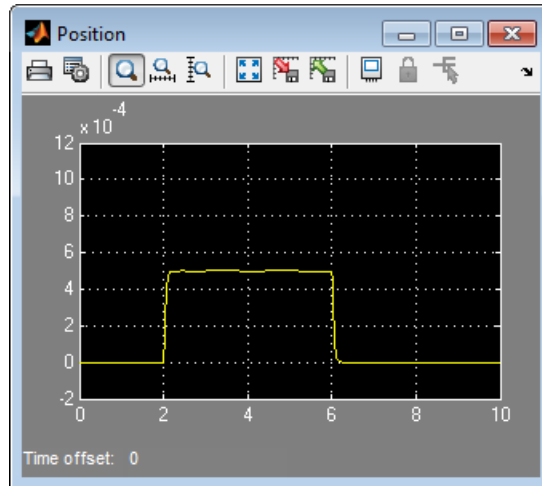
3 Run the simulation. The simulation results are shown in the following illustration.



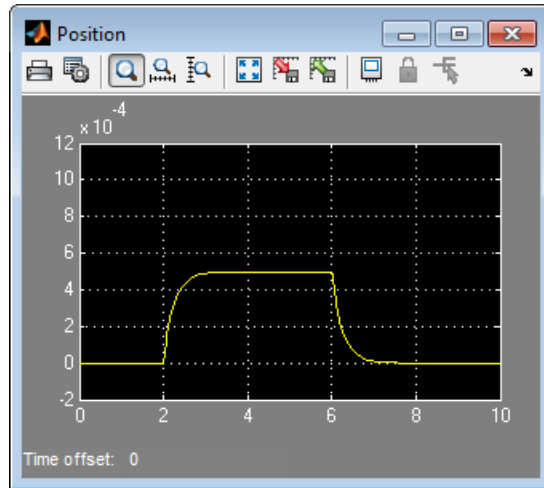
Changing the Model Parameters

In our model, the force acts on a mass against a translational spring and damper, connected in parallel. This example shows how changes in the spring stiffness and damper viscosity affect the mass displacement.

- 1 Double-click the Translational Spring block. Set its **Spring rate** to 2000 N/m.
- 2 Run the simulation. The increase in spring stiffness results in smaller amplitude of mass displacement, as shown in the following illustration.




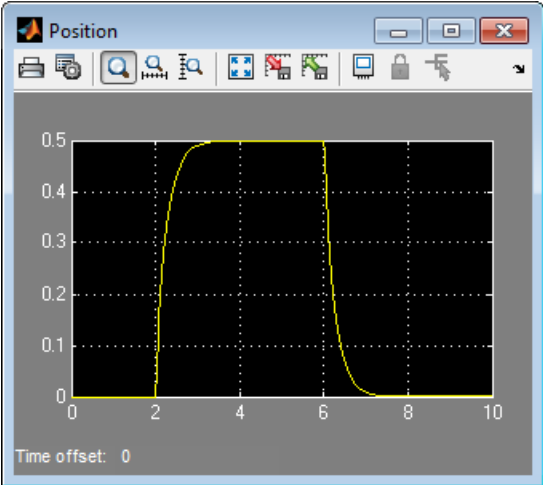
- 3 Next, double-click the Translational Damper block. Set its **Damping coefficient** to 500 N/(m/s).
- 4 Run the simulation. Because of the increase in viscosity, the mass is slower both in reaching its maximum displacement and in returning to the initial position, as shown in the following illustration.



Changing the Mass Position Output Units

In our model, we have used the PS-Simulink Converter block in its default parameter configuration, which does not specify units. Therefore, the Position scope outputs the mass displacement in the default length units, that is, in meters. This example shows how to change the output units for the mass displacement to millimeters.

- 1 Double-click the PS-Simulink Converter block. Type mm in the **Output signal unit** combo box and click **OK**.
- 2 Run the simulation. In the Position scope window, click  to autoscale the scope axes. The mass displacement is now output in millimeters, as shown in the following illustration.



Modeling Best Practices

In this section...
“Grounding Rules” on page 1-36
“Avoiding Numerical Simulation Issues” on page 1-40

Grounding Rules

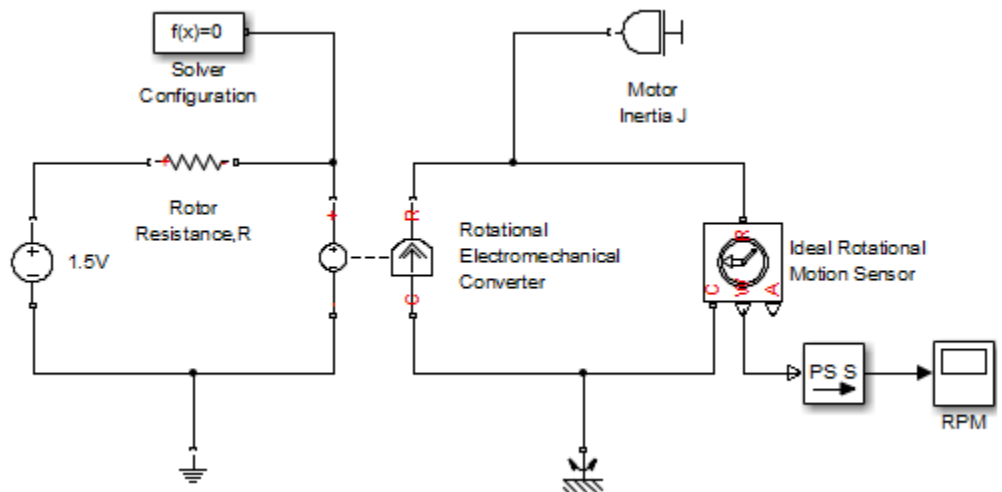
This section contains guidelines for using domain-specific reference blocks (such as Electrical Reference, Mechanical Translational Reference, and so on) in Simscape diagrams, along with examples of correct and incorrect configurations.

Add reference blocks to your models according to the following rules:

- “Each Domain Requires at Least One Reference Block” on page 1-36
- “Each Circuit Requires at Least One Reference Block” on page 1-37
- “Multiple Connections to the Domain Reference Are Allowed Within a Circuit” on page 1-39

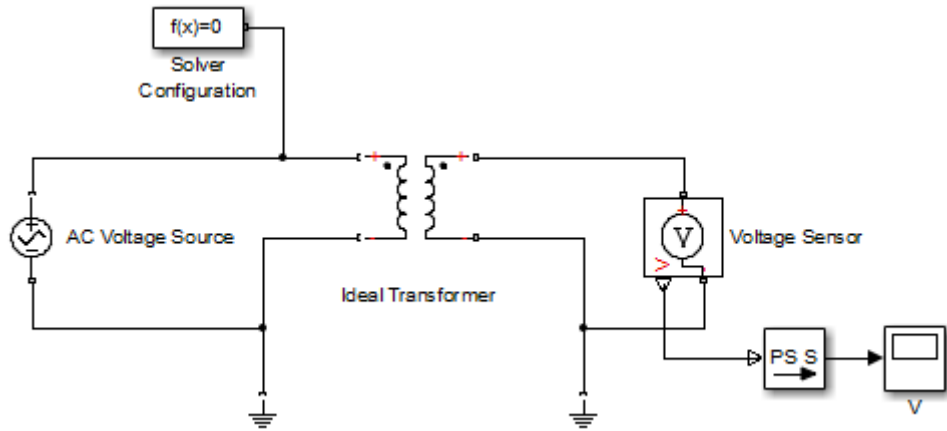
Each Domain Requires at Least One Reference Block

Within a physical network, each domain must contain at least one reference block of the appropriate type. For example, the electromechanical model shown in the following diagram has both Electrical Reference and Rotational Reference blocks attached to the appropriate circuits.

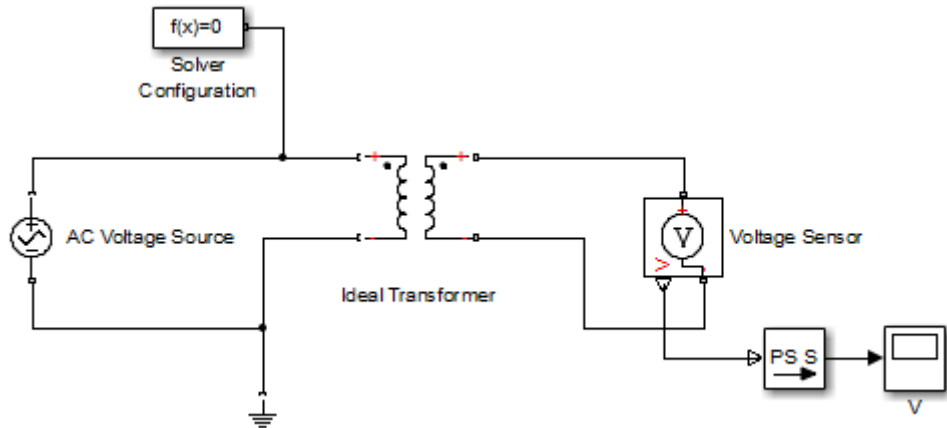


Each Circuit Requires at Least One Reference Block

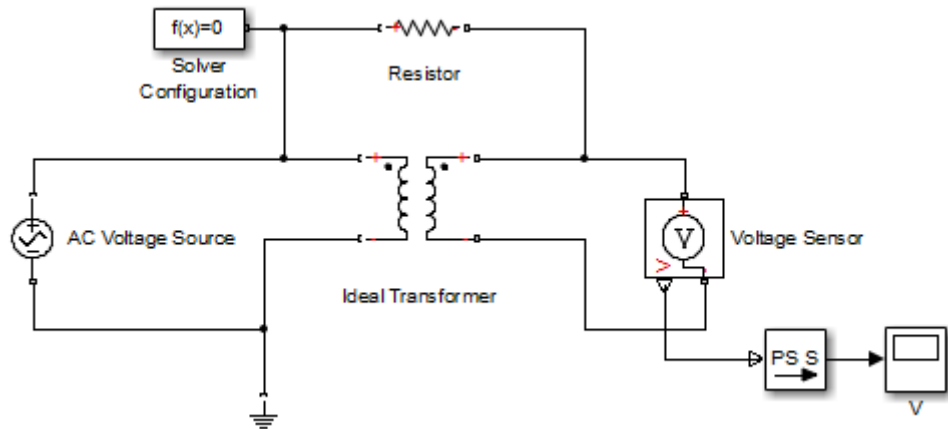
Each topologically distinct circuit within a domain must contain at least one reference block. Some blocks, such as an Ideal Transformer, interface two parts of the network but do not convey information about signal levels relative to the reference block. In the following diagram, there are two separate electrical circuits, and the Electrical Reference blocks are required on both sides of the Ideal Transformer block.



The next diagram would produce an error because it is lacking an electrical reference in the circuit of the secondary winding.



The following diagram, however, will not produce an error because the resistor defines the output voltage relative to the ground reference.

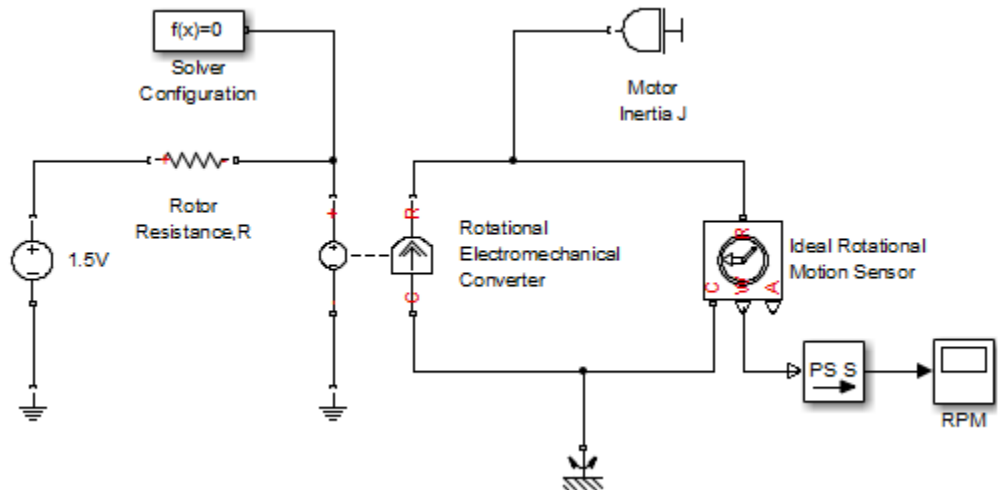


Multiple Connections to the Domain Reference Are Allowed Within a Circuit

More than one reference block may be used within a circuit to define multiple connections to the domain reference:

- Electrical conserving ports of all the blocks that are directly connected to ground must be connected to an Electrical Reference block.
- All translational ports that are rigidly clamped to the frame (ground) must be connected to a Mechanical Translational Reference block.
- All rotational ports that are rigidly clamped to the frame (ground) must be connected to a Mechanical Rotational Reference block.
- Hydraulic conserving ports of all the blocks that are referenced to atmosphere (for example, suction ports of hydraulic pumps, or return ports of valves, cylinders, pipelines, if they are considered directly connected to atmosphere) must be connected to a Hydraulic Reference block.

For example, the following diagram correctly indicates two separate connections to an electrical ground.



Avoiding Numerical Simulation Issues

Certain configurations of physical modeling blocks can cause numerical difficulties or slow down your simulation. When this happens, Simscape solver issues a warning in the MATLAB workspace and, if it fails to initialize, a Simscape error.

In electrical circuits, common examples that can cause this behavior include voltage sources connected in parallel with capacitors, inductors connected in series with current sources, voltage sources connected in parallel, and current sources connected in series. Often, the cause of the numerical difficulty is immediately apparent. For example, two voltage sources in parallel must have identical voltage values; otherwise, the ports connecting them would not be physical conserving ports. In practical circuits, topologies such as parallel voltage sources are possible, and small difference in their instantaneous voltages is possible due to parasitic series resistance.

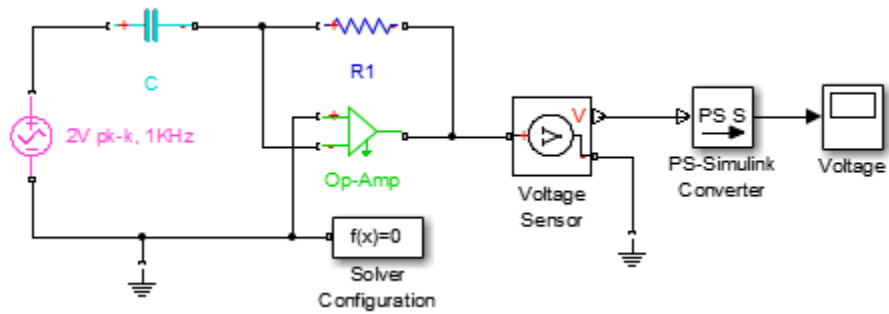
Note Mathematically, these topologies result in *Index-2 differential algebraic equations* (DAEs). Their solution requires two differentiations of the constraint equations and, as such, it is numerically better to avoid these component topologies where possible.

There are two approaches to resolving these difficulties. The first is to change the circuit to an equivalent simpler one. In the example of two parallel voltage sources, one source can be simply deleted. The same applies to two series current sources, the deleted one being replaced by a short circuit. For some circuit topologies, however, it is not possible to find an equivalent simpler one that resolves the problem, and the second approach is needed.

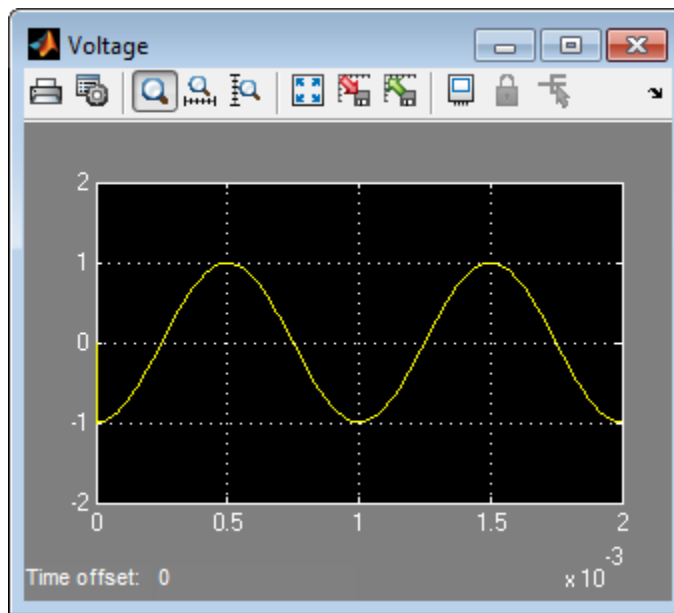
The second approach is to include small parasitic resistances in the component. In the Simscape Foundation library, the Capacitor and Inductor blocks include such parasitic terms, so that you can connect capacitances in parallel with voltage sources and inductors in series with current sources. If your circuit does not have any such topologies, then you can change the default parasitic terms to zero. Note that other blocks do not contain these parasitic terms, for example, the Mutual Inductor block. Therefore, if you wanted to connect a mutual inductor primary in series with a current source, you would need to introduce your own parasitic conductance across the primary winding.

Example of Using a Parasitic Resistance to Avoid Numerical Simulation Issues

The following diagram models a differentiator that might be used as part of a Proportional-Integral-Derivative (PID) controller. You can open this model by typing `ssc_differentiator` in the MATLAB Command Window.



Simulate the model, and you will see that the output is minus the derivative of the input sinusoid.



Now open the capacitor C block dialog, and set the series resistance to zero. The model now runs very slowly, and issues a warning:

Warning: problems possible for transient initialization, as well as stepsize control for transient solve, due to equations of one or more components:

```
'ssc_differentiator/2V pk-k, 1KHz'  
'ssc_differentiator/Op-Amp'  
'ssc_differentiator/C'
```

The cause of the warning is that the circuit effectively connects the voltage source in parallel with the capacitor. This is because an ideal op-amp satisfies $V+ = V-$, where $V+$ and $V-$ are the noninverting and inverting inputs, respectively. This is an example where it is not possible to replace the circuit with an equivalent simpler one, and a parasitic small resistance has to be introduced.

Modeling Pneumatic Systems

In this section...

- “Intended Applications” on page 1-44
- “Assumptions and Limitations” on page 1-44
- “Fundamental Equations” on page 1-45
- “Network Variables” on page 1-46
- “Connection Constraints” on page 1-47
- “References” on page 1-47

Intended Applications

The Foundation library contains basic pneumatic elements, such as orifices, chambers, and pneumatic-mechanical converters, as well as pneumatic sensors and sources. Use these blocks to model pneumatic systems, for applications such as:

- Factory automation — basic pneumatic linear/rotational actuators, valves (variable orifices), and air supply
- Robotics — robotic arms and haptic interfaces
- Gaseous transportation systems and pipelines

You can also use these blocks to model dry air and low-pressure flows, for example, for HVAC applications.

Assumptions and Limitations

Pneumatic block models are based on the following assumptions:

- Working fluid is an ideal gas satisfying the ideal gas law.
- Specific heats at constant pressure and constant volume, c_p and c_v , are constant.

- Processes are adiabatic, that is, there is no heat transfer between components and the environment (except for components with a separate thermal port).
- Gravitational effects can be neglected, that is, underlying equations contain no head pressures due to gravity.

Fundamental Equations

The energy balance for a control volume [1] is

$$\frac{dE_{cv}}{dt} = Q_{cv} - W_{cv} + \sum_i \left(m_i \left(h_i + \frac{v_i^2}{2} + gz_i \right) \right) - \sum_o \left(m_o \left(h_o + \frac{v_o^2}{2} + gz_o \right) \right)$$

where

E_{cv}	Control volume total energy
Q_{cv}	Heat energy per second added to the gas through the boundary
W_{cv}	Mechanical work per second performed by the gas
h_i, h_o	Inlet and outlet enthalpies
v_i, v_o	Gas inlet and outlet velocities
g	Acceleration due to gravity
z_i, z_o	Elevations at inlet and outlet ports
m_i, m_o	Mass flow rates in and out of the control volume

The equation is an accounting balance for the energy of the control volume. It states that the rate of energy increase or decrease within the control volume equals the difference between the rates of energy transfer in and out across the boundary. The mechanisms of energy transfer are heat and work, as for closed systems, and the energy that accompanies the mass entering and exiting.

Pneumatic block models make several simplifying assumptions, as described previously.

The ideal gas law relates pressure, density, and temperature:

$$p = \rho RT$$

where

p	Absolute pressure
ρ	Gas density
R	Specific gas constant
T	Absolute gas temperature

Also, the specific enthalpies for an ideal gas at temperature T and constant pressure and constant volume are given by:

$$h = c_p T$$

$$h = c_v T$$

The pneumatic components also use the mass continuity equation:

$$\frac{d\rho}{dt} = \frac{1}{V}(m_i - m_o)$$

where ρ is the density of the gas within the component. For components with no internal mass of gas, the equation simplifies to:

$$G = m_i = m_o$$

where G is the mass flow rate through the component.

For specific equations used in each block, see the block reference pages.

Network Variables

The Across variables are pressure and temperature, and the Through variables are mass flow rate and heat flow. Note that these choices result in

a pseudo-bond graph, because the product of pressure and mass flow rate is not power.

Connection Constraints

Every node in a pneumatic network must have a defined temperature as well as pressure. This rule places some constraints on how you connect the pneumatic elements. In effect, every node should have a volume of fluid associated with it. When the ideal gas law is applied, this volume of fluid determines the relationship between temperature and pressure. Some elements already have a volume of fluid associated with them, and therefore having just one of these components connected to a node satisfies this condition. Such blocks include Constant Volume Pneumatic Chamber, Pneumatic Piston Chamber, Rotary Pneumatic Piston Chamber, and Pneumatic Atmospheric Reference.

An exception to the above rule (that every node must have a volume of fluid associated with it) occurs when two nodes are connected by a component for which the heat equation says that the temperatures are equal. In this case, just one of the nodes needs to be connected to a component with associated volume of fluid. Such components include the pressure and flow rate sources.

For models that represent an actual pneumatic network, these constraints should have no impact. For example, connecting two orifices in series makes no physical sense because the underlying assumption of the orifice equation is that gas is discharged into a volume of fluid. Therefore, modeling actual physical systems should automatically satisfy these constraints.

References

[1] Moran M.J. and Shapiro H.N. *Fundamentals of Engineering Thermodynamics*. Second edition. New York: John Wiley & Sons, 1992.

Thermal Liquid Models

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Library” on page 2-7
- “Thermal Liquid Modeling Framework” on page 2-11
- “Heat Transfer in Insulated Oil Pipeline” on page 2-15

Modeling Thermal Liquid Systems

In this section...

“When to Use Thermal Liquid Blocks” on page 2-2

“Modeling Workflow” on page 2-3

“Establish Model Requirements” on page 2-3

“Model Physical Components” on page 2-4

“Prepare Model for Analysis” on page 2-5

“Run Simulation” on page 2-5

When to Use Thermal Liquid Blocks

The Thermal Liquid library expands the fluid modeling capability of Simscape. With this library, you can account for thermal effects in a fluid system. For example, you can model the warming effect of viscous dissipation in a pipe. You can also account for the temperature dependence of fluid properties, e.g., density and viscosity.

To decide whether Thermal Liquid blocks fit your modeling needs, consider the fluid system you are trying to represent. Other Simscape blocks—e.g. Hydraulic or Pneumatic—may better suit your application. Assess the following:

- Number of phases

Is the fluid medium single phase or multiphase?

- Relevant phases

Is the fluid medium a gas, a liquid, or a multiphase mixture?

- Thermal effects

Does temperature change significantly in the time scale of the simulation?

Are thermal effects important for analysis? Are the temperature dependences of the liquid properties important?

As a rule, use Thermal Liquid blocks for fluid systems in which a single-phase liquid experiences significant temperature changes. For gaseous systems,

use Pneumatic blocks instead. For isothermal liquid systems, use Hydraulic blocks.

Modeling Workflow

The suggested workflow for Thermal Liquid models includes four steps:

- 1** Establish model requirements — Define the purpose and scope of the model. Then, identify the relevant components and interactions in the model. Use this information as a guide when building the model.
- 2** Model physical components — Determine the appropriate blocks for modeling the relevant components and interactions. Then, add the blocks to the model canvas and connect them according to the Simscape connection rules. Specify the block parameters.
- 3** Prepare model for analysis — Add sensors to the model. Alternatively, configure the model for Simscape data logging. Check the physical units of each sensed variable.
- 4** Run simulation — Configure the solver settings. Then, run the simulation. If necessary, refine the model until you achieve the desired fidelity level.

Establish Model Requirements

The foundation of a good model is a clear understanding of its purpose and requirements. What are you trying to accomplish with the model? What are the relevant components, processes, and states? Determine what is essential and what is not. Start simple, using a rough approximation of the physical system as a guide. Then, iteratively add detail to reach the appropriate model fidelity for your application.

An insulated oil pipeline buried underground provides an example. As oil flows through the pipeline, it experiences conductive heat losses due to the cooler pipeline surroundings. Heat flows across three material layers—pipe wall, insulant, and soil—causing oil temperature to drop. However, only conduction across soil and insulant layers matter. A typical pipe wall is thin and conductive, and its effect on conductive heat loss is minimal at best. Omitting this process simplifies the model and speeds up simulation.

You also must determine the dimensions and properties of each component. During modeling, you specify these parameters in the Simscape blocks for the components. Obtain the physical properties of the liquid medium. Manufacturer data sheets typically provide this data. You can also use analytical expressions to define the physical property lookup tables.

When modeling pipes, consider the impact that dynamic compressibility and flow inertia have on the transient system behavior. If the time scale of an effect exceeds the simulation run time, the impact is usually negligible. During modeling, turn off negligible effects to improve simulation speed. Characteristic time scales for dynamic compressibility and flow inertia are approximately L/c and L/v , respectively, where:

- L is the length of the pipe.
- v is the mean flow velocity through the pipe.
- c is the speed of sound in the liquid medium.

If you are unsure whether an effect is relevant to your model, simulate the model with and without that effect. Then, compare the two simulation results. If the difference is substantial, leave that effect in place. The result is greater model fidelity at small time scales, e.g., during transients associated with flow reversal in a pipe.

Model Physical Components

Start by adding a Thermal Liquid Settings (TL) block to the model canvas. Use this block to provide the physical properties of the liquid medium. This block is not strictly required, but without it the liquid properties are reset to their default values, given for water. In the block dialog box, enter the physical property lookup tables that you acquired during the planning stage.

Identify the appropriate blocks for representing the physical components and their interactions. Components can be simple, requiring a single block, or custom, requiring multiple blocks typically within a Subsystem block. Add the blocks to the model canvas and connect them according to the Simscape connection rules.

The `ssc_tl_hydraulic_fluid_warming` example shows simple and custom components. The Mass Flow Rate Source (TL) represents an ideal power

source. It is a simple component. The Double-acting cylinder subsystem block represents the mechanical part of a hydraulic actuator. It contains two Translational Mechanical Converter (TL) blocks and is a custom component.

Once you have connected the blocks, specify the relevant parameters. These include dimensions, physical states, empirical correlation coefficients, and initial conditions. In Pipe (TL), Rotational Mechanical Converter (TL), and Translational Mechanical Converter (TL) blocks, select the appropriate setting for effects such as dynamic compressibility and flow inertia.

Note For accurate simulation results, always replace the default parameter values with data appropriate for your model.

Prepare Model for Analysis

To analyze a model, you must set up that model for data collection. The simplest approach is to add sensor blocks to the model. The Thermal Liquid library provides two sensor block types: one for Through variables (mass flow rate and heat flux), the other for Across variables (pressure and temperature). By using the PS-Simulink Converter block, you can specify the physical units of the sensed variable.

An alternative approach is to use Simscape data logging. This approach, which uses MATLAB commands instead of blocks, provides access to a broader range of model variables and parameters. One example is the kinematic viscosity of the liquid medium inside a pipeline segment. You can analyze this parameter using Simscape data logging but not sensor blocks.

To configure a model for Simscape data logging, see “How to Log Simulation Data” on page 4-3. For an example of how to plot logged data, see “Log and Plot Simulation Data” on page 4-7.

Run Simulation

The final step in the modeling workflow is to simulate the model. Before running simulation, check that the numerical solver is appropriate for your model. To do this, use the **Model Configuration Parameters** dialog box.

For physical models, variable-step solvers such as `ode15s` typically perform best. Reduce step sizes and tolerances for greater simulation accuracy. Increase them instead for faster simulation.

Run the simulation. Plot simulation data from sensors and Simscape data logging, or process it for further analysis. If necessary, refine the model. For example, correct simulation issues or to improve model fidelity.

Related Examples

- “Heat Transfer in Insulated Oil Pipeline” on page 2-15

Concepts

- “Thermal Liquid Library” on page 2-7
- “Thermal Liquid Modeling Framework” on page 2-11

Thermal Liquid Library

In this section...

“Why Use Thermal Liquid Blocks?” on page 2-7

“Representing Thermal Liquid Components” on page 2-7

“Specifying Thermal Liquid Medium” on page 2-9

“Modeling Multidomain Systems” on page 2-9

Why Use Thermal Liquid Blocks?

The thermal behavior of liquid systems is of interest in many engineering applications. Liquids can store energy and release it back to their surroundings, often doing work in the process. Oil flow through an underground pipeline and hydraulic fluid flow in an aircraft actuator are two examples.

When temperature fluctuations are negligible, liquids behave as isothermal fluids, which simplifies the modeling process. However, when detailed thermal analysis is a goal, or when temperature fluctuations are significant, this assumption is no longer suitable.

The Thermal Liquid library provides a modeling tool that you can use to analyze the thermal behavior of *thermal* liquid systems. Three featured examples show some applications well-suited for Thermal Liquid modeling:

- `ssc_tl_oil_pipeline` — Model oil temperature along an insulated underground pipeline.
- `ssc_tl_hydraulic_fluid_warming` — Model hydraulic fluid warming due to viscous dissipation inside a hydraulic actuator.
- `ssc_tl_water_hammer` — Model the water hammer effect due to a fast-turning hydraulic valve.

Representing Thermal Liquid Components

Thermal liquid systems can range in complexity from basic to highly specialized. To model a basic system, simple components often suffice. These are components such as chambers, pipes, pumps, and the liquid medium

itself. Simple components are often industry independent and can be modeled using a single Thermal Liquid block. For example, you can model a pipeline segment using a single Pipe (TL) block.

To model a specialized system, generally you use custom components. These are components that you cannot represent by a single Thermal Liquid block. The five-way directional control valve in the `ssc_tl_hydraulic_fluid_warming` example is one such component. Custom components are often industry specific and must be modeled by grouping Thermal Liquid blocks into more complex subsystems.

The Thermal Liquid library shares the structure of other Simscape Foundation libraries. Four sublibraries supply the Thermal Liquid blocks: Elements, Sources, Sensors, and Utilities. With these sublibraries you can represent the most common components of a thermal liquid system. The table summarizes these components.

Component Type	Description	Thermal Liquid Blocks
Liquid storage	Store liquid in chambers or reservoirs.	Constant Volume Chamber (TL), Temperature Reservoir (TL), Controlled Temperature Reservoir (TL)
Liquid transport	Transport thermal liquid through closed conduits such as pipes.	Pipe (TL)
Flow restriction	Restrict thermal liquid flow, e.g., due to valves or fittings.	Local Restriction (TL), Variable Local Restriction (TL)
Mechanical interfaces	Interface thermal liquid and mechanical systems, e.g., to convert liquid mechanical energy into useful work.	Translational Mechanical Converter (TL), Rotational Mechanical Converter (TL)

Component Type	Description	Thermal Liquid Blocks
Power sources	Provide a power source to the thermal liquid system, e.g. , pressure difference or mass flow rate.	Mass Flow Rate Source (TL), Pressure Source (TL), Controlled Mass Flow Rate Source (TL), Controlled Pressure Source (TL)
Sensors	Collect measurement data for analysis of parameters, such as mass flow rate, thermal flux, pressure, and temperature.	Pressure & Temperature Sensor (TL), Mass Flow Rate & Thermal Flux Sensor (TL)
Thermal liquid	Specify thermodynamic properties and pressure-temperature validity region of thermal liquid medium.	Thermal Liquid Settings (TL)

Specifying Thermal Liquid Medium

The Thermal Liquid Settings (TL) block specifies the thermodynamic properties of the liquid medium. These properties are assumed functions of both pressure and temperature. This assumption boosts model fidelity, especially in models in which pressure, temperature, or both, vary widely.

The block accepts two-way lookup tables as input. These tables provide the different thermodynamic property values at discrete pressures and temperatures. You can populate these tables using empirical data from product data sheets or values calculated from analytical expressions.

Modeling Multidomain Systems

Thermal Liquid blocks can contain different types of conserving ports. These ports include not only Thermal Liquid conserving ports but also thermal and mechanical conserving ports. By using these ports, you can interface a Thermal Liquid subsystem with thermal and mechanical subsystems.

For instance, you can use the thermal conserving port of a Pipe (TL) block to model conductive heat transfer through a pipe wall. Oil pipeline modeling is one application. The example `ssc_tl_oil_pipeline` shows this approach.

Similarly, you can use the translational mechanical conserving ports of a Translational Mechanical Converter (TL) block to convert hydraulic pressure in a thermal liquid system into a mechanical actuation force. Hydraulic actuator modeling is one application. The example `ssc_tl_hydraulic_fluid_warming` shows this approach.

The table lists the Thermal Liquid blocks that have thermal or mechanical conserving ports. You can use these blocks to create a multidomain model containing thermal liquid, thermal, and mechanical subsystems.

Thermal Liquid Block	Thermal Conserving Port	Mechanical Conserving Port
Constant Volume Chamber (TL)	✓	
Pipe (TL)	✓	
Rotational Mechanical Converter (TL)	✓	✓
Translational Mechanical Converter (TL)	✓	✓

Related Examples

- “Heat Transfer in Insulated Oil Pipeline” on page 2-15

Concepts

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Modeling Framework” on page 2-11

Thermal Liquid Modeling Framework

In this section...

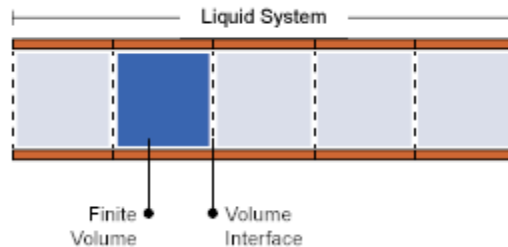
“How Blocks Represent Components” on page 2-11

“How Ports Represent Interfaces” on page 2-12

“Thermal Flux Scheme” on page 2-13

How Blocks Represent Components

Thermal Liquid models are based on the finite volume method. This method discretizes a thermal liquid system into multiple control volumes that interact via shared interfaces. An oil pipeline system is one example: you can model this system as a set of pipeline segments that connect serially along the pipeline length.

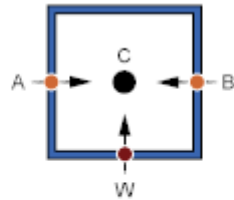


Discretization of Pipeline System

A control volume can represent a thermal liquid component, such as an oil pipeline, or a part of a component, such as a pipeline segment. You can discretize a thermal liquid system and its components as finely as you need, for example to increase simulation accuracy. However, the finer the discretization, the greater the model complexity—and the slower the simulation.

Thermal Liquid blocks represent the control volume of a component using an internal node. This node provides the liquid pressure and temperature inside the component. The node is not visible, but you can access its parameters and

variables using Simscape data logging. For more information, see “About Simulation Data Logging” on page 4-2.



- A, B — Thermal Liquid Conserving Port
- W — Thermal Conserving Port
- C — Internal Node

Simscape™ Nodes in Pipe (TL) Block

Two physical principles govern the dynamic evolution of liquid pressure and temperature at the internal node of a control volume: mass conservation and energy conservation. Pressure and temperature computation is carried out for the control volume surrounding the internal node. This control volume is the total volume of the thermal liquid component the block represents.

A second set of nodes represents the interfaces through which a finite volume can interact with its neighbors. These nodes are visible as Simscape conserving ports, of which Thermal Liquid conserving ports are the most important. By allowing the exchange of mass, momentum, and energy between adjacent liquid volumes, Thermal Liquid conserving ports govern the dynamic evolution of the finite volume as it tends to a steady state.

How Ports Represent Interfaces

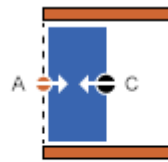
Thermal Liquid conserving ports provide the liquid pressure and temperature at the interfaces they represent. They also provide the flow rates of mass and heat, which govern the interactions between thermal liquid components. Pressure and temperature are the Across variables of the Thermal Liquid domain, while the flow rates are the Through variables.

Two physical principles govern the mass and heat flow rates through a Thermal Liquid conserving port: momentum conservation and energy

conservation. The mass flow rate at a port is computed from the momentum conservation principle. The heat flow rate at a port is computed from the thermal energy conservation principle.

The flow rate computations are carried out for half the control volume of a thermal liquid component. The half control volume is bounded on one end by the interface the port represents, and on another end by a parallel surface passing through the control volume centroid.

The figure shows the half control volume for flow rate computations at interface A of a pipeline segment. Interface A corresponds to Thermal Liquid conserving port A of a Pipe (TL) block. Node C corresponds to the internal node of the block, which is coincident with the control volume centroid.



- A — Control Volume Interface
- C — Control Volume Centroid

Half Control Volume for Flow Rate Calculations

Thermal Flux Scheme

Blocks in the Thermal Liquid library implement a full thermal flux scheme. Using this scheme, the net heat flux through a Thermal Liquid conserving port contains both convective and conductive flux contributions. By including thermal conduction in the flow direction, Thermal Liquid blocks provide more realistic simulation of the physical system they represent.

Other advantages of the full flux scheme include enhanced simulation robustness of thermal liquid models. This robustness becomes relevant in models where the conductive flux contribution can be dominant. Examples include instances of low mass flow rates and flow reversal, during which the convective flux becomes negligible or vanishes altogether.

Related Examples

- “Heat Transfer in Insulated Oil Pipeline” on page 2-15

Concepts

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Library” on page 2-7

Heat Transfer in Insulated Oil Pipeline

In this section...

“Oil Pipelines” on page 2-15

“Modeling Considerations” on page 2-16

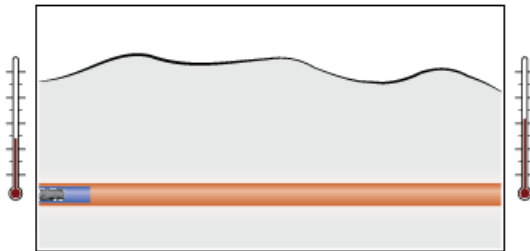
“Simscape Model” on page 2-18

“Run Simulation” on page 2-20

“Run Optimization Script” on page 2-27

Oil Pipelines

Temperature plays an important role in oil pipeline design. Below the so-called cloud point, paraffin waxes precipitate from crude oil and start to accumulate along the pipe wall interior. The waxy deposits restrict oil flow, increasing the power requirements of the pipeline. At still-lower temperatures—below the pour point of oil—these crystals become so numerous that, if allowed to quiesce, oil becomes semisolid.



In cold climates, conductive heat losses through the pipe wall can be significant. To keep oil in its favorable temperature range, pipelines include some temperature control measures. Heating stations placed at intervals along the pipeline help to warm the oil. An insulant liner covering the pipe wall interior helps to retard the cooling rate of the oil.

Viscous dissipation provides an additional heat source. As adjacent parcels of oil flow against each other, they experience energy losses that appear in the

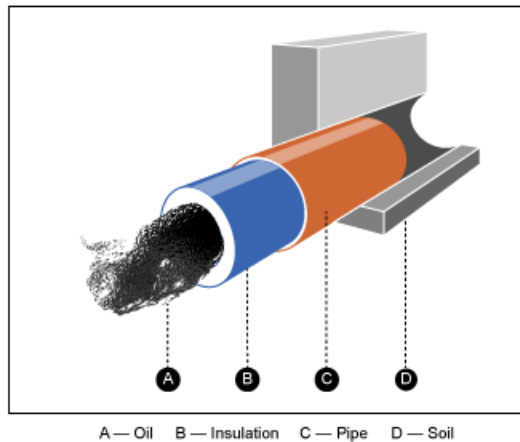
form of heat. The warming effect is small, but sufficient to at least partially offset the conductive heat losses that occur through the insulant liner.

At a certain insulation thickness, viscous dissipation exactly balances the conductive heat loss. Oil stays at its ideal temperature throughout the pipeline length and the need for heating stations is reduced. From a design standpoint, this insulation thickness is optimal.

In this example, you simulate an insulated oil pipeline segment. You then run an optimization script to determine the optimal insulation thickness. This example is based on Simscape model `ssc_tl_oil_pipeline`.

Modeling Considerations

The physical system in this example is an oil pipeline segment. Insulation lines the pipe wall interior, while soil covers the pipe wall exterior, retarding conductive heat loss. The simplifying assumption is made that the physical system is symmetric about the pipe center line.



Flow through the pipeline segment is assumed fully developed: the velocity profile of the flowing oil remains constant along the pipeline length. In addition, oil is assumed Newtonian and compressible: shear stress is proportional to the shear strain, and mass density varies with both temperature and pressure.

Oil enters the pipeline segment at a fixed temperature, $T_{Upstream}$, with a fixed mass flow rate, $V\dot{d}ot * rho0$, where:

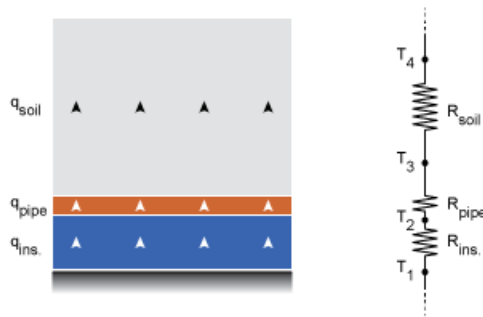
- $V\dot{d}ot$ is the volumetric flow rate of oil through the pipe.
- $rho0$ is the mass density of oil entering the pipeline segment.

Inside the pipeline segment, viscous dissipation heats the flowing oil, while thermal conduction through the pipe wall cools it. The balance between the two processes governs the temperature of oil exiting the pipeline segment.

The amount of heat *gained* through viscous dissipation depends partly on oil viscosity and mass flow rate. The greater these quantities are, the greater the viscous heat gain is—and the warmer the oil tends to get. The amount of heat *lost* via thermal conduction depends partly on the thermal resistances of the insulation, pipe wall, and soil layer. The smaller the thermal resistances are, the greater the conductive heat loss is—and the cooler the oil tends to get.

Using an electrical circuit analogy, the combined thermal resistance of three material layers arranged in series equals the sum of the individual thermal resistances:

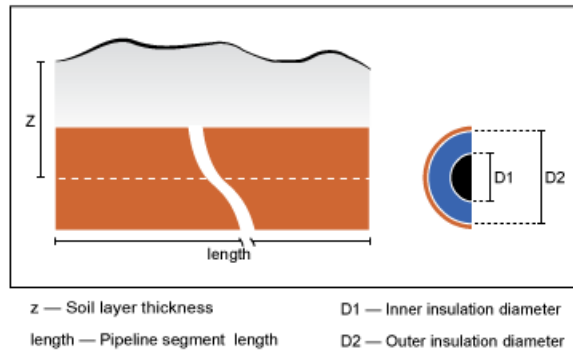
$$R_{combined} = R_{wall} + R_{ins.} + R_{soil}$$



Assuming the pipe wall is thin and its material a good thermal conductor, you can safely ignore the thermal resistance of the pipe wall. The combined thermal resistance is then simply the sum of the insulation and soil contributions, $R_{ins.}$ and R_{soil} .

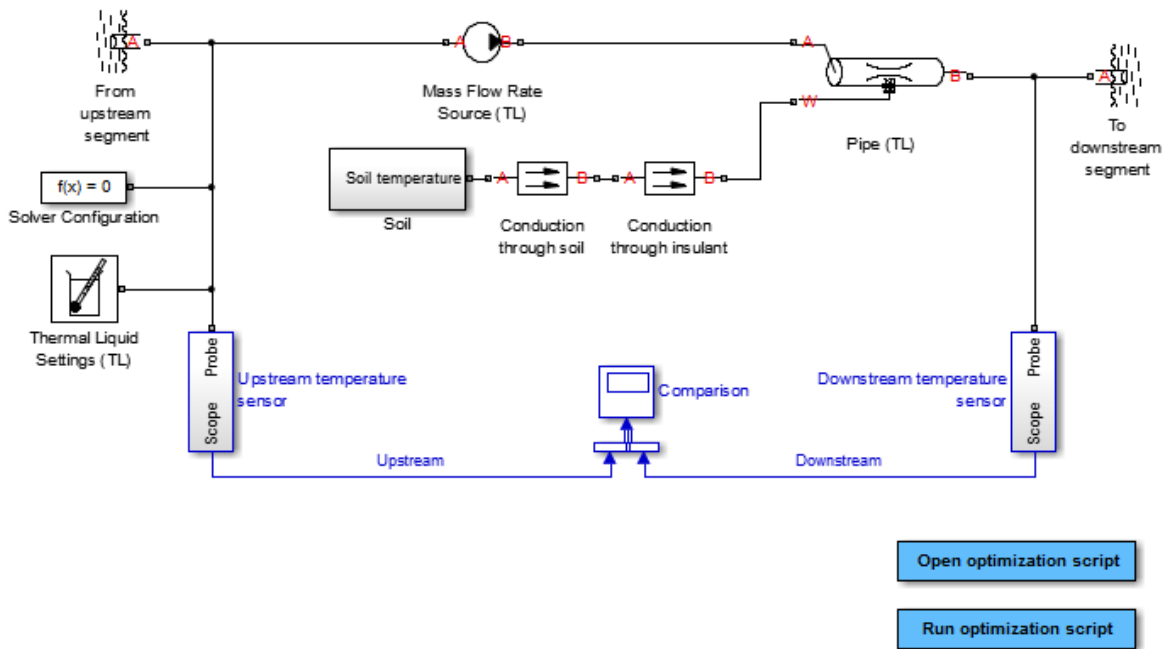
The thermal resistance of the insulation layer is directly proportional to its thickness, $(D2-D1)/2$, and inversely proportional to its thermal conductivity, $k_{Insulant}$. Likewise, the thermal resistance of the soil layer is directly proportional to its thickness, z , and inversely proportional to its thermal conductivity, k_{Soil} .

The figure shows the relevant dimensions of the pipeline segment. Variable names match those specified in the model. The inner insulation diameter, $D1$, is also the hydraulic diameter of the pipeline segment.



Simscape Model

The Simscape model `ssc_tl_oil_pipeline` represents an insulated oil pipeline segment buried underground. To open this model, at the MATLAB command prompt, enter `ssc_tl_oil_pipeline`. The figure shows the model.



The Pipe (TL) block represents the physical system in this example, i.e., the oil pipeline segment. Port A represents its inlet and port B its outlet. Port W represents thermal conduction through the pipe wall. The block accounts for viscous heating.

The Mass Flow Rate Source (TL) block provides the flow rate through the pipe. The From upstream segment block acts as a temperature source for the pipe inlet, while the To downstream segment block acts as a temperature sink at the pipe outlet.

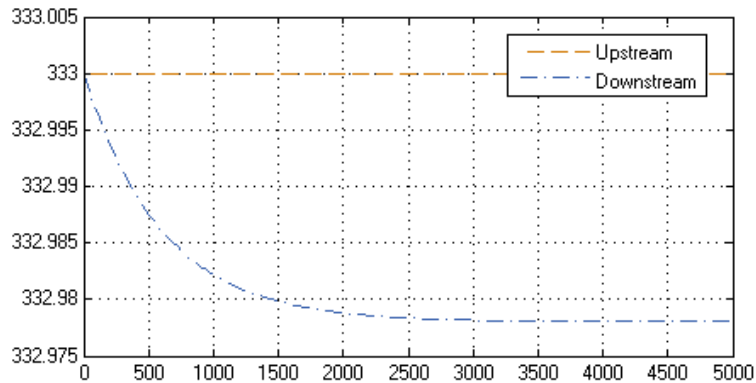
The Conduction through insulant and Conduction through soil blocks represent thermal conduction through insulant and soil layers, respectively. These blocks appear in the Simscape Thermal library as Conductive Heat Transfer. The Soil subsystem block provides the temperature boundary condition at the soil surface.

The Thermal Liquid Settings (TL) block provides the physical properties of the oil, expressed as two-sided lookup tables containing the temperature and pressure dependence of the properties. The table summarizes these blocks.

Block	Description
Pipe (TL)	Pipeline segment
Conduction through insulant	Insulant thermal conduction
Conduction through soil	Soil thermal conduction
Soil (Subsystem)	Soil temperature
From upstream segment	Pipe inlet temperature sink
To downstream segment	Pipe outlet temperature sink
Mass Flow Rate Source (TL)	Oil mass flow rate
Thermal Liquid Settings (TL)	Oil thermodynamic properties

Run Simulation

To analyze the performance of the oil pipeline segment, simulate the model. The Comparison scope plots the upstream and downstream oil temperatures. Open this scope. The insulation thickness is near its optimal value, resulting in only a small temperature change over a 1000 meter length. At a rate of ~ 0.020 K/km, oil temperature changes approximately 2 K over a 100 kilometer length.



Plot Physical Properties Using Data Logging

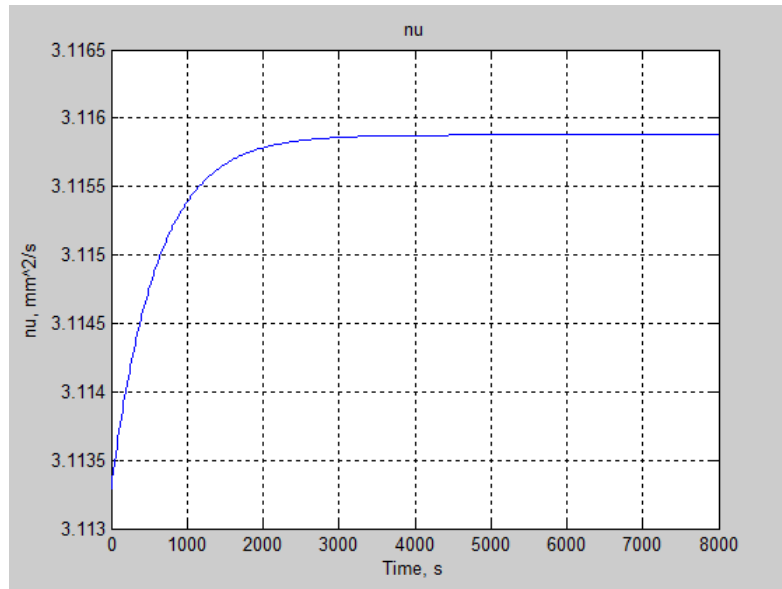
By using Simscape data logging, you can plot the physical properties of the oil as a function of simulation time. Such a plot clearly shows any variability in the value of a physical property. One example is the kinematic viscosity of oil in the pipeline segment, represented by the Pipe (TL) block.

1 At the MATLAB command line, enter `simlog.Pipe_TL.print`.

In the data tree, the kinematic viscosity `nu` appears under the node `pipe_model`, which itself appears under the node `simlog.Pipe_TL`. The logging object for the kinematic viscosity of oil in the pipe, then, is `simlog.Pipe_TL.pipe_model.nu`.

```
>> simlog.Pipe_TL.print
Pipe_TL
+-A
| +-T
| +-p
+-B
| +-T
| +-p
+-W
| +-T
+-pipe_model
+-A
| +-T
| +-p
+-B
| +-T
| +-p
+-Phi_A
+-Phi_B
+-Phi_W
+-Phi_convection_A
+-Phi_convection_B
+-T
+-W
| +-T
+-alpha
+-beta
+-cp
+-k
+-mdot_A
+-mdot_B
+-nu
+-p
+-rho
+-rho_A
+-rho_B
+-u
+-u_A
+-u_B
+-viscous_friction_A
+-viscous_friction_B
```


- At the MATLAB command line, enter `simscape.logging.plot({simlog.Pipe_TL.pipe_model.nu})`.



As expected, the kinematic viscosity remains approximately constant throughout the simulation, reflecting the minimal temperature changes that occur in the oil.

Note For more information about Simscape logging, see “About Simulation Data Logging” on page 4-2.

Simulate Effects of Changing Insulation Diameter

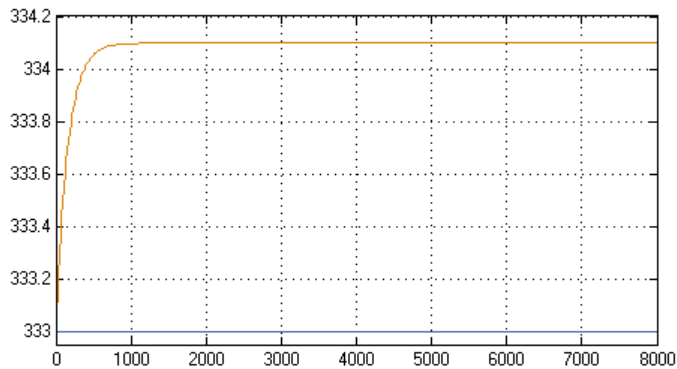
Experiment with different values for the insulation inner diameter. By varying this parameter, you offset the balance between viscous dissipation, which heats the oil, and thermal conduction, which cools the oil.

- Open Model Explorer.
- In the **Model Hierarchy** pane, select **Base Workspace**.

3 In the **Contents** pane, click the value of parameter D1.

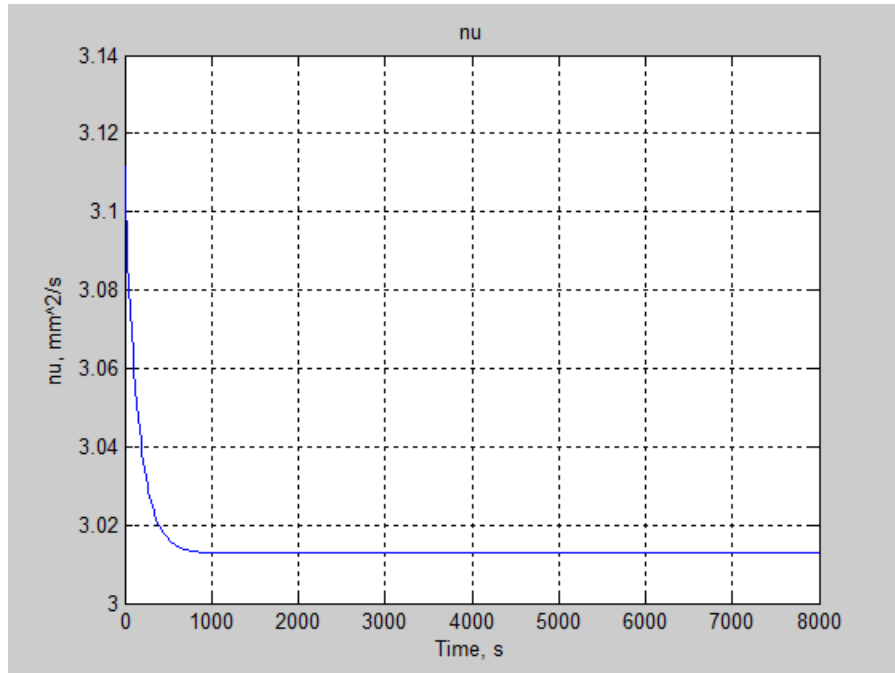
4 Enter 0.20.

By reducing the inner diameter of the insulation layer to 0.20, you increase the insulation thickness, slowing down heat loss through the pipe wall via thermal conduction. Run the simulation. Then, open the Comparison scope and autoscale to view full plot.

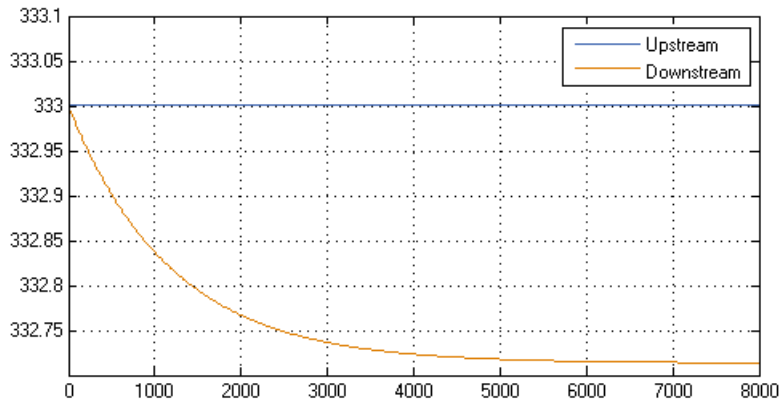


The new plot shows an oil temperature at the pipe outlet (top curve) that significantly exceeds that at the pipe inlet (bottom line). Viscous dissipation now dominates the thermal energy balance in the pipeline segment. The new insulation thickness poses a design problem: in a long pipeline, a 1.1 K/km heating rate can raise the oil temperature substantially at the receiving end of the pipeline.

Plotting the kinematic viscosity as a function of time shows that its variability is now quite significant also. At the MATLAB command line, enter the logging command: `simscape.logging.plot({simlog.Pipe_TL.pipe_model.nu})`.

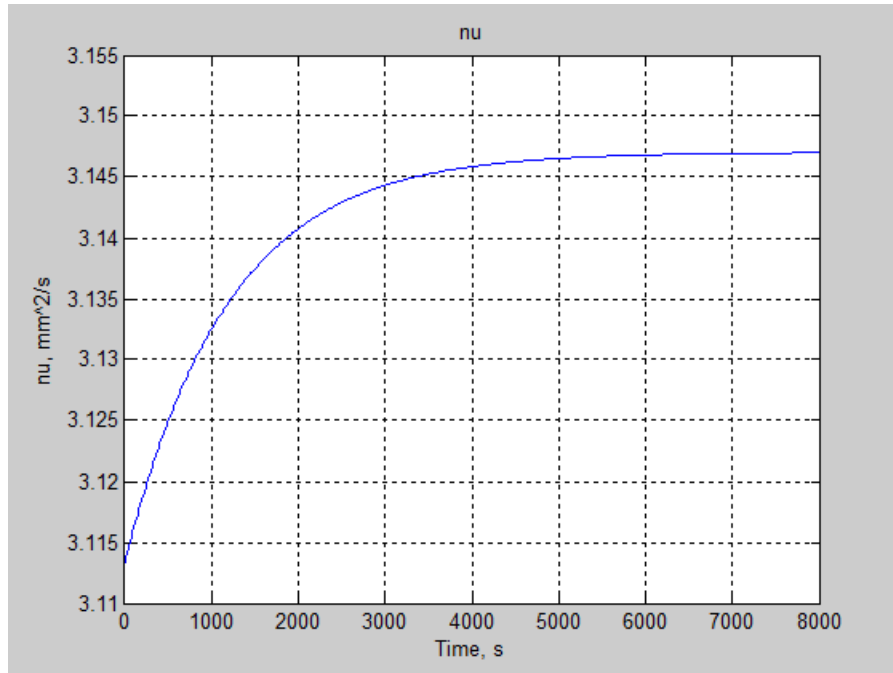


Try increasing the inner diameter of the insulation layer, D_1 , to 0.55. By increasing this value, you decrease the insulation thickness, accelerating heat loss through the pipe wall via thermal conduction. Then, run the simulation. Open the Comparison scope and autoscale to view the full plot.



The resulting plot shows that the oil temperature at the pipe outlet is now significantly lower than that at the pipe inlet. Thermal conduction clearly dominates the thermal energy balance in the pipeline segment. This insulation thickness also poses a design issue: at a rate of 0.25K/km, oil flowing through a long pipeline will cool down substantially.

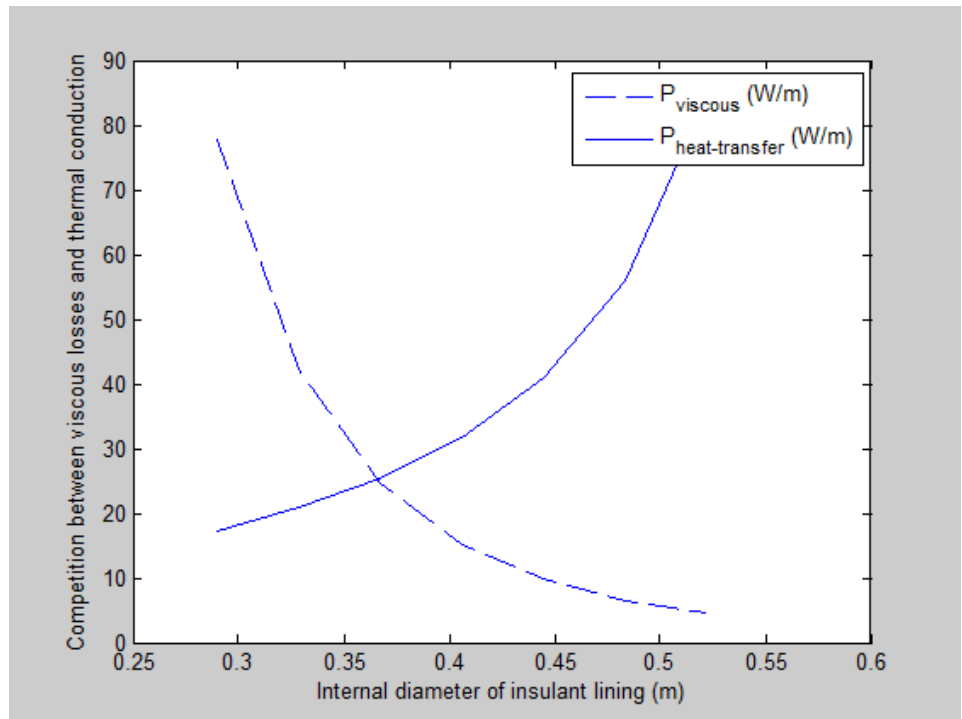
Plot the kinematic viscosity as a function of time using Simscape logging. Because the temperature change is now more modest, changes in viscosity are less significant.



Run Optimization Script

The model provides an optimization script that you can run to determine the optimal inner diameter of the pipe insulation, D_1 . The script iterates the model simulation at different D_1 values, plotting the rates of viscous warming and conductive cooling against each other. The intersection point between the two curves identifies the optimal insulation thickness for the model:

- 1 In the model window, double-click **Run optimization script**.

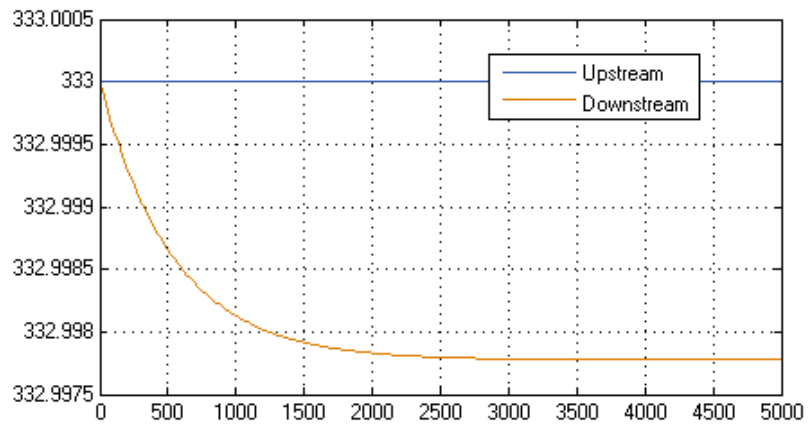


- 2 In the plot that opens, visually determine the horizontal-axis value for the intersection point between the two curves.

The optimal inner diameter of the insulation layer is 0.37 m. Update parameter D1 to this value:

- 1 Open Model Explorer.
- 2 In the **Model Hierarchy** pane, click **Base Workspace**.
- 3 In the **Contents** pane, click the value of D1.
- 4 Enter 0.37.

Now, run the simulation. Open the Comparison scope and autoscale to view the full plot. The temperature difference between the inlet and the outlet is negligible.



Concepts

- “Modeling Thermal Liquid Systems” on page 2-2
- “Thermal Liquid Library” on page 2-7
- “Thermal Liquid Modeling Framework” on page 2-11

Model Simulation

- “How Simscape Models Represent Physical Systems” on page 3-2
- “How Simscape Simulation Works” on page 3-5
- “Setting Up Solvers for Physical Models” on page 3-11
- “Customizing Solvers for Physical Models” on page 3-19
- “Troubleshooting Simulation Errors” on page 3-28
- “Code Generation” on page 3-35
- “Real-Time Simulation” on page 3-39
- “Finding an Operating Point” on page 3-48
- “Linearizing at an Operating Point” on page 3-55
- “Linearize an Electronic Circuit” on page 3-64
- “Limitations” on page 3-77
- “References” on page 3-83

How Simscape Models Represent Physical Systems

In this section...

“Representations of Physical Systems” on page 3-2

“Differential, Differential-Algebraic, and Algebraic Systems” on page 3-2

“Stiffness” on page 3-3

“Events and Zero Crossings” on page 3-3

“Working with Simscape Representation” on page 3-3

Representations of Physical Systems

This section describes important characteristics of the mathematical representations of physical systems, and how Simscape software implements such representations. You might find this overview helpful if you:

- Require details of such representations to improve your model fidelity or simulation performance.
- Are constructing your Simscape model or its components with the Simscape language.
- Need to troubleshoot Simscape modeling or simulation failures.

Mathematical representations are the foundation for physical simulation. For more information about simulation, see “How Simscape Simulation Works” on page 3-5.

Differential, Differential-Algebraic, and Algebraic Systems

The mathematical representation of a physical system contains *ordinary differential equations* (ODEs), *algebraic equations*, or both.

- ODEs govern the rates of change of *system variables* and contain some or all of the time derivatives of the system variables.
- Algebraic equations specify functional constraints among system variables, but contain no time derivatives of system variables.

- Without algebraic constraints, the system is differential (ODEs).
- Without ODEs, the system is algebraic.
- With ODEs and algebraic constraints, the system is mixed *differential-algebraic* (DAEs).

A system variable is differential or algebraic, depending on whether or not its time derivative appears in the system equations.

Stiffness

A mathematical problem is *stiff* if the solution you are seeking varies slowly, but there are other solutions within the error tolerances that vary rapidly. A stiff system has several intrinsic time scales of very different magnitude [1].

A stiff physical system has one or more components that behave “stiffly” in the ordinary sense, such as a spring with a large spring constant. Mathematical equivalents include quasi-incompressible fluids and low electrical inductance. Such systems often exhibit high frequency oscillations in some of their components or modes.

Events and Zero Crossings

Events are discontinuous changes in system state or dynamics as the system evolves in time; for example, a valve opening, or a hard stop.

A *zero crossing* is a specific event type, represented by the value of a mathematical function changing sign.

Working with Simscape Representation

A Simscape model is equivalent to a set of equations representing one or more physical systems as physical networks.

- Start by assuming that your physical network is a DAE system: a mix of differential and algebraic equations and variables.

Remember that some physical networks are represented by ODEs only.

- Physical networks may contain stiff differential equations.

- Identify discrete and continuous components that might change discontinuously during a simulation.

Creating and Detecting Zero Crossings in Simscape Models

Simulink and Simscape software have specific methods for detecting and locating zero-crossing events. For general information, see “Zero-Crossing Detection” in the Simulink documentation.

Your model can contain zero-crossing conditions arising from several sources:

- Simscape and normal Simulink blocks copied from their respective block libraries
- Expressions programmed in the Simscape language

You can disable zero-crossing detection on individual blocks, or globally across the entire model. Zero-crossing detection often improves simulation accuracy, but can slow simulation speed.

Tip If the exact times of zero crossings are important in your model, then keep zero-crossing detection enabled. Disabling it can lead to major simulation inaccuracies.

Enabling and Disabling Zero-Crossing Conditions in Simscape Language. In the Simscape language, you can create or avoid Simulink zero-crossing conditions in your model by switching between different implementations of discontinuous conditional expressions. You can:

- Use relational operators, which create zero-crossing conditions. For example, programming the operator relation: $a < b$ creates a zero-crossing condition.
- Use relational functions, which do not create zero-crossing conditions. For example, programming the functional relation: $1t(a,b)$ does not create a zero-crossing condition.

How Simscape Simulation Works

In this section...

“Simscape Simulation Phases” on page 3-5

“Model Validation” on page 3-7

“Network Construction” on page 3-7

“Equation Construction” on page 3-8

“Initial Conditions Computation” on page 3-8

“Transient Initialization” on page 3-9

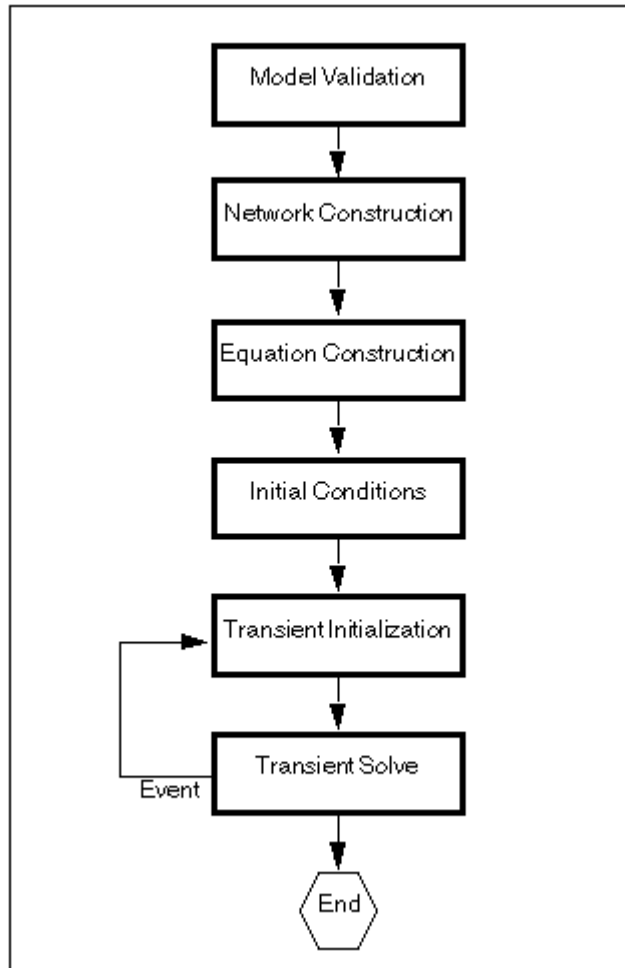
“Transient Solve” on page 3-9

Simscape Simulation Phases

You might find this brief overview helpful for constructing models and understanding errors. For more information, see “How Simscape Models Represent Physical Systems” on page 3-2.

Simscape software gives you multiple ways to simulate and analyze physical systems in the Simulink environment. Running a physical model simulation is similar to simulating any Simulink model. It entails setting various simulation options, starting the simulation, and viewing the simulation results. This topic describes various aspects of simulation specific to Simscape models. For specifics of simulating and analyzing with individual Simscape add-on products, refer to the documentation for those individual add-on products.

This flow chart presents the Simscape simulation sequence.



The flow chart consists of the following major phases:

- 1** “Model Validation” on page 3-7
- 2** “Network Construction” on page 3-7
- 3** “Equation Construction” on page 3-8
- 4** “Initial Conditions Computation” on page 3-8

5 “Transient Initialization” on page 3-9

6 “Transient Solve” on page 3-9

Model Validation

The Simscape solver first validates the model configuration and checks your data entries from the block dialog boxes.

- All Simscape blocks in a diagram must be connected into one or more physical networks. Unconnected Conserving ports are not allowed.
- Each topologically distinct physical network in a diagram requires exactly one Solver Configuration block.
- If your model contains hydraulic elements, each topologically distinct hydraulic circuit in a diagram must connect to a Custom Hydraulic Fluid block (or Hydraulic Fluid block, available with SimHydraulics block libraries). These blocks define the fluid properties that act as global parameters for all the blocks that connect to the hydraulic circuit. If no hydraulic fluid block is attached to a loop, the hydraulic blocks in this loop use the default fluid. However, more than one hydraulic fluid block in a loop generates an error.

Similarly, if your model contains pneumatic elements, default gas properties for a pneumatic network are for dry air and ambient conditions of 101325 Pa and 20 degrees Celsius. If you attach a Gas Properties block to a pneumatic circuit, you can change gas properties and ambient conditions for all the blocks connected to the circuit. However, more than one Gas Properties block in a pneumatic circuit generates an error.

- Signal units specified in a Simulink-PS Converter block must match the input type expected by the Simscape block connected to it. For example, when you provide the input signal for an Ideal Angular Velocity Source block, specify angular velocity units, such as rad/s or rpm, in the Simulink-PS Converter block, or leave it unitless. Similarly, units specified in a PS-Simulink Converter block must match the type of physical signal provided by the Simscape block output.

Network Construction

After validating the model, the Simscape solver constructs the physical network based on the following principles:

- Two directly connected Conserving ports have the same values for all their Across variables (such as voltage or angular velocity).
- Any Through variable (such as current or torque) transferred along the Physical connection line is divided among the multiple components connected by the branches. For each Through variable, the sum of all its values flowing into a branch point equals the sum of all its values flowing out.

Equation Construction

Based on the network configuration, the parameter values in the block dialog boxes, and the global parameters defined by the fluid properties, if applicable, the Simscape solver constructs the system of equations for the model.

These equations contain system variables of the following types:

- *Dynamic* — Time derivatives of these variables appear in equations. Dynamic variables are the independent states for simulation.
- *Algebraic* — Time derivatives of these variables do not appear in equations. The states of algebraic variables are always dependent, on dynamic variables, other algebraic variables, or inputs.

Initial Conditions Computation

The Simscape solver computes the initial conditions only once, in the beginning of simulation ($t=0$). In the Solver block dialog box, the default is that the **Start simulation from steady state** check box is not selected.

The solver computes the initial conditions by setting all dynamic variables to 0, except those corresponding to blocks that have an initial condition field in their block dialog boxes, and solving for all the system variables. The blocks with initial conditions have their dynamic variables set according to the user-provided value in the block dialog. Initial conditions can be set only on dynamic variables, because dynamic variables are the independent states for simulation. For example, the Translational Spring block has the **Initial deformation** parameter, so the corresponding spring position state is set to the initial offset specified in the block dialog box. To find which blocks have initial conditions specified through their dialog boxes, refer to the block reference documentation.

The initial conditions for dependent dynamic states must be set consistently. For example, the initial voltages on two parallel, ideal capacitors must be equal. When the solver detects dependent dynamic variables, it performs a check and issues an error if the initial conditions on dynamic states are not set consistently.

Finding an Initial Steady State

When you select the **Start simulation from steady state** check box, the solver attempts to find the steady state that would result if the inputs to the system were held constant for a long enough time, starting from the initial state obtained from the initial conditions computation just described. If the steady-state solve succeeds, the state found is some steady state (within tolerance), but not necessarily the state expected from the given initial conditions. Steady state means that the system variables are no longer changing with time. Simulation then starts from this steady state.

Note If the simulation fails at or near the start time when you use the **Start simulation from steady state** option, try clearing the check box and simulating with the default initial conditions computation only.

Transient Initialization

After computing the initial conditions, or after a subsequent event (such as a discontinuity resulting, for example, from a valve opening, or from a hard stop), the Simscape solver performs transient initialization. Transient initialization fixes all dynamic variables and solves for algebraic variables and derivatives of dynamic variables. The goal of transient initialization is to provide a consistent set of initial conditions for the next phase, transient solve.

Transient Solve

Finally, the Simscape solver performs transient solve of the system of equations. In transient solve, continuous differential equations are integrated in time to compute all the variables as a function of time.

The solver continues to perform the simulation according to the results of the transient solve until the solver encounters an event, such as a zero crossing or discontinuity. The event may be within the physical network or elsewhere in

the Simulink model. If the solver encounters an event, the solver returns to the phase of transient initialization, and then back to transient solve. This cycle continues until the end of simulation.

Setting Up Solvers for Physical Models

In this section...

“About Simulink and Simscape Solvers” on page 3-11

“Choosing Simulink and Simscape Solvers” on page 3-11

“Harmonizing Simulink and Simscape Solvers” on page 3-13

About Simulink and Simscape Solvers

This section explains how to select solvers for physical simulation. Proper simulation of Simscape models requires certain changes to Simulink defaults and consideration of physical simulation trade-offs. For recommended choices, see “Customizing Solvers for Physical Models” on page 3-19.

Choosing Simulink and Simscape Solvers

Simulink and Simscape solver technologies provide a range of tools to simulate physical systems, including the powerful Simscape technique of local solvers. You choose global, or model-wide, solvers through Simulink. After making these choices, check that they are consistent; see “Harmonizing Simulink and Simscape Solvers” on page 3-13.

- “Working with Global Simulink Solvers” on page 3-11
- “Working with Local Simscape Solvers” on page 3-12

Working with Global Simulink Solvers

In the Configuration Parameters dialog box of your model, on the **Solver** pane, the solver and related settings that you select are global choices. For more information, see “Choose a Solver” in the Simulink documentation.

When you first create a model, the default Simulink solver is ode45. To select a different solver, follow a procedure similar to the procedure in “Modifying Initial Settings” on page 1-26.

- You can choose one from a suite of both variable-step and fixed-step solvers. A variable-step solver is the default.

- You can also select from among explicit and implicit solvers. An explicit solver is the default. But for physical models, MathWorks® recommends implicit solvers, such as ode14x, ode23t, and ode15s. Implicit solvers require fewer time steps than explicit solvers, such as ode45, ode113, and ode1.

See “Switching from the Default Explicit Solver to Other Simulink Solvers” on page 3-14.

- If all the Simulink and Simscape states in your model are discrete, Simulink automatically switches to a discrete solver and issues a warning. Otherwise, a continuous solver is the default.
- By default, Simulink variable-step solvers attempt to locate events in time by zero-crossing detection. See “Enabling or Disabling Simulink Zero-Crossing Detection” on page 3-17.

Working with Local Simscape Solvers

You can switch one or more physical networks to a local implicit, fixed-step Simscape solver by selecting **Use local solver** in the network Solver Configuration block. The solver and related settings you make in each Solver Configuration block are specific to the connected physical network and can differ from network to network.

A physical network using a local solver appears to the global Simulink solver as if it has discrete states. You can still use any continuous global solver.

Choosing Local Solvers and Sample Times. To use a local solver, choose a solver type (Backward Euler or Trapezoidal Rule) and a sample time. Backward Euler is the default.

Choosing Fixed-Cost Simulation. You can select a fixed-cost simulation for one or more physical networks by selecting **Use fixed-cost runtime consistency iterations**, as well as **Use local solver**, and fixing the number of nonlinear and mode iterations. Fixed-cost simulation requires a global fixed-step solver.

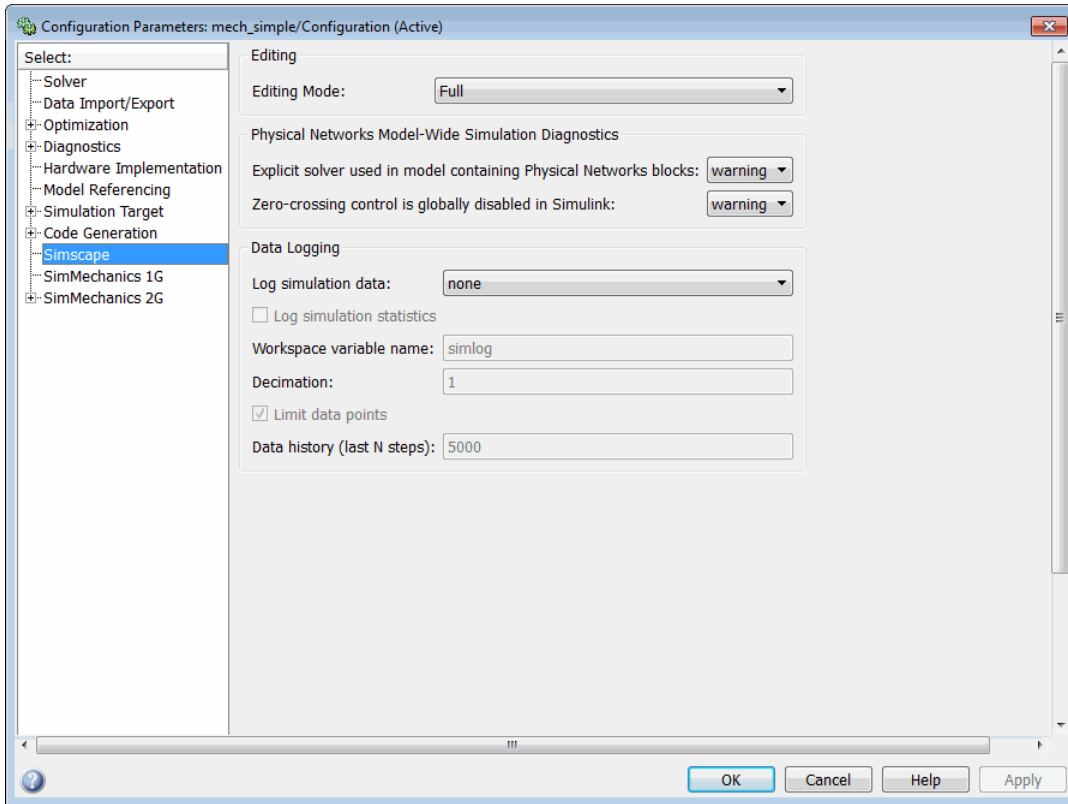
Choosing Multirate Simulation. With the local solver option, you can perform multirate simulations, with:

- Different sample times in different physical networks, through their respective Solver Configuration blocks
- A sample-based Simulink block in the model with a sample time different from the Solver Configuration block or blocks

Harmonizing Simulink and Simscape Solvers

Your Simulink and Simscape solver choices must work together consistently. To ensure consistency of your Simulink and Simscape solver choices for a particular model, open the model Configuration Parameters dialog box. From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. Review and adjust the following settings.

- “Switching from the Default Explicit Solver to Other Simulink Solvers” on page 3-14
- “Filtering Input Signals and Providing Time Derivatives” on page 3-15
- “Enabling or Disabling Simulink Zero-Crossing Detection” on page 3-17
- “Making Multirate Simulation Consistent” on page 3-18



Simscape™ Pane of the Configuration Parameters Dialog Box

Switching from the Default Explicit Solver to Other Simulink Solvers

If you do not modify the default (explicit) solver, your performance may not be optimal. Implicit solvers are better for most physical simulations. For more information about implicit solvers and physical systems, see “Customizing Solvers for Physical Models” on page 3-19.

Diagnostic Messages About Explicit Solvers. When you use an explicit solver in a model containing Simscape blocks, the system issues a warning to alert you to a potential problem.

To turn off this default warning or to change it to an error message, go to the **Simscape** pane of the Configuration Parameters dialog box:

1 From the **Explicit solver used in model containing Physical Networks blocks** drop-down list, select the option that you want:

- **warning** — If the model uses an explicit solver, the system issues a warning upon simulation. This is the default option that alerts you to a potential problem if you use the default solver.
- **error** — If the model uses an explicit solver, the system issues an error message upon simulation. If your model is stiff, and you do not want to use explicit solvers, select this option to avoid future errors.
- **none** — If the model uses an explicit solver, the system issues no warning or error message upon simulation. If you want to work with explicit solvers, in particular for models that are not stiff, select this option.

2 Click **OK**.

Filtering Input Signals and Providing Time Derivatives

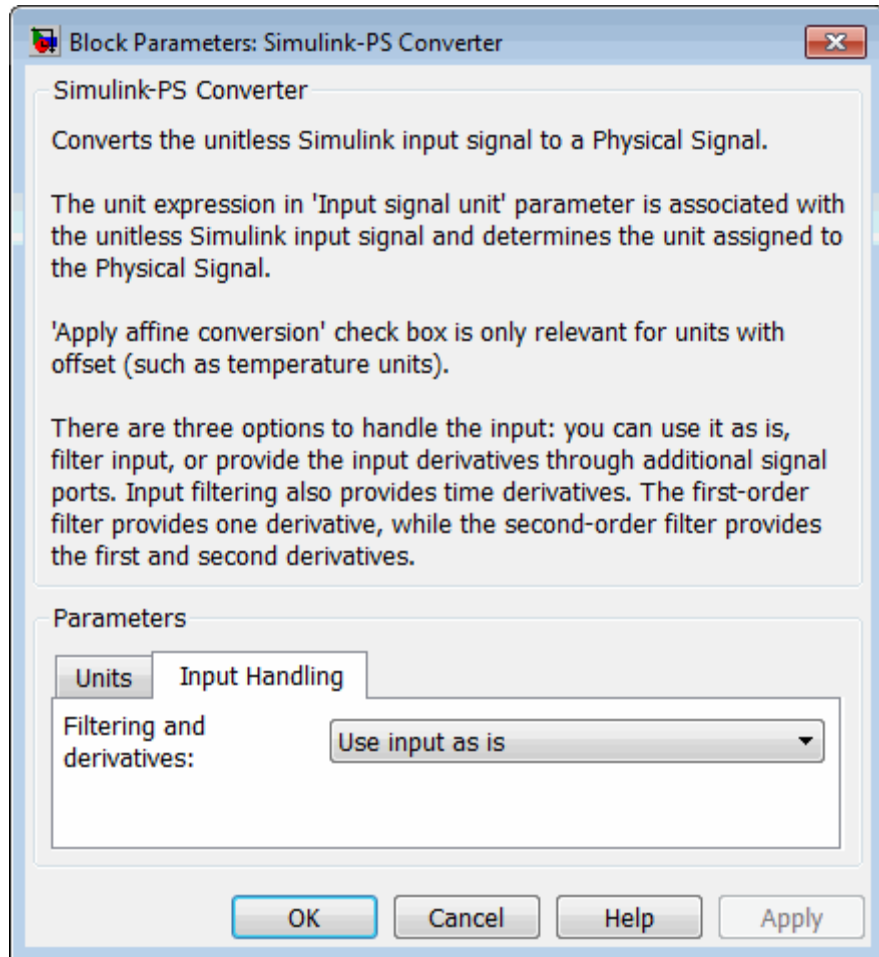
You may need to provide time derivatives of some of the input signals, especially if you use an explicit solver. One way of providing the necessary input derivatives is by filtering the input through a low-pass filter. Input filtering makes the input signal smoother and generally improves model performance. The additional benefit is that the Simscape engine computes the time derivatives of the filtered input. The first-order filter provides one derivative, while the second-order filter provides the first and second derivatives. If you use input filtering, it is very important to select the appropriate value for the filter time constant.

The filter time constant controls the filtering of the input signal. The filtered input follows the true input but is smoothed, with a lag on the order of the time constant that you choose. Set the time constant to a value no larger than the smallest time interval in the system that interests you. If you choose a very small time constant, the filtered input signal is closer to the true input signal. However, this filtered input signal increases the stiffness of the system and slows the simulation.

Instead of using input filtering, you can provide time derivatives for the input signal directly, as additional physical signals.

You can control the way you provide time derivatives for each input signal by configuring the Simulink-PS Converter block connected to that input signal:

- 1 Open the Simulink-PS Converter block dialog box.
- 2 Click the **Input handling** tab.



- 3 To turn on input filtering, set the **Filtering and derivatives** parameter to **Filter input**. Select the first-order or second-order filter, by using the

Input filtering order parameter, and set the appropriate **Input filtering time constant** parameter value for your model.

4 To avoid filtering the input signal, set the **Filtering and derivatives** parameter to **Provide input derivative(s)**. Then set the **Input derivatives** parameter value:

- **Provide first derivative** — If you select this option, an additional Simulink input port appears on the Simulink-PS Converter block, to let you connect the signal providing input derivatives.
- **Provide first and second derivatives** — If you select this option, two additional Simulink input ports appear on the Simulink-PS Converter block, to let you connect the signals providing input derivatives.

Enabling or Disabling Simulink Zero-Crossing Detection

By default, Simulink tracks an important class of simulation events by detecting zero crossings. With a global variable-step solver and without a local solver, Simulink attempts to locate the simulated times of zero crossings, if present. See “Working with Simscape Representation” on page 3-3.

Diagnostic Messages About Globally Disabling Zero-Crossing Detection.

You can globally disable zero-crossing detection in the **Solver** pane of the Configuration Parameters dialog box, under **Zero-crossing options**. If you do, and if you are using a global variable-step solver without a local solver, the system issues a warning or error when you simulate with Simscape blocks.

You can choose between warning and error messages in the **Simscape** pane of the Configuration Parameters dialog box.

1 From the **Zero-crossing control is globally disabled in Simulink** drop-down list, select the option that you want, if you globally disable zero-crossing detection:

- **warning** — The system issues a warning message upon simulation. This option is the default.
- **error** — The system issues an error message upon simulation, which stops.

2 Click **OK**.

Making Multirate Simulation Consistent

The sample time or step size of the global Simulink solver must be the smallest time step of all the solvers in a multirate Simscape simulation.

To avoid simulation errors in sample time propagation, go to the **Solver** pane in the Configuration Parameters dialog box and select the **Automatically handle rate transition for data transfer** check box.

Customizing Solvers for Physical Models

In this section...
“Important Concepts and Choices in Physical Simulation” on page 3-19
“Making Optimal Solver Choices for Physical Simulation” on page 3-22

Important Concepts and Choices in Physical Simulation

This section describes advanced concepts and trade-offs you might want to consider as you configure and test solvers and other simulation settings for your Simscape model. For a summary of recommended settings, see “Making Optimal Solver Choices for Physical Simulation” on page 3-22. For background information, consult “How Simscape Models Represent Physical Systems” on page 3-2 and “How Simscape Simulation Works” on page 3-5.

- “Variable-Step and Fixed-Step Solvers” on page 3-19
- “Explicit and Implicit Solvers” on page 3-20
- “Full and Sparse Linear Algebra” on page 3-21
- “Event Detection and Location” on page 3-21
- “Unbounded, Bounded, and Fixed-Cost Simulation” on page 3-21
- “Global and Local Solvers” on page 3-22

Variable-Step and Fixed-Step Solvers

Variable-step solvers are the usual choice for design, prototyping, and exploratory simulation, and to precisely locate events during simulation. They are not useful for real-time simulation and can be costly if there are many events.

A variable-step solver automatically adjusts its step size as it moves forward in time to adapt to how well it controls solution error. You control the accuracy and speed of the variable-step solution by adjusting the solver tolerance. With many variable-step solvers, you can also limit the minimum and maximum time step size.

Fixed-step solvers are recommended or required if you want to make performance comparisons across platforms and operating systems, to generate a code version of your model, and to bound or fix simulation cost. A typical application is real-time simulation. For more information, see “Code Generation” on page 3-35 and “Real-Time Simulation” on page 3-39.

With a fixed-step solver, you specify the time step size to control the accuracy and speed of your simulation. Fixed-step solvers do not adapt to improve accuracy or to locate events. These limitations can lead to significant simulation inaccuracies.

Explicit and Implicit Solvers

The degree of stiffness and the presence of algebraic constraints in your model influence the choice between an *explicit* or *implicit* solver. Explicit and implicit solvers use different numerical methods to simulate a system.

- If the system is a nonstiff ODE system, choose an explicit solver. Explicit solvers require less computational effort than implicit solvers, if other simulation characteristics are fixed.

To find a solution for each time step, an explicit solver uses a formula based on the local gradient of the ODE system.

- If the system is stiff, use an implicit solver. Though an explicit solver may require less computational effort, for stiff problems an implicit solver is more accurate and often essential to obtain a solution. Implicit solvers require per-step iterations within the simulated time steps. With some implicit solvers, you can limit or fix these iterations.

An implicit solver starts with the solution at the current step and iteratively solves for the solution at the next time step with an algebraic solver. An implicit algorithm does more work per simulation step, but can take fewer, larger steps.

- If the system contains DAEs, even if it is not stiff, use an implicit solver. Such solvers are designed to simultaneously solve algebraic constraints and integrate differential equations.

Full and Sparse Linear Algebra

When you simulate a system with more than one state, the solver manipulates the mathematical system with matrices. For a large number of states, sparse linear algebra methods applied to large matrices can make the simulation more efficient.

Event Detection and Location

Events, in most cases, occur between simulated time steps.

- Fixed-step solvers detect events after “stepping over” them, but cannot adaptively locate events in time. This can lead to large inaccuracies or failure to converge on a solution.
- Variable-step solvers can both detect events and estimate the instants when they occur by adapting the timing and length of the time steps.

Tip To estimate the timing of events or rapid changes in your simulation, use a variable-step solver.

If your simulation has to frequently adapt to events or rapid changes by changing its step size, much or all of the advantage of implicit solvers over explicit solvers is lost.

Unbounded, Bounded, and Fixed-Cost Simulation

In certain cases, such as real-time simulation, you need to simulate with an execution time that is not only bounded, but practically fixed to a predictable value. Fixing execution time can also improve performance when simulating frequent events.

The real-time cost of a variable-step simulation is potentially unlimited. The solver can take an indefinite amount of real time to solve a system over a finite simulated time, because the number and size of the time steps are adapted to the system. You can configure a fixed-step solver to take a bounded amount of real time to complete a simulation, although the exact amount of real time might still be difficult to predict before simulation. Even a fixed-step solver can take multiple iterations to find a solution at each time

step. Such iterations are variable and not generally limited in number; the solver iterates as much as it needs to.

Fixing execution time implies *fixed-cost* simulation, which both fixes the time step and limits the number of per-step iterations. Fixed-cost simulation prevents *execution overruns*, when the execution time is longer than the simulation sample time. A bounded execution time without a known fixed cost might still cause some steps to overrun the sample time.

The actual amount of computational effort required by a solver is based on a number of other factors as well, including model complexity and computer processor. For more information, see “Real-Time Simulation” on page 3-39.

Global and Local Solvers

You can use different solvers on different parts of the system. For example, you might want to use implicit solvers on stiff parts of a system and explicit solvers everywhere else. Such *local solvers* make the simulation more efficient and reduce computational cost.

Such multisolver simulations must coordinate the separate sequences of time steps of each solver and each subsystem so that the various solvers can pass simulation updates to one another on some or all of the shared time steps.

Making Optimal Solver Choices for Physical Simulation

For the key simulation concepts to consider before making these choices, see “Important Concepts and Choices in Physical Simulation” on page 3-19.

- “Simulating with Variable Time Step” on page 3-23
- “Simulating with Fixed Time Step — Local and Global Fixed-Step Solvers” on page 3-23
- “Simulating with Fixed Cost” on page 3-24
- “Troubleshooting and Improving Solver Performance” on page 3-25
- “Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System” on page 3-26

Simulating with Variable Time Step

For a typical Simscape model, MathWorks recommends the Simulink variable-step solvers ode15s and ode23t. Of these two global solvers:

- The ode15s solver is more stable, but tends to damp out oscillations.
- The ode23t solver captures oscillations better but is less stable.

With Simscape models, these solvers solve the differential and algebraic parts of the physical model simultaneously, making the simulation more accurate and efficient.

Simulating with Fixed Time Step – Local and Global Fixed-Step Solvers

In a Simscape model, MathWorks recommends that you implement fixed-step solvers by continuing to use a global variable-step solver and switching the physical networks within your model to local fixed-step solvers through each network Solver Configuration block. The local solver choices are Backward Euler and Trapezoidal Rule. Of these two local solvers:

- The Backward Euler tends to damp out oscillations, but is more stable, especially if you increase the time step.
- The Trapezoidal Rule solver captures oscillations better but is less stable.

Regardless of which local solver you choose, the Backward Euler method is always applied:

- Right at the start of simulation.
- Right after an instantaneous change, when the corresponding block undergoes an internal discrete change. Such changes include clutches locking and unlocking, valve actuators opening and closing, and the switching of the Asynchronous Sample & Hold block.

Switching to Discrete States and Solvers.

- If you switch a physical network to a local solver, the global solver treats that network as having discrete states.

- If other physical networks in your model are not using local solvers, or if the non-Simscape parts of your model have continuous states, then you must use a continuous global solver.
- If all physical networks in your model use local solvers, and any non-Simscape parts of your model have only discrete states, then the global solver effectively sees only discrete states. In that case, MathWorks recommends a discrete, fixed-step global solver. If you are attempting a fixed-cost simulation with discrete states, you must use a discrete, fixed-step global solver.

For Maximum Accuracy with Fixed-Step Simulation. If solution accuracy is your single overriding requirement, use the global Simulink fixed-step solver `ode14x`, without local solvers. This implicit solver is the best global fixed-step choice for physical systems. While it is more accurate than the Simscape local solvers for most models, `ode14x` can be computationally more intensive and slower when you use it by itself than it is when you use it in combination with local solvers.

In this solver, you must limit the number of global implicit iterations per time step. Control these iterations with the **Number Newton's iterations** parameter in the **Solver** pane of the Configuration Parameters dialog box.

Simulating with Fixed Cost

Many Simscape models need to iterate multiple times within one time step to find a solution. If you want to fix the cost of simulation per time step, you must limit the number of these iterations, regardless of whether you are using a local solver, or a global solver like `ode14x`. For more information, see “Unbounded, Bounded, and Fixed-Cost Simulation” on page 3-21 and “Real-Time Simulation” on page 3-39.

To limit the iterations, open the Solver Configuration block of each physical network. Select **Use fixed-cost runtime consistency iterations** and set limits for the number of nonlinear and mode iterations per time step.

Tip Fixed-cost simulation with variable-step solvers is not possible in most simulations. Attempt fixed-cost simulation with a fixed-step solver only and avoid using fixed-cost iterations with variable-step solvers.

Troubleshooting and Improving Solver Performance

Consider the basic trade-off of speed versus accuracy and stability. A larger time step or tolerance results in faster simulation, but also less accurate and less stable simulation. If a system undergoes sudden or rapid changes, larger tolerance or step size can cause major errors. Consider tightening the tolerance or step size if your simulation:

- Is not accurate enough or looks unphysical.
- Exhibits discontinuities in state values.
- Reaches the minimum step size allowed without converging, usually a sign that one or more events or rapid changes occur within a time step.

Any one or all of these steps increase accuracy, but make the simulation run more slowly.

For Local Solvers. Models with friction or hard stops are particularly difficult for local solvers, and may not work or may require a very small time step.

With the Trapezoidal Rule solver, oscillatory “ringing” can become more of a problem as the time step is increased. For a larger time step in a local solver, consider switching to Backward Euler.

For ODE Systems. In certain cases, your model reduces to an ODE system, with no dependent algebraic variables. (See “How Simscape Models Represent Physical Systems” on page 3-2.) If so, you can use any global Simulink solver, with no special physical modeling considerations. An explicit solver is often the best choice in such situations.

- Through careful analysis, you can sometimes determine if your model is represented by an ODE system.
- If you create a Simscape model from a mathematical representation using the Simscape language, you can determine directly if the resulting system is ODE.

For Large Systems. Depending on the number of system states, you can simulate more efficiently if you switch the value of the **Linear Algebra** setting in the Solver Configuration block.

For smaller systems, Full provides faster results. For larger systems, Sparse is typically faster.

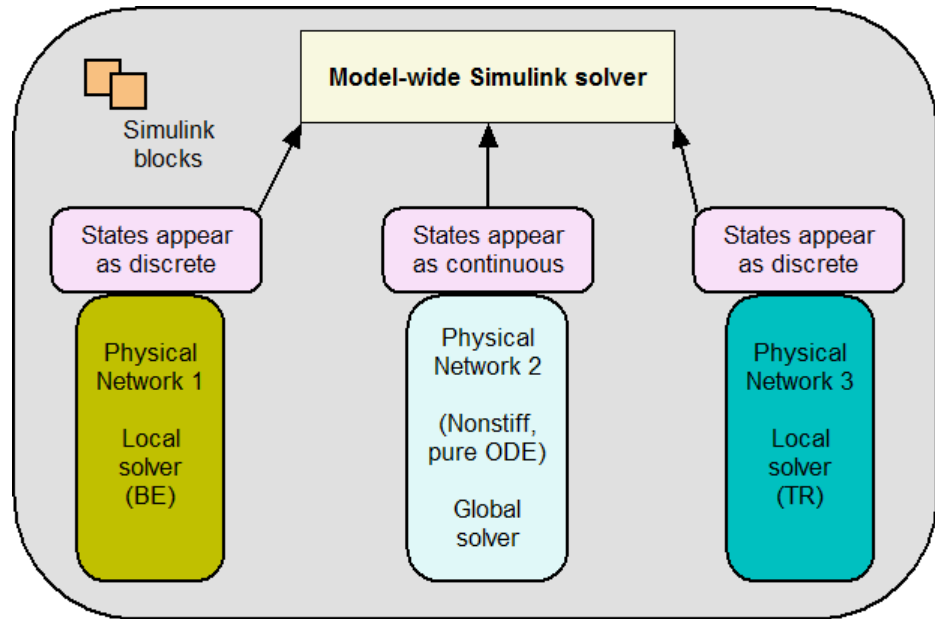
Multiple Local Solvers Example with a Mixed Stiff-Nonstiff System

In this example, a Simscape model contains three physical networks.

- Two networks (numbers 1 and 3) use local solvers, making these two networks appear to the global solver as if they had discrete states. Internally, these networks still have continuous states. These networks are moderately and highly stiff, respectively.

One of these networks (number 1) uses the Backward Euler (BE) local solver. The other (number 3) uses the Trapezoidal Rule (TR) local solver.

- The remaining network (number 2) uses the global Simulink solver. Its states appear to the model as continuous. This network is not stiff and is pure ODE. Use an explicit global solver.
- Because at least one network appears to the model as continuous, you must use a continuous solver. However, if you remove network 2, and if the model contains no continuous Simulink states, Simulink automatically switches to a discrete global solver.



Troubleshooting Simulation Errors

In this section...

“Troubleshooting Tips and Techniques” on page 3-28

“System Configuration Errors” on page 3-29

“Numerical Simulation Issues” on page 3-32

“Initial Conditions Solve Failure” on page 3-32

“Transient Simulation Issues” on page 3-33

Troubleshooting Tips and Techniques

Simscape simulations can stop before completion with one or more error messages. This section discusses generic error types and error-fixing strategies. You might find the previous section, “How Simscape Simulation Works” on page 3-5, useful for identifying and tracing errors.

If a simulation failed:

- Review the model configuration. If your error message contains a list of blocks, look at these blocks first. Also look for:
 - Wrong connections — Verify that the model makes sense as a physical system. For example, look for actuators connected against each other, so that they try to move in opposite directions, or incorrect connections to reference nodes that prevent movement. In electrical circuits, verify polarity and connections to ground.
 - Wrong units — Simscape unit manager offers great flexibility in using physical units. However, you must exercise care in specifying the correct units, especially in the Simulink-PS Converter and PS-Simulink Converter blocks. Start analyzing the circuit by opening all the converter blocks and checking the correctness of specified units.
- Try to simplify the circuit. Unnecessary circuit complexity is the most common cause of simulation errors.
- Break the system into subsystems and test every unit until you are positive that the unit behaves as expected.

- Build the system by gradually increasing its complexity.

MathWorks recommends that you build, simulate, and test your model incrementally. Start with an idealized, simplified model of your system, simulate it, verify that it works the way you expected. Then incrementally make your model more realistic, factoring in effects such as friction loss, motor shaft compliance, hard stops, and the other things that describe real-world phenomena. Simulate and test your model at every incremental step. Use subsystems to capture the model hierarchy, and simulate and test your subsystems separately before testing the whole model configuration. This approach helps you keep your models well organized and makes it easier to troubleshoot them.

System Configuration Errors

- “Missing Solver Configuration Block” on page 3-29
- “Extra Fluid Block or Gas Properties Block” on page 3-29
- “Missing Reference Block” on page 3-30
- “Basic Errors in Physical System Representation” on page 3-30

Missing Solver Configuration Block

Each topologically distinct Simscape block diagram requires exactly one Solver Configuration block to be connected to it. The Solver Configuration block specifies the global environment information and provides parameters for the solver that your model needs before you can begin simulation.

If you get an error message about a missing Solver Configuration block, open the Simscape Utilities library and add the Solver Configuration block anywhere on the circuit.

Extra Fluid Block or Gas Properties Block

If your model contains hydraulic elements, each topologically distinct hydraulic circuit in a diagram requires a Custom Hydraulic Fluid block (or Hydraulic Fluid block, available with SimHydraulics block libraries) to be connected to it. These blocks define the fluid properties that act as global parameters for all the blocks connected to the hydraulic circuit. If no

hydraulic fluid block is attached to a loop, the hydraulic blocks in this loop use the default fluid. However, more than one hydraulic fluid block in a loop generates an error.

Similarly, more than one Gas Properties block in a pneumatic circuit generates an error.

If you get an error message about too many domain-specific global parameter blocks attached to the network, look for an extra Hydraulic Fluid block, Custom Hydraulic Fluid block, or Gas Properties block and remove it.

Missing Reference Block

Simscape libraries contain domain-specific reference blocks, which represent reference points for the conserving ports of the appropriate type. For example, each topologically distinct electrical circuit must contain at least one Electrical Reference block, which represents connection to ground. Similarly, hydraulic conserving ports of all the blocks that are referenced to atmosphere (for example, suction ports of hydraulic pumps, or return ports of valves, cylinders, pipelines, if they are considered directly connected to atmosphere) must be connected to a Hydraulic Reference block, which represents connection to atmospheric pressure. Mechanical translational ports that are rigidly clamped to the frame (ground) must be connected to a Mechanical Translational Reference block, and so on.

If you get an error message about a missing reference block, or node, check your system configuration and add the appropriate reference block based on the rules described above. The missing reference node diagnostic messages include information about the particular block and variable that needs a reference node. This is especially helpful when multiple domains are involved in the model. For more information and examples of best modeling practices, see “Grounding Rules” on page 1-36.

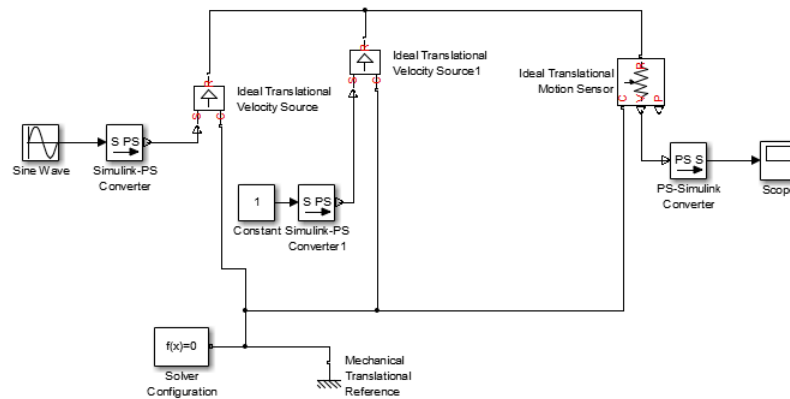
Basic Errors in Physical System Representation

Physical systems are represented in the Simscape modeling environment as Physical Networks according to the Kirchhoff’s generalized circuit laws. Certain model configurations violate these laws and are therefore illegal. There are two broad violations:

- Sources of domain-specific Across variable connected in parallel (for example, voltage sources, hydraulic pressure sources, or velocity sources)
- Sources of domain-specific Through variable connected in series (for example, electric current sources, hydraulic flow rate sources, force or torque sources)

These configurations are impossible in the real world and illegal theoretically. If your model contains such a configuration, upon simulation the solver issues an error followed by a list of blocks, as shown in the following example.

Example. The model shown in the following illustration contains two Ideal Translational Velocity Sources connected in parallel. This produces a loop of independent velocity sources, and the solver cannot construct a consistent system of equations for the circuit.



When you try to simulate the model, the solver issues an error message with links to the Ideal Translational Velocity Source and Ideal Translational Velocity Source1 blocks. To fix the circuit, you can either replace the two velocity sources by a single Ideal Translational Velocity Source block, or add a Translational Damper block between them.

Numerical Simulation Issues

- “Dependent Dynamic States” on page 3-32
- “Parameter Discontinuities” on page 3-32

Numerical simulation issues can be either a result of certain circuit configurations or of parameter discontinuities.

Dependent Dynamic States

Certain circuit configurations can result in dependent dynamic states, or the so-called higher-index differential algebraic equations (DAEs). Simscape solver can handle dependencies among dynamic states that are linear in the states and independent of time and inputs to the system. For example, capacitors connected in parallel or inductors connected in series will not cause any problems. Other circuit configurations with dependent dynamic states, in certain cases, may slow down the simulation or lead to an error when the solver fails to initialize.

Problems may occur when dynamic states have a nonlinear algebraic relationship. An example is two inertias connected by a nonlinear gear constraint, such as an elliptical gear. In case of simulation failure, the Simscape solver may be able to identify the components involved, and provide an error message with links to the blocks and to the equations within each block.

Parameter Discontinuities

Nonlinear parameters, dependent on time or other variables, may also lead to numerical simulation issues as a result of parameter discontinuity. These issues usually manifest themselves at the transient initialization stage (see “Transient Simulation Issues” on page 3-33).

Initial Conditions Solve Failure

The initial conditions solve, which solves for all system variables (with initial conditions specified on some system variables), may fail. This has several possible causes:

- System configuration error. In this case, the Simulation Diagnostics window usually contains additional, more specific, error messages, such as a missing reference node, or a warning about the component equations, followed by a list of components involved. See “System Configuration Errors” on page 3-29 for more information.
- Dependent dynamic state. In this case, the Simulation Diagnostics window also may contain additional, more specific, error messages, such as a warning about the component equations, followed by a list of components involved. See “Dependent Dynamic States” on page 3-32 for more information.
- The constraint residual tolerance may be too tight to produce a consistent solution to the algebraic constraints at the beginning of simulation. You can try to increase the **Constraint Residual Tolerance** parameter value (that is, relax the tolerance) in the Solver Configuration block.

If the Simulation Diagnostics window has other, more specific, error messages, address them first and try rerunning the simulation. See also “Troubleshooting Tips and Techniques” on page 3-28.

Transient Simulation Issues

- “Transient Initialization Not Converging” on page 3-33
- “Step-Size-Related Errors — Dependent States — High Stiffness” on page 3-34

Transient initialization happens at the beginning of simulation (after computing the initial conditions) or after a subsequent event, such as a discontinuity (for example, when a hard stop hits the stop). It is performed by fixing all dynamic variables and solving for algebraic variables and derivatives of dynamic variables. The goal of transient initialization is to provide a consistent set of initial conditions for the next transient solve step.

Transient Initialization Not Converging

Error messages stating that transient initialization failed to converge, or that a set of consistent initial conditions could not be generated, indicate transient initialization issues. They can be a result of parameter discontinuity.

Review your model to find the possible sources of discontinuity. See also “Troubleshooting Tips and Techniques” on page 3-28.

You can also try to decrease the **Constraint Residual Tolerance** parameter value (that is, tighten the tolerance) in the Solver Configuration block.

Step-Size-Related Errors – Dependent States – High Stiffness

A typical step-size-related error message may state that the system is unable to reduce the step size without violating the minimum step size for a certain number of consecutive times. This error message indicates numerical difficulties in solving the Differential Algebraic Equations (DAEs) for the model. This might be caused by dependent dynamic states (higher-index DAEs) or by the high stiffness of the system. You can try the following:

- Tighten the solver tolerance (decrease the **Relative Tolerance** parameter value in the Configuration Parameters dialog box)
- Specify a value, other than auto, for the **Absolute Tolerance** parameter in the Configuration Parameters dialog box. Experiment with this parameter value.
- Tighten the residual tolerance (decrease the **Constraint Residual Tolerance** parameter value in the Solver Configuration block)
- Increase the value of the **Number of consecutive min step size violations allowed** parameter in the Configuration Parameters dialog box (set it to a value greater than the number of consecutive step size violations given in the error message)
- Review the model configuration and try to simplify the circuit, or add small parasitic terms to your circuit to avoid dependent dynamic states. For more information, see “Numerical Simulation Issues” on page 3-32.

Code Generation

In this section...
“About Code Generation from Simscape Models” on page 3-35
“Reasons for Generating Code” on page 3-35
“Using Code-Related Products and Features” on page 3-36
“How Simscape Code Generation Differs from Simulink” on page 3-36

About Code Generation from Simscape Models

You can use Simulink Coder™ software to generate stand-alone C or C++ code from your Physical Networks models and enhance simulation speed and portability. Certain features of Simulink software also make use of generated or external code. This section explains code-related tasks you can perform with your Simscape models.

Code versions of Simscape models typically require fixed-step Simulink solvers, which are discussed in the Simulink documentation. Some features of Simscape software are restricted when you translate a model into code. See “How Simscape Code Generation Differs from Simulink” on page 3-36, as well as “Limitations” on page 3-77.

Note Code generated from Simscape models is intended for rapid prototyping and hardware-in-the-loop applications. It is not intended for use as production code in embedded controller applications.

Add-on products based on the Simscape platform also support code generation, with some variations and exceptions described in their respective documentation.

Reasons for Generating Code

Code generation has many purposes and methods. There are two essential rationales:

- Compiled code versions of Simulink and Simscape models run faster than the original block diagram models. The time savings can be dramatic.
- An equally important consideration for Simscape models is the stand-alone implementation of generated and compiled code. Once you convert part or all of your model to code, you can deploy the stand-alone executable program on virtually any platform, independently of MATLAB.

Converting a model or subsystem to code also hides the original model or subsystem.

Using Code-Related Products and Features

With Simulink, Simulink Coder, and xPC Target™ software, using several code-related technologies, you can link existing code to your models and generate code versions of your models.

Code-Related Task	Component or Feature
Link existing code written in C or other supported languages to Simulink models	Simulink S-functions to generate customized blocks
Speed up Simulink simulations	Accelerator mode Rapid Accelerator mode
Generate stand-alone fixed-step code from Simulink models	Simulink Coder software
Generate stand-alone variable-step code from Simulink models	Simulink Coder Rapid Simulation Target (RSim)
Convert Simulink model to code and compile and run it on a target PC	Simulink Coder and xPC Target software

How Simscape Code Generation Differs from Simulink

In general, using the code generated from Simscape models is similar to using code generated from regular Simulink models. However, there are certain differences.

Simscape and Simulink Code Generated Separately

Simulink Coder software generates code from the Simscape blocks separately from the Simulink blocks in your model. The generated Simscape code does not pass through `model.rtw` or the Target Language Compiler. All the code generated from a single model resides in the same directory, however.

Compiler and Processor Architecture Requirements

To generate and execute Simscape code, you must have a compiler and a processor that support:

- 64-bit precision floating-point arithmetic
- 32-bit integer size

For details on supported compiler versions, see

http://www.mathworks.com/support/compilers/current_release

Precompiled Libraries Provided for Selected Compilers

Simscape software and its add-on products provide static runtime libraries precompiled for compilers supported by Simulink Coder software. These are the standard UNIX compilers for UNIX operating systems, `gcc` and Microsoft® Visual Studio® for 32-bit Windows®, and Microsoft Visual Studio for 64-bit Windows.

For all other compilers, the static runtime libraries needed by code generated from Simscape models are compiled once per model during the code generation build process.

Simscape Code Reuse Not Supported

Reusable subsystems in Simulink reuse code that is generated once from the subsystem. You cannot generate reusable code from subsystems containing Simscape blocks.

Tunable Parameters Not Supported

A tunable parameter is a Simulink run-time parameter that you can change while the simulation is running. Simscape blocks do not support tunable parameters in either simulations or generated code.

Simscape Run-Time Parameter Inlining Override of Global Exceptions

If you choose to enable parameter inlining for code generated from a Simscape model, the software inlines all its run-time parameters. If you choose to make some of the global Simscape block parameters exceptions to inlining, the exceptions are ignored. You can change global tunable parameters only by regenerating code from the model.

Real-Time Simulation

In this section...

“What Is Real-Time Simulation?” on page 3-39

“Requirements for Real-Time Simulation” on page 3-40

“Simulating Physical Models in Real Time” on page 3-41

“Preparing a Model for Real-Time Simulation” on page 3-42

“Troubleshooting Real-Time Simulation Problems” on page 3-45

What Is Real-Time Simulation?

Real-time simulation of an engineering system becomes possible when you replace physical devices with virtual devices. This replacement reduces costs and improves the quality of physical and control systems, including their software, by enabling more complete testing of the entire system. It also enables continuous testing, without interruption and under possibly dangerous conditions. Real-time simulation allows you to test even when you have no prototypes.

Real-time simulation becomes a necessity if you want to simulate a system realistically responding to its environment. Such realistic simulation means that the inputs and outputs in the virtual world of simulation must be read or updated synchronously with the real world. When the simulation clock reaches a certain time in real-time simulation, the same amount of time must have passed in the real world.

Using Real-Time Simulation to Test Virtual Controllers and Systems

In desktop simulation, you use models to develop and test control and signal processing algorithms. Once the designs are complete and you have converted these algorithms to embedded code, you must test that code as well as the actual controller. If the model is capable of running in real time, you can use the model created in the design phase to test the embedded code and processor, instead of connecting it directly to a hardware prototype. Such real-to-virtual substitution, simulating in real time, is referred to as *hardware-in-the-loop* (HIL) testing.

Example

Systems with a human in the simulation loop require real-time simulation. For example, flight simulators that train pilots require real-time simulation of the plane, its control system, the weather, and other environmental conditions.

Requirements for Real-Time Simulation

Configuring a model and a numerical integrator to simulate in real time is often more challenging than ordinary simulation. You simulate with a more restrictive version of the universal computational tradeoff of accuracy versus speed.

The simulation execution time per time step must be consistently short enough to permit any other tasks that the simulation environment must perform, such as reading sensor input or generating output actuator signals. This requirement must be satisfied even if the simulation changes its qualitative character: the system stiffness might change, and discrete components can switch states. Such changes occasionally require more computations to achieve an accurate result.

Bounding and Stabilizing Execution Time with Fixed-Step Solvers and Fixed-Cost Simulation

When real-time simulation is the goal, the execution time per simulation time step must be bounded. Variable-step solvers, which are often used in desktop simulation, take smaller steps to accurately capture events that occur during the simulation. But you cannot vary the step size in a real-time simulation. Instead, you must

- Choose a fixed-step solver that can capture the system dynamics accurately and minimize the amount of computation required per time step, without changing the step size. If the system states are all discrete, the fixed-step solver can be discrete as well.

If you choose a small enough step size, most fixed-step solvers produce the same simulation results as a variable-step solver. However, different fixed-step solvers (implicit/explicit, lower/higher order, and so on) require different step sizes to produce accurate results. They also require different amounts of computation per time step.

- Choose a fixed step size large enough to permit stable real-time simulation. The step size must not be so large that the simulation results are inaccurate, but not so small that real-time simulation is impossible.

You often need trial and error to find the right combination of settings that satisfy both criteria.

Real-time simulation requires not only bounding the execution time, but fixing it to a stable value. This requires a fixed-cost simulation method. For more information, see “Customizing Solvers for Physical Models” on page 3-19.

Simulating Physical Models in Real Time

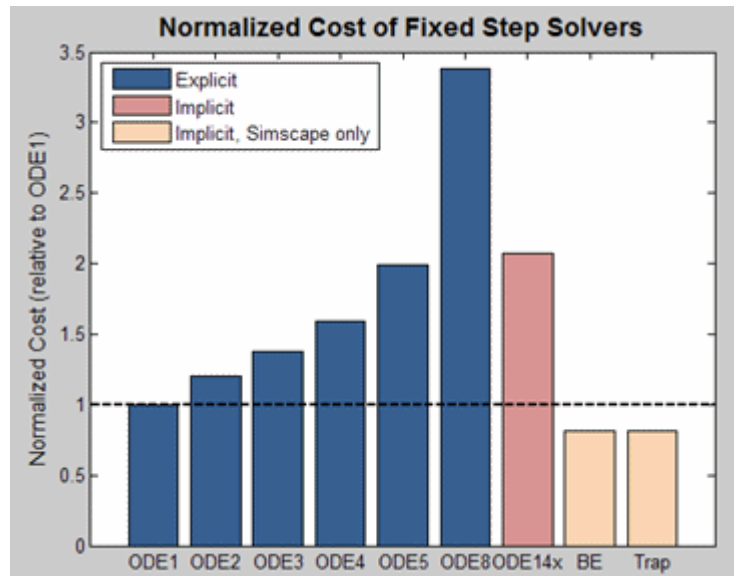
Achieving real-time simulation with any Simscape model includes:

- Enabling simulation with fixed-step, fixed-cost solvers
- Converting the model with Simulink Coder to code for a particular computer hardware target
- Testing real-time simulation on PC-compatible hardware with xPC Target, if desired

For more information, see “Code Generation” on page 3-35.

Preparation for real-time simulation requires particular choices and adjustment of Simulink variable-step solvers. Actual real-time simulation requires Simulink fixed-step solvers. Certain Simscape features enable and enhance real-time simulation of physical systems with Simulink fixed-step solvers, both explicit and implicit. These features include fixed-cost algorithms and local solvers, with the trapezoidal rule or backward Euler method. See “Customizing Solvers for Physical Models” on page 3-19.

This figure plots the normalized computational cost of all fixed-step solvers available for Simscape models, obtained for a nonlinear model example with one physical network. For comparison, the step size was kept the same, with similar settings for the total number of solver iterations.



Preparing a Model for Real-Time Simulation

To move from desktop to real-time simulation on your real-time hardware target, adjust the following simulation properties until the simulation can execute in real time and deliver results close to the results from desktop simulation:

- Solver choice
- Number of solver iterations
- Simulation time step size
- Model size and fidelity

Follow these high-level tasks to prepare a model for real-time simulation. Each task is also a link to specific instructions for that part of the procedure.

- 1** “Simulate and Converge with Variable-Step Solver” on page 3-43
- 2** “Check Variable Time Steps for Optimal Step Size” on page 3-43

- 3 “Simulate with Fixed-Cost Solver and Compare to Variable-Step Simulation” on page 3-44
- 4 “Adjust Step Size and Iterations to Approximate Variable-Step Simulation Results” on page 3-44
- 5 “Attempt to Simulate in Real Time” on page 3-45
- 6 “Respond to Real-Time Simulation Failures” on page 3-45

Simulate and Converge with Variable-Step Solver

The first task is to obtain a converged set of results with a variable-step solver.

To ensure that the results obtained with the fixed-step solver are accurate, you need a set of reference results. You can obtain these by simulating the system with a variable-step solver. Ensure that the results converge by tightening the error tolerances until the simulation results do not change significantly.

Check Variable Time Steps for Optimal Step Size

The second task is to examine the time step sizes during the desktop simulation and determine if the model is likely to run with a large enough step size to permit real-time simulation.

A variable-step solver varies the step size to keep the solution within error tolerances and to react to zero crossing events. If the solver abruptly reduces the step size to a small value (for example, $1e-15$ s), the solver is trying to accurately identify a zero crossing event. A fixed-step solver might have trouble capturing these events at a step size large enough to permit real-time simulation.

Analysis of these particular variable time steps provides an estimate of a step size that can be used to run the simulation. Modifying or eliminating the effects are causing these events makes it easier to simulate the system with a fixed-step solver at a reasonably large step size and produce results comparable to the variable-step simulation. See “Troubleshooting Real-Time Simulation Problems” on page 3-45.

Simulate with Fixed-Cost Solver and Compare to Variable-Step Simulation

The third task is to simulate the system with a fixed-step, fixed-cost solver and compare these results to the reference results from the variable-step simulation.

Limiting Per-Step Solver Iterations. Simulating physical systems often requires multiple iterations per time step to converge on a solution. To perform a fixed-cost simulation, you must limit these iterations. In each physical network Solver Configuration block, select the **Use fixed-cost runtime consistency iterations** check box and enter the number of allowed iterations.

Switching to Local Solvers. You can further minimize the computations done per time step by choosing a local solver on each physical network in the model. To switch to a local solver in a physical network, open the Solver Configuration block of that network and select **Use local solver**. By using this option, you can use an implicit fixed-step solver only on the stiff portions of the model and an explicit fixed-step solver on the remainder of the model. This minimizes the computations done per time step, making it more likely that the model can run in real time.

Adjust Step Size and Iterations to Approximate Variable-Step Simulation Results

The fourth task is to reduce the step size and adjust the number of nonlinear iterations, in order to produce results that are sufficiently close to the reference results from variable-step simulation. The step size must still be large enough for a safety margin to prevent an execution overrun.

During each time step, the real-time simulation must calculate the result for the next time step (simulation execution), and read inputs and write outputs (I/O processing and other tasks). If these actions take less time than the specified time step, the processor remains idle during the remainder of the step. Choosing a computationally more intensive solver, increasing the number of nonlinear iterations, or reducing the step size both increases the simulation accuracy and reduces the amount of idle time, raising the risk that the simulation cannot run in real time. Adjusting these settings in the opposite way increases the amount of idle time but reduce accuracy.

Estimating the budget for the execution time helps ensure that you choose a feasible combination of settings. If you know the amount of time spent processing inputs and outputs and performing other actions, as well as the percentage of idle time that you want, the amount of time available for simulation execution can be calculated as follows:

$$\text{Simulation Execution Time Budget} = \text{Step Size} - [\text{I/O Processing Time} + (\text{Desired Percentage of Idle Time}) (\text{Step Size})]$$

Estimating Real-Time Execution Time. You can use the desktop simulation speed to estimate the execution time on a real-time hardware target. Many factors affect the real-time target execution time, so that comparing processor speeds might not be sufficient.

A better method is to measure the execution time of desktop simulation and then to determine the average execution time per time step on the real-time target for a particular model. Knowing how these execution times compare for one model means that you can estimate execution time on the real-time target from the desktop simulation execution time when you test other models.

Attempt to Simulate in Real Time

The fifth task is to use the selected solver, the number of nonlinear iterations, and the step size to simulate on the real-time target and to verify if the simulation can run in real time.

If the simulation does not run in real time on the target hardware, the model might not be real-time capable.

Respond to Real-Time Simulation Failures

If the simulation does not run in real time on the selected real-time target, perform a sixth, contingent task, described in “Troubleshooting Real-Time Simulation Problems” on page 3-45.

Troubleshooting Real-Time Simulation Problems

If the simulation does not run in real time on the real-time platform, or if the simulation performance is unacceptable, you should determine the causes and find an appropriate solution. The combination of effects captured in the

model and the speed of the real-time platform might make it impossible to find solver settings that permit it to run in real time. Consider the following options to make it real-time capable.

Once you modify your model, return to the third, fourth, and fifth tasks of “Preparing a Model for Real-Time Simulation” on page 3-42 to identify and implement the appropriate settings to enable real-time simulation.

Speeding Up Real-Time Execution

You can speed up the real-time simulation by using a faster real-time target computer.

Alternatively, you can achieve the same goal by determining new model settings that permit a larger step size or reduce the execution time (for example, by reducing the number of nonlinear iterations).

Simulating Parts of the System in Parallel

If possible, configure the model to evaluate multiple physical networks in parallel. You can do this if the networks are not dependent upon one another. You need experience and experimentation with your model, the generated code, and the real-time target to make effective use of this option.

Eliminating Effects That Require Intensive Computation

Certain effects in your model can prevent real-time simulation. Such effects include instantaneous events and rapid changes in parts of the system with very small time constants. Identify and modify or remove these elements before searching again for a combination of solver settings and step size that permits real-time simulation.

Identifying Elements Causing Rapid or Instantaneous Changes. Watch for certain system elements becoming excited to high frequencies. Examine the system eigenmodes to isolate which system states have the highest frequency. Mapping those states to individual components often points to the source of the problem. Because you can only do this at a particular operating point, choose an operating point corresponding to simulation times in the variable-step simulation that had small step sizes. At such simulation times, the variable-step solver is struggling to simulate a rapid change.

With scripts written in MATLAB, you can interrogate the model, identify these components quickly, and narrow the search for the effects that you need to modify. You can automate and extend these searches to other models with tools like the Simulink Model Advisor. The troublesome components that you need to locate include:

- Elements that create events and change the solution nearly instantaneously. A fixed-step solver might not be able to step over such rapid changes and find the right solution on the other side of the event. If it fails to find the solution, the solver may become unstable. Examples of elements that create these kinds of events include:
 - Hard stops or backlash
 - Stick-slip friction
 - Switches or clutches
- Elements with very small time constants. The dynamics of these elements require a small step size so that a fixed-step solver can accurately simulate them, perhaps too small for real-time simulation. Examples of systems with a small time constant include:
 - Small masses attached to stiff springs with minimal damping
 - Electrical circuits with small capacitance and inductance and low resistance
 - Hydraulic circuits with small compressible volumes

Modifying or Removing Elements Causing Rapid or Instantaneous Changes. Once you have identified these elements, change or eliminate them by:

- Replacing nonlinear components with linearized versions
- Replacing complex equations with lookup tables for their solution
- Replacing complicated components with simplified models by using system identification theory on their input and output data
- Smoothing discontinuous functions (step changes) by using filters, delays, and other techniques.

Finding an Operating Point

In this section...

“What Is an Operating Point?” on page 3-48

“Some Operating Point Search Methods” on page 3-49

“Finding Operating Points in Physical Models” on page 3-50

What Is an Operating Point?

An *operating point* of a system is a dynamic configuration that satisfies design and use requirements called *operating specifications*. You can express such operating specifications as requirements on the system state \mathbf{x} and inputs \mathbf{u} . It is not always possible to find a dynamic state that satisfies all operating conditions. Also, a system might have multiple operating points satisfying the same requirements.

Operating points are essential for designing and implementing system controllers. You can optimize a system at an operating point for performance, stability, safety, and reliability.

The most important and common type of operating point is a *steady state*, where some or all of the system dynamic variables are constant.

Using Operating Points for Linearization

An important motive for finding operating points is *linearization*, which determines the system response to small disturbances at an operating point. Linearization results influence the design of feedback controllers to govern dynamic behavior near the operating point. A full linearization analysis requires one or more system outputs, \mathbf{y} , in addition to inputs.

See “Linearizing at an Operating Point” on page 3-55.

Example

A pilot flying an aircraft wants to find, for a given environment, a state of the aircraft engine and control surfaces that produces level, constant-velocity, and constant-altitude flight relative to the ground. The requirements of “level,”

"constant velocity," "constant altitude," and "relative to the ground" constitute operating specifications. This operating point is a steady state of the aircraft velocity, altitude, and orientation in space.

Some Operating Point Search Methods

You can provide predefined state and input vectors, \mathbf{x}_0 and \mathbf{u}_0 , to specify an operating point. If you do not know an operating point in advance, you have two methods of identifying an operating point that satisfies operating specifications.

- “Time-Based Search” on page 3-49: Observing the actual or simulated behavior of the system in time is more general, but less precise, and usually requires a trial-and-error process to find a precise operating point.
- “State-Based Search” on page 3-49: If you know the system dynamics, you can solve for steady states, at least in principle.

Time-Based Search

You can sometimes find operating points and steady states by trial and error while operating or simulating over some length of time and varying the system parameters, inputs, and initial conditions. In such a time-based approach, you isolate and study instants or intervals of time when a system satisfies the operating specifications. The system state and inputs under those conditions constitute the operating point, which you can also specify by an operating or simulation time.

State-Based Search

The alternative to trial-and-error searching for steady states is *trimming*. In this state-based approach, you bypass time-based simulation and find solutions for inputs, outputs, states, and state derivatives satisfying an operating specification. Trimming specifies inputs and part of a state and solves the system dynamics for the rest of the state. The resulting full state and input vectors, \mathbf{x}_0 and \mathbf{u}_0 , constitute the operating point.

In general, there is no guarantee that such solutions, \mathbf{x}_0 , exist for given operating specifications and inputs, \mathbf{u}_0 .

Checking Discrete System States

An operating point includes the state of discrete system variables that change in a discontinuous way. In general, you cannot find these states by small, continuous changes of system variables. Such states usually require systematic exploration of the discrete variables over the full range of their possible values.

Finding Operating Points in Physical Models

You have a number of ways to find an operating point in a Simscape model. You can impose operating specifications and isolate operating points using Simscape and Simulink features.

Tip To find a steady state, the Simscape steady-state solver is the most direct method. For a comprehensive suite of operating point and linearization tools, MathWorks recommends Simulink Control Design™ software.

To analyze operating points, you work with the full state vector of your model, which contains:

- Simulink components, which can be continuous or discrete.
- Simscape components, which are continuous.

Whichever method that you choose to find an operating point, if you want to use it for linearization, you must save the operating point information in the form of an operating point object, a simulation time t_0 , or a state vector \mathbf{x}_0 and input vector \mathbf{u}_0 .

- “Simulating in Time to Search for an Operating Point” on page 3-51
- “Using the Simscape Steady-State Solver” on page 3-51
- “Using Simulink® Control Design™ Techniques to Find Operating Points” on page 3-52
- “Using Sources to Find Operating Points Not Recommended” on page 3-54
- “Simulink trim Function Not Supported with Simscape Models” on page 3-54

Simulating in Time to Search for an Operating Point

One way to identify operating points is to simulate your model and inspect its state \mathbf{x} and output \mathbf{y} as a time series.

- 1 In your Simscape model, set up sensor outputs for whatever block outputs you want to observe.
- 2 Connect Scope blocks, To Workspace blocks, or both, to your Simscape block outputs to observe and record simulation behavior.
- 3 In the **Data Import/Export** pane of your model Configuration Parameters settings, select the **Time**, **States**, and **Output** check boxes to record this simulation information in your workspace.

Using the Simscape Steady-State Solver

Most commonly, the operating point that you want is a steady state. The Simscape steady-state solver allows you to isolate steady states more exactly than you can with ordinary simulation. It is the only practical method to isolate steady states of a strongly nonlinear character. You can search for multiple steady states with the steady-state solver by varying the model inputs, parameters, and initial conditions.

Before simulation starts, the steady-state solver determines the model steady state $\mathbf{x}(t=0) = \mathbf{x}_0$. In general, the system does not remain in this initial steady state \mathbf{x}_0 during simulation, because the system inputs \mathbf{u} change independently, and the system has to respond by changing its state $\mathbf{x}(t)$.

To implement the steady-state solver:

- 1 In each, some, or all of the physical networks in your Simscape model, open the Solver Configuration block.
- 2 In each block dialog box, select the **Start simulation from steady state** check box.
- 3 In the model Configuration Parameters settings, on the **Data Import/Export** pane, select the **States** check box to record the time series of \mathbf{x} values in your workspace.

If you also have input signals \mathbf{u} in the model, you can capture those inputs by connecting To Workspace blocks to the input Simulink signal lines.

4 Close these dialog boxes and start simulation.

The first vector of values $\mathbf{x}(t=0)$ that you capture during simulation reflects the steady state \mathbf{x}_0 that the Simscape solver identified.

Tip Finding an initial steady state is part of the nondefault Simscape simulation sequence. See “Initial Conditions Computation” on page 3-8.

You can simplify the initial steady-state computation by setting the simulation time to 0. The simulation then solves for one time step only (time zero) and returns a single state vector $\mathbf{x}(t=0)$.

Using Simulink Control Design Techniques to Find Operating Points

Note The techniques described in this section require the Simulink Control Design product.

You must use the features of this product on the Simulink lines in your model, not directly on Simscape physical network lines or blocks. Simulink Control Design offers both command line and graphical interfaces for finding and analyzing operating points.

Simulink Control Design methods are state-based, giving you full access to state names and values, and allow you to impose operating specifications or use simulation snapshots. They work well for simple to moderately complex Simscape models. MathWorks does not recommend these methods for highly complex Simscape models.

To find operating points, it is simplest to use the `operspec` and `findop` functions, customizing where necessary. Create an operating specification object with `operspec`, then compute an operating point object with `findop`. The `findop` function attempts to find an operating point that satisfies

the operating specifications and reports on its success or failure. If the search is successful, `find_op` returns state values satisfying the operating specifications.

You have several choices for operating specifications for the components of the state vector.

Assumed Operating Condition	Operating Specification
Default	Request that all state component derivatives be zero. This is a steady-state for the whole model, not just a Simscape network within the model.
Nondefault	Request any value you want independently for each state component.
Nondefault	Request that a particular state component derivative be zero. This is a steady-state condition for that state component.

Tip Making full use of Simulink Control Design software to locate operating points requires that the model be able to run without trying to start in a steady state. In the Solver Configuration blocks of your model, ensure that the **Start simulation from steady state** check boxes are cleared.

Additional Simulink Control Design Methods. You can also use the graphical user interface, through the model menu bar: **Analysis > Control Design > Linear Analysis**. This interface gives you access to state, input, and output names, structure, and initial values.

For more details on the use of operating point specification objects, related functions, and the graphical interface, see the Simulink Control Design documentation.

Using Sources to Find Operating Points Not Recommended

You can impose an operating specification on part of a Simscape model by inserting source blocks from the Simscape Foundation Library. These impose specified values of system variables in parts of the model. You can simulate and save the state vector.

However, you cannot obtain an operating point for the original system (without the source blocks) by saving the state values from the model and then removing the source blocks. In general, the number, order, and identity of state components change after adding and removing Simscape blocks in a model.

Simulink `trim` Function Not Supported with Simscape Models

The Simulink `trim` function is not supported for models containing Simscape components.

Linearizing at an Operating Point

In this section...

“What Is Linearization?” on page 3-55

“Some Linearization Methods” on page 3-58

“Linearizing a Physical Model” on page 3-59

What Is Linearization?

Determining the response of a system to small perturbations at an operating point is a critical step in system and controller design. Once you find an operating point, you can linearize the model about that operating point to explore the response and stability of the system. To find an operating point in a Simscape model, see “Finding an Operating Point” on page 3-48.

- “What Is a Linearized Model?” on page 3-55
- “Example” on page 3-56
- “Choosing a Good Operating Point for Linearization” on page 3-56
- “Linearizable and Nonlinearizable Operating Points in a Hydraulic Two-Way Valve System” on page 3-57

What Is a Linearized Model?

Near an operating point, you can express the system state \mathbf{x} , inputs \mathbf{u} , and outputs \mathbf{y} relative to that operating point in terms of $\mathbf{x} - \mathbf{x}_0$, $\mathbf{u} - \mathbf{u}_0$, and $\mathbf{y} - \mathbf{y}_0$. For convenience, shift the vectors by subtracting the operating point: $\mathbf{x} - \mathbf{x}_0 \rightarrow \mathbf{x}$, and so on.

If the system dynamics do not explicitly depend on time and the operating point is a steady state, the system response to state and input perturbations near the steady state is approximately governed by a *linear time-invariant* (LTI) state space model:

$$\frac{d\mathbf{x}}{dt} = \mathbf{A} \mathbf{x} + \mathbf{B} \mathbf{u}$$

$$\mathbf{y} = \mathbf{C} \mathbf{x} + \mathbf{D} \mathbf{u}.$$

The matrices A, B, C, D have components and structures that are independent of the simulation time. A system is stable to changes in state at an operating point if the eigenvalues of A are negative.

If the operating point is not a steady state or the system dynamics depends explicitly on time, the linearized dynamics near the operating point is more complicated. The matrices A, B, C, D are not constant and depend on the simulation time t_0 , as well as the operating point \mathbf{x}_0 and \mathbf{u}_0 [3].

Tip While you can linearize a closed system with no inputs or outputs and obtain a nonzero A matrix, obtaining a nontrivial linearized input-output model requires at least one input component in \mathbf{u} and one output component in \mathbf{y} .

Example

A pilot is flying, or simulating, an aircraft in level, constant-velocity, and constant-altitude flight relative to the ground, in a known environment. A crucial question for the aircraft pilot and designers is: will the aircraft return to the steady state if perturbed from it by a disturbance, such as a wind gust — in other words, is this steady state stable? If the operating point is unstable, the aircraft trajectory can diverge from the steady state, requiring human or automatic intervention to maintain steady flight.

Choosing a Good Operating Point for Linearization

Although steady-state and other operating points (state \mathbf{x}_0 and inputs \mathbf{u}_0) might exist for your model, that is no guarantee that such operating points are suitable for linearization. The critical question is: how good is the linearized approximation compared to the exact system dynamics?

- When perturbed slightly, a problematic operating point might exhibit strong asymmetries, with strongly nonlinear behavior when perturbed in certain ways and smoother behavior in other ways.
- Small perturbations might change discrete states in a large, discontinuous way that violates the linear approximation.

Operating points with a strongly nonlinear or discontinuous character are not suitable for linearization. You should analyze such models in full simulation, away from any discontinuities, and perturb the system by varying its inputs, parameters, and initial conditions.

Tip Check for such an unsuitable operating point by linearizing at several nearby operating points. If the results differ greatly, the operating point is strongly nonlinear or discontinuous.

Linearizable and Nonlinearizable Operating Points in a Hydraulic Two-Way Valve System

Note One step in this example (creating Bode plots) requires Control System Toolbox™ software.

The `ssc_hydraulic_system_2_way_valve` model simulates a hydraulic system made up of a valve and piston, with a controller based on the linear position of the piston. To examine the valve characteristics, open the Cylinder DA Custom block. Run the model with the Position scope open.

The piston has two natural operating points, at zero position and at the stop. Without feedback control, both operating points are steady states. Create an open-loop system by deleting the feedback signal line from Ideal Translational Motion Sensor to PS Subtract. Then run the model again to see these steady states.

- The operating point at piston position of 0 m is appropriate for linearization, because of this state's quasi-linear character. In this position, the piston can move freely forward and backward.
- The operating point at the stop (stroke at 0.3 m) is not appropriate for linearization. This state's response is quasi-linear if perturbed by moving the piston back slightly, but highly nonlinear if you attempt to push the piston beyond the stop.

Open the modified version of the model, `ssc_hydraulic_system_2_way_valve_trimlin`. This version is set up for

stability analysis with the hydraulic plant confined to its own subsystem and model-level input and output ports.

To see the simulation results, run the model first. Check the suitability of these operating points for linearization by linearizing to obtain the A , B , C , D matrices.

- 1 Linearize at zero piston position. To do this automatically, double-click the Linearize block. The result is plotted as a Bode diagram.

Then linearize at a few other positions close to zero (slightly more than 0 m).

- 2 Linearize at the piston stop, then a few other positions just short of the stop (a little less than 0.3 m).

Once you have a series of A matrices near zero and near the stop, you can define scalar-invariant metrics from A for comparison purposes. One such metric is the trace of A , the sum of its eigenvalues.

- The suitability of the operating point at zero means that the trace of A varies little over operating points near zero.
- The unsuitability of the operating point at the stop means that the trace of A varies significantly over operating points near the stop.

These characteristics indicate that this operating point is highly nonlinear and changes too rapidly for small perturbations to make a good linearization point.

Some Linearization Methods

Once you know an operating point, you have three practical methods for investigating the system response to small disturbances.

Full Simulation- or Operation-Based Perturbations

You can experiment with the system or a system simulation by making repeated, different, and slight changes to the system parameters, inputs, and initial conditions, while operating at a steady state. This method requires costly trial and error and generates uncontrolled and imprecise approximations.

Analytic Approximations to Known State Dynamics

If you know the system state dynamics and an operating point \mathbf{x}_0 and \mathbf{u}_0 in analytic form, you can apply standard approximation techniques to derive an analytic form for the A , B , C , D matrices.

Numerical Approximations to Known State Dynamics

If you have a controlled numerical approximation to your system state dynamics and operating point, you can apply standard computational techniques to generate numerical approximations to the A , B , C , D matrices.

Simulink and Simscape features provide methods for generating numerical linearized models.

Linearizing a Physical Model

Use the following methods to create numerical linearized state-space models from a model containing Simscape components.

Tip MathWorks recommends the Simulink Control Design product for linearization analysis.

- “Independent Versus Dependent States” on page 3-59
- “Linearizing with Simulink® Control Design™ Software” on page 3-60
- “Linearizing with the Simulink `linmod` and `dlinmod` Functions” on page 3-61
- “Linearizing with Simulink Linearization Blocks” on page 3-63

Independent Versus Dependent States

An important difference from normal Simulink models is that the states in a physical network are not independent in general, because some states have dependencies on other states through constraints.

- The independent states are a subset of system variables and consist of Simscape dynamic variables and Simulink states.

- The dependent states consist of Simscape algebraic variables and dependent (constrained) dynamic variables.

For more information on Simscape dynamic and algebraic variables, see “How Simscape Simulation Works” on page 3-5.

The complete, unreduced LTI A , B , C , D matrices have the following structure.

- The A matrix, of size n_states by n_states , is all zeros except for a submatrix of size n_ind by n_ind , where n_ind is the number of independent states.
- The B matrix, of size n_states by n_inputs , is all zeros except for a submatrix of size n_ind by n_inputs .
- The C matrix, of size $n_outputs$ by n_states , is all zeros except for a submatrix of size $n_outputs$ by n_ind .
- The D matrix, of size $n_outputs$ by n_inputs , can be nonzeros everywhere.

Obtaining the Independent Subset of States. A minimal linearized solution uses only an independent subset of system states. From the matrices A , B , C , D , you can obtain a minimal input-output linearized model with:

- The `minreal` and `sminreal` functions from Control System Toolbox software
- Automatically with the Simulink Control Design approach

Linearizing with Simulink Control Design Software

Note The techniques described in this section require the Simulink Control Design product.

You must use the features of this product on the Simulink lines in your model, not directly on Simscape physical network lines or blocks.

This approach requires that you start with an operating point object saved from trimming the model to an operating specification, as explained in “Using Simulink® Control Design™ Techniques to Find Operating Points” on page 3-52.

To linearize a model with an operating point object, use the `linearize` function, customizing where necessary. The resulting state-space object contains the matrices A , B , C , D .

Additional Simulink Control Design Methods. You can also use the graphical user interface, through the model menu bar: **Analysis > Control Design > Linear Analysis**. For more details on linearization, operating points and state-space objects, related functions, and the graphical interface, see the Simulink Control Design documentation.

Linearizing with the Simulink `linmod` and `dlinmod` Functions

You have several ways that you can use the Simulink functions `linmod` and `dlinmod`, and the linearization results can differ depending on the method chosen. To use these functions, you do not have to open the model, just have the model file on your MATLAB path.

For more information about Simulink linearization, see “Linearizing Models” in the Simulink documentation.

Tip If your model has continuous states, use `linmod`. (Continuous states are the Simscape default.) If your model has mixed continuous and discrete states, or purely discrete states, use `dlinmod`.

Linearizing a model with the local solver enabled (in the Solver Configuration block) is not supported.

Linearizing with Default State and Input. You can call `linmod` without specifying state or input. Enter `linmod('modelName')` at the command line.

With this form of `linmod`, Simulink linearization solves for consistent initial conditions in the same way it does on the first step of any simulation. Any initial conditions, such as initial offset from equilibrium for a spring, are set as if the simulation were starting from the initial time.

`linmod` allows you to change the time of externally specified signals (but not the internal system dynamics) from the default. For this and more details, see the function reference page.

Linearizing with the Steady-State Solver at an Initial Steady State.

You can linearize at an operating point found by the Simscape steady-state solver:

- 1 Open one or more Solver Configuration blocks in your model.
- 2 Select the **Start simulation from steady state** check box for the physical networks that you want to linearize.
- 3 Close the Solver Configuration dialog boxes and save the modified model.
- 4 Enter `linmod('modelName')` at the command line.

`linmod` linearizes at the first step of simulation. In this case, the initial state is also an operating point, a steady state.

For more about setting up the steady-state solver, see the Solver Configuration block reference page. For more details on its use, see “Using the Simscape Steady-State Solver” on page 3-51.

Linearizing with Specified State and Input – Ensuring Consistency of States.

You can call `linmod` and specify state and input. Enter `linmod('modelName',x0,u0)` at the command line. The extra arguments specify, respectively, the steady state \mathbf{x}_0 and inputs \mathbf{u}_0 for linearizing the simulation. When you specify a state to `linmod`, ensure that it is self-consistent, within solver tolerance.

With this form of `linmod`, Simulink linearization does not solve for initial conditions. Because not all states in the model have to be independent, it is possible, though erroneous, to provide `linmod` with an inconsistent state to linearize about.

If you specify a state that is not self-consistent (within solver tolerance), the Simscape solver issues a warning at the command line when you attempt linearization. The Simscape solver then attempts to make the specified \mathbf{x}_0 consistent by changing some of its components, possibly by large amounts.

Tip You most easily ensure a self-consistent state by taking the state from some simulated time. For example, by selecting the **States** check box on the **Data Import/Export** pane of the model Configuration Parameters dialog box, you can capture a time series of state values in a simulation run.

Linearizing with Simulink Linearization Blocks

You can generate linearized state-space models from your Simscape model by adding a Timed-Based Linearization or Trigger-Based Linearization block to the model and simulating. These blocks combine time-based simulation, up to specified times or internal trigger points, with state-based linearization at those times or trigger points.

For complete details about these blocks, see their respective block reference pages.

Note If your model contains PS Constant Delay or PS Variable Delay blocks, or custom blocks utilizing the `delay` operator in the Simscape language, MathWorks recommends that you linearize the model by using the Timed-Based Linearization or Trigger-Based Linearization block and simulating the model for a time period longer than the specified delay time.

Linearize an Electronic Circuit

In this section...

“About the Nonlinear Bipolar Transistor Circuit” on page 3-64
“Finding Operating Points in a Transistor Circuit with the Simscape Solver” on page 3-70
“Linearizing a Transistor Circuit with Simulink and Related Software” on page 3-71

About the Nonlinear Bipolar Transistor Circuit

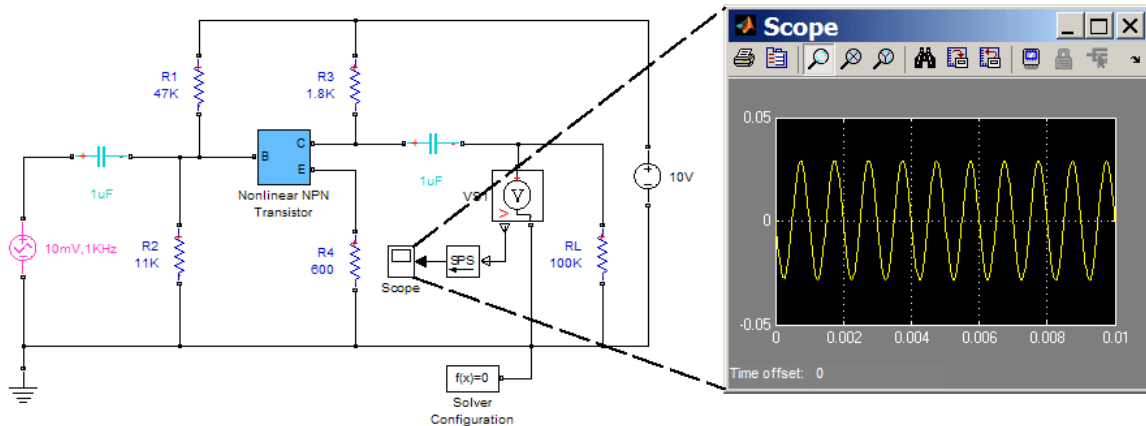
The `ssc_bipolar_nonlinear` model contains a nonlinear bipolar junction transistor circuit, equivalent to an Ebers-Moll circuit [2]. A modified version of the model, `ssc_bipolar_nonlinear_trimlin`, is ready for linearization when you first open it and forms the basis for the following operating point and linearization examples.

- “Simulating the Basic Model Starting at Steady and Nonsteady States” on page 3-64
- “Changing the Steady State and Amplification in the Basic Model” on page 3-66
- “Opening and Simulating a Modified Model Prepared for Linearization Analysis” on page 3-67
- “Approaching Steady State Through Long-Time Transient Simulation” on page 3-69

Simulating the Basic Model Starting at Steady and Nonsteady States

The transistor acts between base-emitter voltage (ports B and E) and collector current (port C). The circuit is driven by an oscillating AC voltage source of 10 mV and 1 kHz, with a constant bias DC voltage of 10 V. The Nonlinear NPN Transistor is the essential component of the circuit and represents a bipolar transistor that amplifies the driving AC voltage. The Scope block displays the voltage coming off the collector.

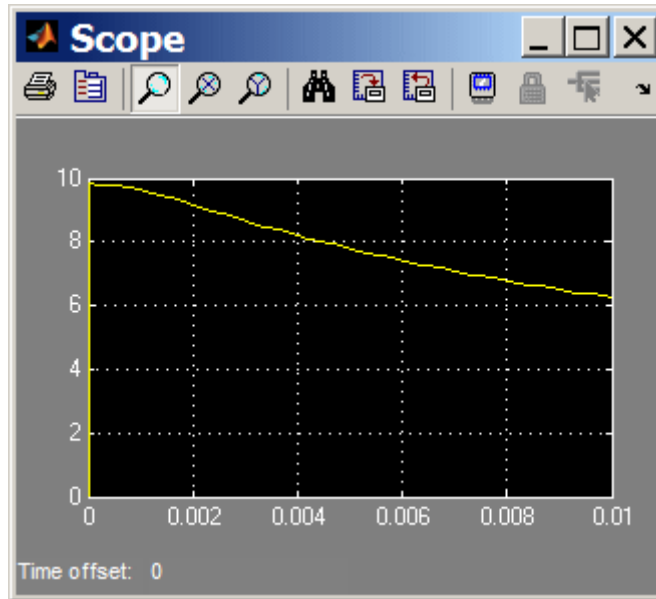
Simulate `ssc_bipolar_nonlinear` with the Scope open to see the basic circuit behavior. The output transistor junction capacitances are set to be initially consistent with the bias subcircuit. The output is a steady sinusoid with zero average, its amplitude amplified by a factor 3 by the transistor and bias subcircuit.



Nonlinear Bipolar Transistor

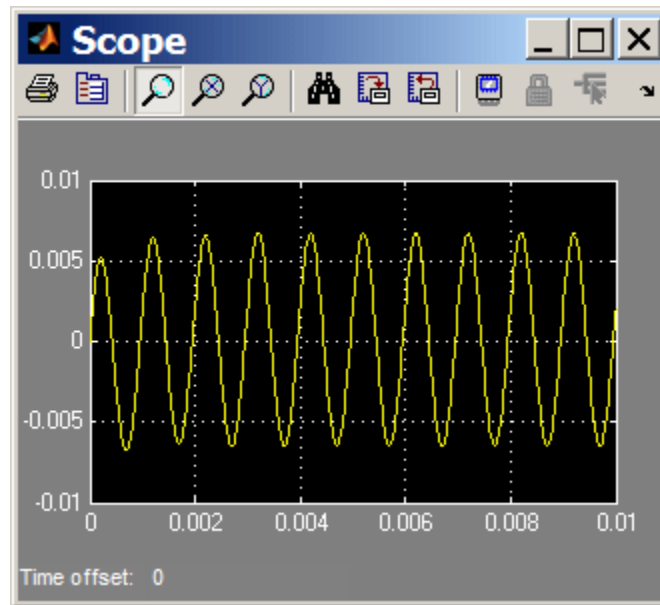
This model shows an implementation of a nonlinear bipolar transistor model based on the Ebers-Moll equivalent circuit. R1 and R2 set the nominal operating point, and the small signal gain is approximately set by the ratio R3/R4. The 1μF decoupling capacitors have been chosen to present negligible impedance at 1KHz.

To see the circuit relax from a nonsteady initial state, open the Solver Configuration block and clear the **Start simulation from steady state** check box. With the Scope open, simulate again. In this case, the output voltage starts at zero because the transistor junction capacitances start with zero charge. Then change the model back to starting at a steady state.



Changing the Steady State and Amplification in the Basic Model

The steady-state collector voltage is controlled mainly by the R2 and R4 resistance values, while the amplification of the driving AC voltage is controlled mainly by the R1 and R3 resistance values. Experiment with changing these resistances to change the steady state and near-steady state behavior of the circuit. For example, change R1 from 47 to 15 kOhms. The collector voltage is now no longer amplified relative to the 10 mV AC source, but attenuated.



If you simulate without starting from a steady state, changing these resistance values also modifies the transient behavior.

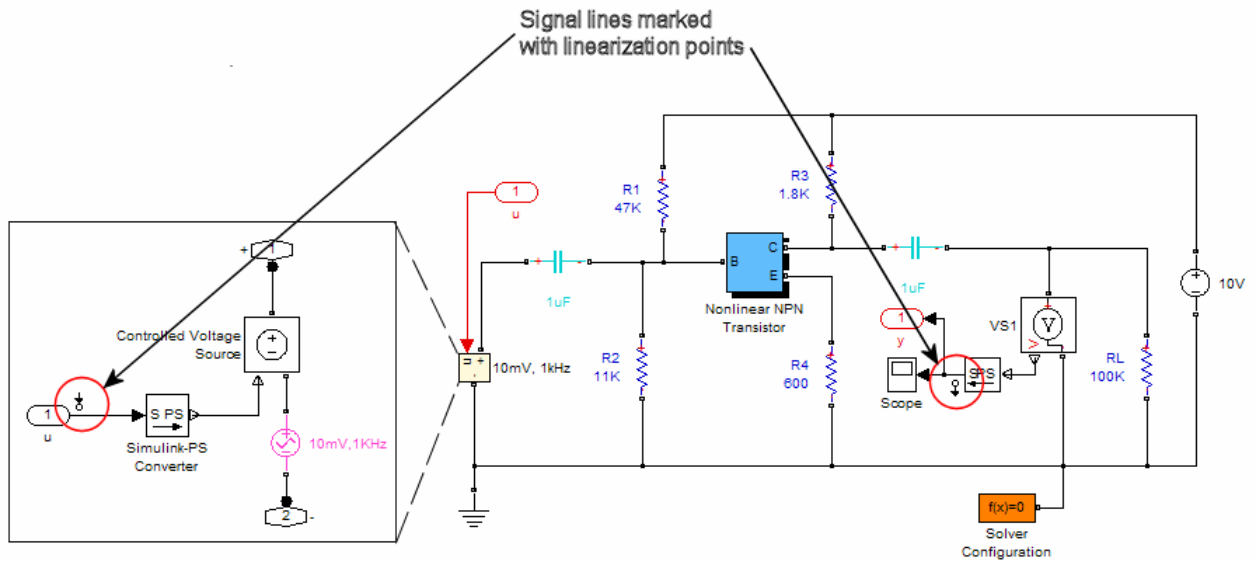
Opening and Simulating a Modified Model Prepared for Linearization Analysis

To obtain a nontrivial linearized input-output model from the Simulink model, you must specify model-level inputs and outputs. The modified `ssc_bipolar_nonlinear_trimlin` meets this requirement in two ways, depending on how you linearize.

- Simulink Control Design software requires that you specify input and output signal lines with linearization points. `ssc_bipolar_nonlinear_trimlin` has such linearization points specified. The specified lines must be Simulink signal lines, not Simscape physical connection lines. For more information on using Simulink Control Design software for trimming and linearization, see documentation for this product.

- Simulink requires top- or model-level input and output ports for linearization with `linmod`. `ssc_bipolar_nonlinear_trimlin` has such ports, marked `u` and `y`.

1 Open `ssc_bipolar_nonlinear_trimlin`. Open the AC voltage source subsystem. The AC Voltage source is now combined with an input port.



2 Right-click the two signal lines indicated, each of which is marked with a linearization point symbol. A context menu is displayed. Select **Linear Analysis Points**. A submenu is displayed, listing the different ways you can mark signal lines for control design analysis.

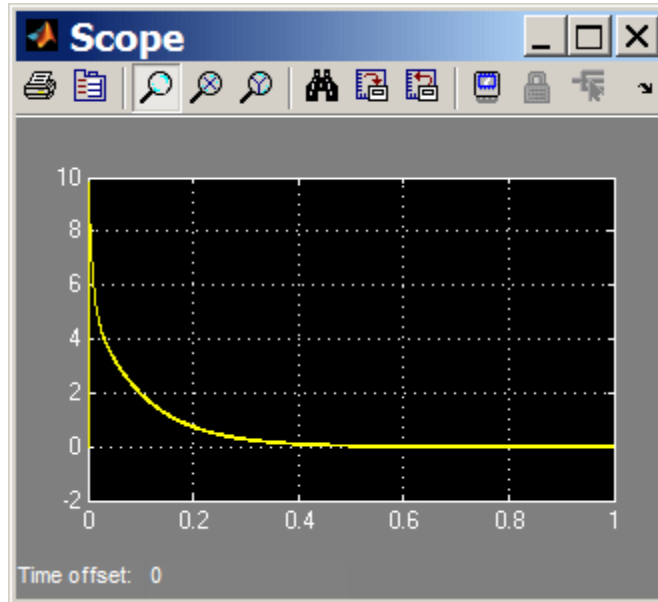
- One signal line is inside the source subsystem and runs from the model-level input port `u`. In the submenu, **Input Point** is selected, indicating that this signal line is designated as an input.
- The other signal line leads to the Scope and runs to a model-level output port `y`. In the submenu, **Output Point** is selected, indicating that this signal line is designated as an output.

3 Open and run this modified version of the model. The simulated behavior of this version is the same as the original model.

Approaching Steady State Through Long-Time Transient Simulation

You can get a more comprehensive understanding of the circuit behavior and how it approaches the steady state by changing the simulation and Scope parameters in `ssc_bipolar_nonlinear_trimlin`.

- 1 Open Solver Configuration and clear the **Start simulation from steady state** check box. Click **OK**.
- 2 Open the Scope. From the Scope menu bar, open **Parameters**.
- 3 On the **General** pane, change **Time range** to 1.0. On the **Data History** pane, clear the **Limit data points to last** check box. Click **OK**.
- 4 Change the simulation time to 1.
- 5 Start the simulation. The circuit starts from its initial nonsteady state, and the transistor collector voltage approaches and eventually settles into steady sinusoidal oscillation.



Finding Operating Points in a Transistor Circuit with the Simscape Solver

Note This example uses the Simscape steady-state solver. Save the results of this example (in particular, the initial steady state vector \mathbf{x}_0) for later use in “Linearizing a Transistor Circuit with Simulink and Related Software” on page 3-71. You should work through the later section after you follow this example.

For general information about the Simscape steady-state solver, see “Finding Operating Points in Physical Models” on page 3-50.

Simulating and Saving the Steady State as an Operating Point

Simulate the modified `ssc_bipolar_nonlinear_trimlin`, with the Simscape steady-state solver enabled, as it is when you first open the model. This steady-state solution is an operating point suitable for linearization. With the simulation starting from this steady state, you can characterize this operating point by one of the following:

- The initial time $t = 0$
- The initial state vector \mathbf{x}_0 . To capture state values during simulation:
 - 1 Open the model Configuration Parameters dialog box.
 - 2 Select the **Data Import/Export** pane.
 - 3 Select the **States** check box under **Save to workspace**.

At the first time step, the state vector values represent the initial state.

- 4 Click **OK**.

After you simulate with an initial steady state, capture the initial state vector by entering:

```
x0 = xout(1,:);
```

Linearizing a Transistor Circuit with Simulink and Related Software

Note This example uses `linmod` and time-based linearization. Before attempting it, work through the preceding example, “Finding Operating Points in a Transistor Circuit with the Simscape Solver” on page 3-70. For this example, use the results of that example, including the steady state value, x_0 .

The nonlinear bipolar transistor model, `ssc_bipolar_nonlinear_trimlin`, introduced in the preceding example, has input and output ports that guarantee a nontrivial input-output LTI model after linearization.

For general information about linearizing with Simulink, see “Linearizing a Physical Model” on page 3-59.

- “Counting Model States” on page 3-71
- “Linearizing the Model at an Initial Steady State with `linmod`” on page 3-72
- “Linearizing the Model at a Specified Operating Point with `linmod`” on page 3-72
- “Linearizing the Model at Multiple Simulation Times with a Linearization Block” on page 3-72
- “Suitability of the Steady State for Linearization — Nonlinearity” on page 3-73
- “Analyzing the Linearization Results — Finding the Minimum Realization” on page 3-74

Counting Model States

The state vector x of `ssc_bipolar_nonlinear_trimlin` contains 16 components. The full model has one input and one output.

Thus, the LTI state-space models derived from linearization have the following matrix sizes: A is 16-by-16; B is 16-by-1; C is 1-by-16; and D is 1-by-1.

Linearizing the Model at an Initial Steady State with `linmod`

First, linearize the model at an initial steady state. In the Solver Configuration block, make sure that you have selected the **Start simulation from steady state** check box. To capture the LTI matrices, enter:

```
[a0,b0,c0,d0] = linmod('ssc_bipolar_nonlinear_trimlin');
```

The state has 16 components. The `linmod` function alone, without an output argument, generates a structure with states, inputs, and outputs, as well as the LTI model.

Linearizing after a Change to Steady State Characteristics. Obtain a different steady state and LTI by changing resistance R1 from 47 to 15 kOhms. Linearize again:

```
[a0_R1,b0_R1,c0_R1,d0_R1] = linmod('ssc_bipolar_nonlinear_trimlin');
```

Then change R1 back to 47 kOhms.

Linearizing the Model at a Specified Operating Point with `linmod`

Second, linearize the model with the initial steady state captured as a state vector, `x0`. Clear the **Start simulation from steady state** check box in the Solver Configuration block. Then enter:

```
u = 0;  
[a1,b1,c1,d1] = linmod('ssc_bipolar_nonlinear_trimlin',x0,u);
```

Verify that the two matrix sets, `a0`, `b0`, `c0`, `d0` and `a1`, `b1`, `c1`, `d1`, do not differ significantly. The operating point `x0` was obtained from the Simscape steady-state solver. If you saved this state as a vector in your workspace, you do not need to invoke the steady-state solver again to solve for it.

Linearizing the Model at Multiple Simulation Times with a Linearization Block

In the Solver Configuration block, reselect the **Start simulation from steady state** check box.

Next, obtain a series of state vectors to linearize at multiple simulation times using the Timed-Based Linearization block. Sample a few times uniformly across one AC cycle near the steady state. The cycle period is 1 millisecond.

- 1 From the Simulink library, insert a Timed-Based Linearization block into your model.
- 2 Open the linearization block. Into the **Linearization time** field, enter [0 0.25e-3 0.5e-3 0.75e-3]. Click **OK**.

This series of times samples the simulation at four, evenly spaced points over one AC cycle.

- 3 In the Configuration Parameters dialog box, in the **Data Import/Export** pane, make sure that you have selected **Time**, **States**, and **Output**.
- 4 Simulate the model. Check your workspace for `tout`, `xout`, `yout`, and the structure called `ssc_bipolar_nonlinear_trimlin_Timed_Based_Linearization`.

The structure has a component for each of the four linearization times. Verify these times by entering:

```
ssc_bipolar_nonlinear_trimlin_Timed_Based_Linearization(1).OperPoint.t
ssc_bipolar_nonlinear_trimlin_Timed_Based_Linearization(2).OperPoint.t
```

and so on. Extract the LTI matrices for these four linearization times:

```
A1 = ssc_bipolar_nonlinear_trimlin_Timed_Based_Linearization(1).a;
B1 = ssc_bipolar_nonlinear_trimlin_Timed_Based_Linearization(1).b;
C1 = ssc_bipolar_nonlinear_trimlin_Timed_Based_Linearization(1).c;
D1 = ssc_bipolar_nonlinear_trimlin_Timed_Based_Linearization(1).d;
```

Create A2, B2, C2, D2 from the second linearization time, and so on.

Suitability of the Steady State for Linearization – Nonlinearity

Of the four LTI models created by time-based linearization, one comes from the steady state itself and the other three from nearby states. If the steady state is a good operating point for linearization, these LTI models do not differ greatly from one another.

Verify this suitability by direct comparison of the components of A1, A2, A3, and A4. You can also derive and compare invariant scalar metrics from the A matrices, such as the traces, to make sure that the LTI models vary only slightly near the steady state.

Accuracy of Linear Approximation to Nonlinear Behavior. With different circuit parameters, you can make this steady state less suitable for linearization. The transistor collector current response to the base-emitter voltage is exponential:

$$I_C = I_S[\exp(V_{BE}/V_T) - 1] ,$$

where I_C is the collector current, I_S is the temperature-dependent saturation current for the transistor, V_{BE} is the base-emitter voltage, and V_T is the normalized voltage = $k_B T/q = 25.3$ mV at room temperature ($T = 20$ °C = 293 K; k_B = Boltzmann's constant, q = electron charge) [2].

Adjusting the circuit characteristics can enhance this nonlinear character and degrade the accuracy of a linear approximation to the circuit's response near the steady state.

Analyzing the Linearization Results – Finding the Minimum Realization

Note To work through this section, you must have the Control System Toolbox product.

Not all the states of the LTI models derived in this example are independent. Confirm this by calculating the determinant of one of the A matrices; for example, $\det(a0)$. These determinants vanish and imply one or more zero eigenvalues.

Reducing and Analyzing the First Steady State. Before you can use one of these LTI models, reduce the LTI matrices to a minimal realization. Obtain a minimal realization with the `minreal` function:

```
[a2,b2,c2,d2] = minreal(a0,b0,c0,d0);  
12 states removed.
```

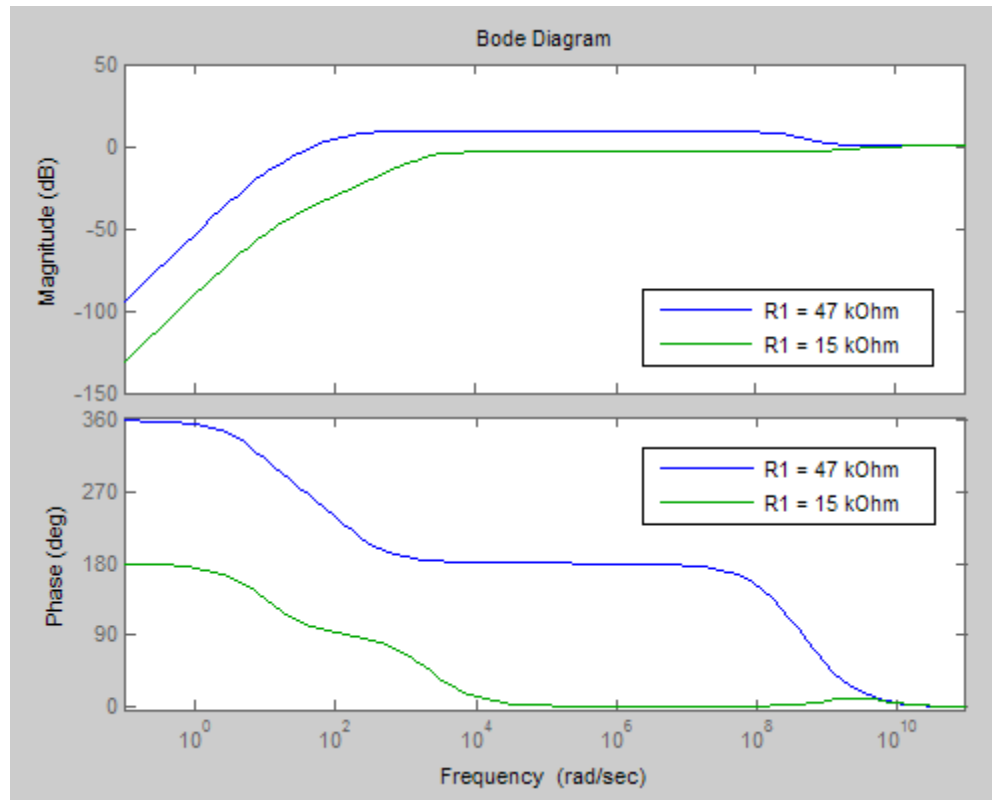
Extracting the minimal realization eliminates 12 dependent states from the LTI model, leaving four independent states. Analyze the control characteristics of the reduced a_2 , b_2 , c_2 , d_2 LTI model with a Bode plot:

```
bode(a2,b2,c2,d2) % Creates first Bode plot
```

Reducing and Analyzing a Second Steady State. The circuit with R_1 changed from 47 to 15 kOhm has a different steady state and response. Reduce this LTI model to a minimal realization as well and analyze its control characteristics:

```
[a2_R1,b2_R1,c2_R1,d2_R1] = minreal(a0_R1,b0_R1,c0_R1,d0_R1); % 12 states removed.  
hold on % Keeps first Bode plot open  
bode(a2_R1,b2_R1,c2_R1,d2_R1) % Superposes second Bode plot on first
```

The second LTI model is reduced to four independent states.



Limitations

In this section...

“Sample Time and Solver Restrictions” on page 3-77

“Algebraic Loops” on page 3-77

“Restricted Simulink Tools” on page 3-78

“Unsupported Simulink Tools” on page 3-80

“Simulink Tools Not Compatible with Simscape Blocks” on page 3-80

“Code Generation” on page 3-81

Sample Time and Solver Restrictions

The default sample times of Simscape blocks are continuous. You cannot simulate Simscape blocks with discrete solvers using the default sample times.

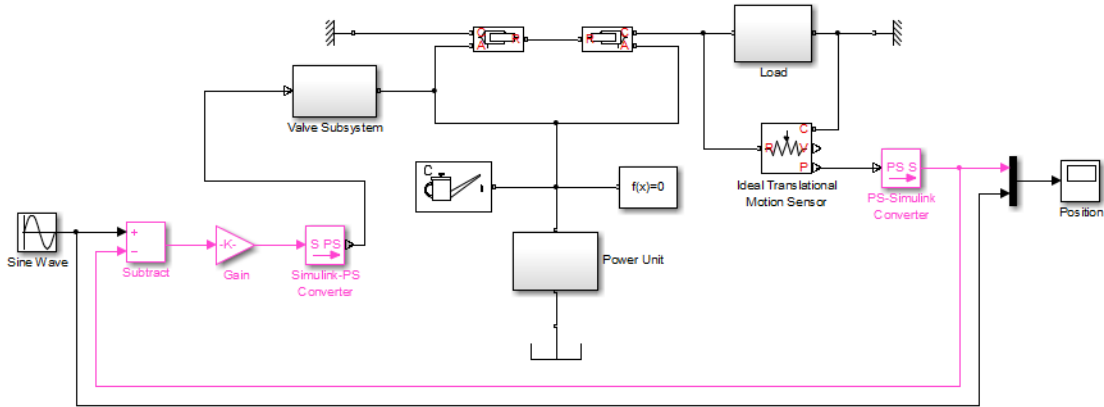
If you switch to a local solver in the Solver Configuration block, the states of the associated physical network become discrete. If there are no continuous Simulink or Simscape states anywhere in a model, you are free to use a discrete solver to simulate the model.

You cannot override the sample time of a nonvirtual subsystem containing Simscape blocks.

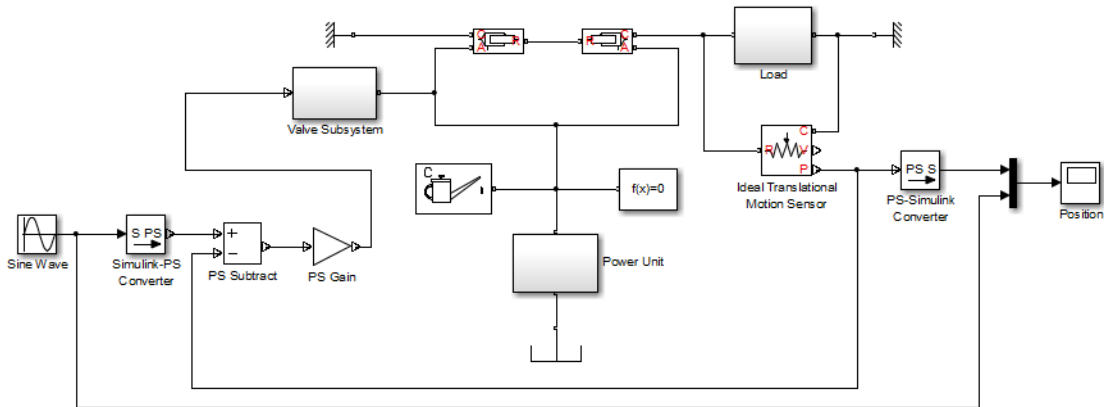
Algebraic Loops

A Simscape physical network should not exist within a Simulink algebraic loop. This means that you should not directly connect an output of a PS-Simulink Converter block to an input of a Simulink-PS Converter block of the same physical network.

For example, the following model contains a direct feedthrough between the PS-Simulink Converter block and the Simulink-PS Converter block (highlighted in magenta). To avoid the algebraic loop, you can insert a Transfer Function block anywhere along the highlighted loop.



A better way to avoid an algebraic loop without introducing additional dynamics is shown in the modified model below.



Restricted Simulink Tools

Certain Simulink tools are restricted for use with Simscape software:

- You can use the Simulink `set_param` and `get_param` commands to set or get Simscape block parameters, if the parameters correspond to fields in the block dialog box. MathWorks does not recommend that you use these commands to find or change any other block parameters.

If you make changes to block parameters at the command line, run your model first before saving it. Otherwise, you might save invalid block parameters. Any block parameter changes that you make with `set_param` are not validated unless you run the model.

- Simscape blocks accept `Simulink.Parameter` objects as parameter values in `get_param` and `set_param`, within the restrictions specified here.
- Enabled subsystems can contain Simscape blocks. Always set the **States when enabling** parameter in the Enable dialog to **held** for the subsystem's Enable port.

Setting **States when enabling** to `reset` is not supported and can lead to fatal simulation errors.

- You can place Simscape blocks within nonvirtual subsystems that support continuous states. Nonvirtual subsystems that support continuous states include Enabled subsystems and Atomic subsystems. However, physical connections and physical signals must not cross nonvirtual boundaries. When placing Simscape blocks in a nonvirtual subsystem, make sure to place all blocks belonging to a given Physical Network in the same nonvirtual subsystem.
- Simulink configurable subsystems work with Simscape blocks only if all of the block choices have consistent port signatures.
- For Iterator, Function-Call, Triggered, and While Iterator nonvirtual subsystems cannot contain Simscape blocks.
- An atomic subsystem with a user-specified (noninherited) sample time cannot contain Simscape blocks.
- When using `SimState` to save and restore simulations of models, you cannot make any changes to the Simscape blocks in the model between the time at which you save the `SimState` and the time at which you restore the simulation using the `SimState`.

This is an extension of the Simulink limitation prohibiting structural changes to the model between these two points in time (see “Limitations of the `SimState`”). Changes to Simscape block parameters can cause equation changes and result in changes to the state representation. Therefore, modifying parameters of Simscape blocks between saving and restoring the `SimState` is not allowed.

- Linearization with the Simulink `linmod` function or with equivalent Simulink Control Design functions and graphical interfaces is not supported with Simscape models if you use local solvers.
- Model referencing is supported, with some restrictions:
 - All Physical connection lines must be contained within the referenced model. Such lines cannot cross the boundary of the referenced model subsystem in the referencing model.
 - The referencing model and the referenced model must use the same solver.

Unsupported Simulink Tools

Certain Simulink tools and features do not work with Simscape software:

- The Simulink Profiler tool does not work with Simscape models.
- Exporting a model to a format used by an earlier version (**File > Export Model to > Previous Version**) is not supported for models containing Simscape blocks.
- Physical signals and physical connection lines between conserving ports are different from Simulink signals. Therefore, the Signal and Scope Manager tool and the signal label functionality are not supported.

Simulink Tools Not Compatible with Simscape Blocks

Some Simulink tools and features do not work with Simscape blocks:

- Execution order tags do not appear on Simscape blocks.
- Simscape blocks do not invoke user-defined callbacks.
- You cannot set breakpoints on Simscape blocks.
- Reusable subsystems cannot contain Simscape blocks.
- You cannot use the Simulink Fixed-Point Tool with Simscape blocks.
- The Report Generator reports Simscape block properties incompletely.

Code Generation

Code generation is supported for Simscape physical modeling software and its family of add-on products. However, there are restrictions on code generated from Simscape models.

- Code reuse is not supported.
- Encapsulated C++ code generation is not supported.
- Tunable parameters are not supported.
- Run-time parameter inlining ignores global exceptions.
- Simulation of Simscape models on fixed-point processors is not supported.
- Block diagnostics in error messages are not supported. This means that if you get an error message from simulating generated code, it does not contain a list of blocks involved.
- Conversion of models or subsystems containing Simscape blocks to S-functions is not supported.
- Software-in-the-loop (SIL) simulation is not supported.

“Code Generation” on page 3-35 describes Simscape code generation features. “Restricted Simulink Tools” on page 3-78 describes limitations on model referencing.

There are variations and exceptions as well in the code generation features of the add-on products based on Simscape platform. For details, see documentation for individual add-on products.

Code Generation and Fixed-Step Solvers

Most code generation options for Simscape models require the use of fixed-step Simulink solvers. This table summarizes the available solver choices, depending on how you generate code.

Code Generation Option	Solver Choices
Accelerator mode Rapid Accelerator mode	Variable-step or fixed-step
Simulink Coder software: RSim Target*	Variable-step or fixed-step
Simulink Coder software: Targets other than RSim	Fixed-step only

* For the RSim Target, Simscape software supports only the Simulink solver module. In the model Configuration Parameters dialog box, see the **Code Generation: RSim Target: Solver selection** menu. The default is automatic selection, which might fail to choose the Simulink solver module.

References

- [1] Moler, C. B., *Numerical Computing with MATLAB*, Philadelphia, Society for Industrial and Applied Mathematics, 2004, chapter 7
- [2] Horowitz, P., and Hill, W., *The Art of Electronics*, 2nd Ed., Cambridge, Cambridge University Press, 1989, chapter 2
- [3] Brogan, W. L., *Modern Control Theory*, 2nd Ed., Englewood Cliffs, New Jersey, Prentice-Hall, 1985

Data Logging

- “About Simulation Data Logging” on page 4-2
- “How to Log Simulation Data” on page 4-3
- “Log and Plot Simulation Data” on page 4-7
- “Log Simulation Statistics” on page 4-13

About Simulation Data Logging

In this section...
“Suggested Workflows” on page 4-2
“Limitations” on page 4-2

Suggested Workflows

You can log simulation data to the workspace for debugging and verification. Data logging lets you analyze how internal block variables change with time during simulation. For example, you may want to see that the pressure in a hydraulic cylinder is above some minimum value, or compare it against the pump pressure. If you log simulation data to the workspace, you can later query, plot, and analyze it without rerunning the simulation.

Simulation data logging can also replace connecting sensors and scopes to track simulation data. These blocks increase the model complexity and slow down simulation. The “Log and Plot Simulation Data” on page 4-7 shows how you can log and plot simulation data instead of adding sensors to your model. It also shows how you can print the complete logging tree for a model, and plot simulation results for a selected variable.

For additional information, see the reference pages for the classes `simscape.logging.Node`, `simscape.logging.Series`, and their associated methods.

Limitations

Simulation data logging is not supported for:

- Model reference
- Generated code
- Accelerator mode
- Rapid Accelerator mode

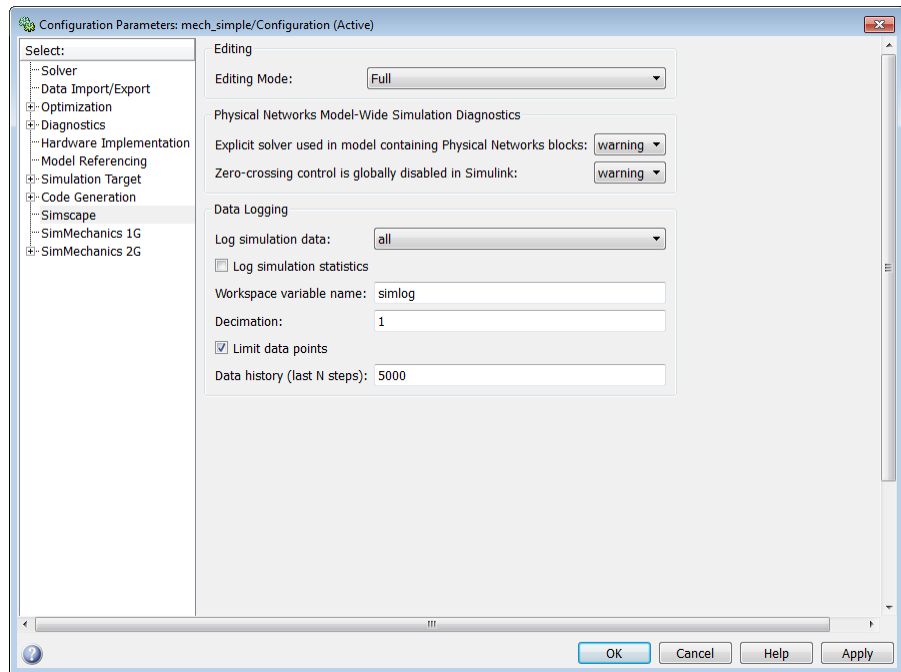
How to Log Simulation Data

In this section...
“How to Enable Data Logging” on page 4-3
“Data Logging Options” on page 4-4

How to Enable Data Logging

By default, simulation data is not logged. To turn on the data logging for a model, use the **Log simulation data** configuration parameter.

- 1** In the model window, from the top menu bar, select **Simulation > Model Configuration Parameters**. The Configuration Parameters dialog box opens.
- 2** In the Configuration Parameters dialog box, in the left pane, select **Simscape**. The right pane displays the **Log simulation data** option, which is set to none, by default.
- 3** From the drop-down list, select **all**, then click **OK**.



- 4 Simulate the model. This creates a workspace variable named `simlog` (as specified by the **Workspace variable name** parameter), which contains the simulation data.

For information on how to access and use the data stored in this variable, see “Log and Plot Simulation Data” on page 4-7.

Data Logging Options

When you set the **Log simulation data** configuration parameter to `all`, other options in the Data Logging group box become available.

- **Log simulation statistics** — Select this checkbox if you want to access and analyze information on zero crossings during simulation. By default, this checkbox is not selected and the zero-crossing data is not logged. For more information on using this checkbox, see “Log Simulation Statistics” on page 4-13.

- **Workspace variable name** — Specifies the name of the workspace variable that stores the simulation data. Subsequent simulations overwrite the data in the simulation log variable. If you want to compare data from two models or two simulation runs, use different names for the respective log variables. The default variable name is `simlog`.
- **Decimation** — Use this parameter to limit the number of data points saved, by outputting data points for every n th time step, where n is the decimation factor. The default is 1, which means that all points are logged. Specifying a different value results in the first step, and every n th step thereafter, being logged. For example, specifying 2 logs data points for every other time step, while specifying 10 logs data points for just one in ten steps.
- **Limit data points** — Use this checkbox in conjunction with the **Data history (last N steps)** parameter to limit the number of data points saved. The checkbox is selected by default. If you clear it, the simulation log variable contains the data points for the whole simulation, at the price of slower simulation speed and heavier memory consumption.
- **Data history (last N steps)** — Specify the number of simulation steps to limit the number of data points output to the workspace. The simulation log variable contains the data points corresponding to the last N steps of the simulation, where N is the value that you specify for the **Data history (last N steps)** parameter. You have to select the **Limit data points** checkbox to make this parameter available. The default value logs simulation data for the last 5000 steps. You can specify any other positive integer number. If the simulation contains fewer steps than the number specified, the simulation log variable contains the data points for the whole simulation.

Saving data to the workspace can slow down the simulation and consume memory. To avoid this, you can use either the **Decimation** parameter, or **Limit data points** in conjunction with **Data history (last N steps)**, or both methods, to limit the number of data points saved. The two methods work independently from each other and can be used separately or together. For example, if you specify a decimation factor of 2 and keep the default value of 5000 for the **Data history (last N steps)** parameter, your workspace variable will contain downsampled data from the last 10000 time steps in the simulation.

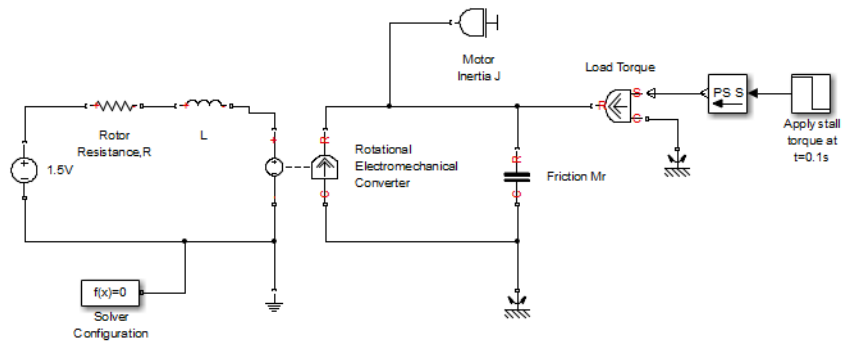
Note The **Output options** parameter, on the **Data Import/Export** pane of the Configuration Parameters dialog box, also affects which data points are logged. For more information, see “Data Import/Export Pane” in the Simulink documentation.

After changing your data logging preferences, rerun the simulation to generate a new data log.

Log and Plot Simulation Data

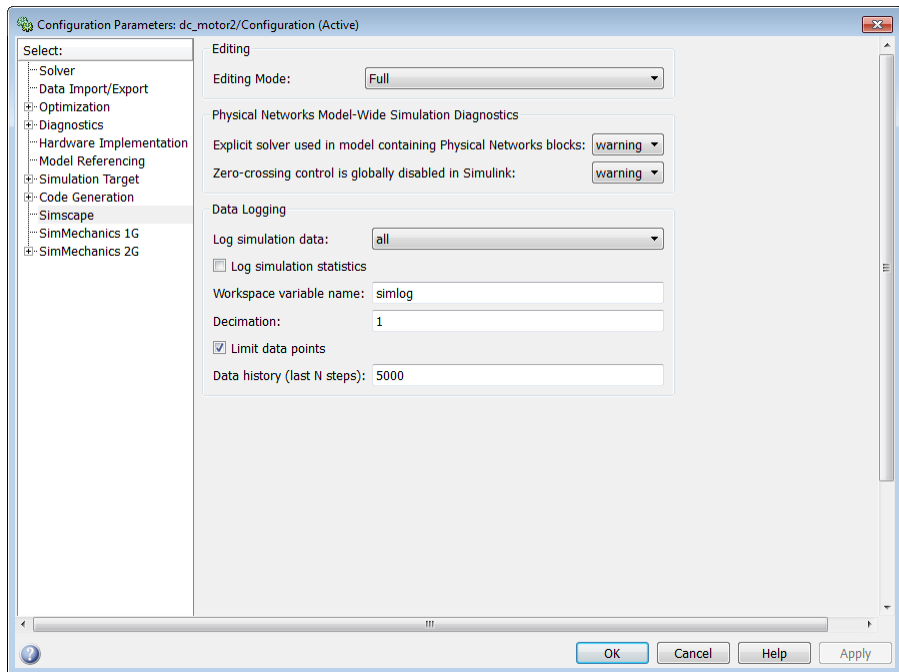
This example shows how you can log and plot simulation data instead of adding sensors to your model.

The model shown represents a permanent magnet DC motor.



This model is very similar to the Permanent Magnet DC Motor example, but, unlike the example model, it does not include the Ideal Rotational Motion Sensor and the Current Sensor blocks, along with the respective PS-Simulink Converter blocks and scopes. For a detailed description of the Permanent Magnet DC Motor example, see “Evaluating Performance of a DC Motor”.

- 1 Build the model, as shown in the preceding illustration.
- 2 To enable data logging, open the Configuration Parameters dialog box, in the left pane, select **Simscape**, then set the **Log simulation data** parameter to **all** and click **OK**.



- 3** Simulate the model. This creates a workspace variable named `simlog` (as specified by the **Workspace variable name** parameter), which contains the simulation data.
- 4** The `simlog` variable has the same hierarchy as the model. To see the whole variable structure, at the command prompt, type:

```
simlog.print
```

This command prints the whole data tree.

```
dc_motor2
+-Electrical_Reference2
| +-V
| | +-v
| +-i
+-Friction_Mr
| +-C
| | +-w
```

```
| +-R
| | +-w
| +-t
| +-w
+-L
| +-i
| +-i_L
| +-n
| | +-v
| +-p
| | +-v
| +-v
+-Load_Torque
| +-C
| | +-w
| +-R
| | +-w
| +-S
| +-t
| +-w
+-Mechanical_Rotational_Reference
| +-W
| | +-w
| +-t
+-Mechanical_Rotational_Reference1
| +-W
| | +-w
| +-t
+-Motor_Inertia_J
| +-I
| | +-w
| +-t
+-Rotational_Electromechanical_Converter
| +-C
| | +-w
| +-R
| | +-w
| +-i
| +-n
| | +-v
```

```
| +-p
| | +-v
| +-t
| +-v
| +-w
+-Rotor_Resistance_R
| +-i
| +-n
| | +-v
| +-p
| | +-v
| +-v
+-Simulink_PS_Converter
+-x1_5V
  +-i
  +-n
  | +-v
  +-p
  | +-v
  +-v
```

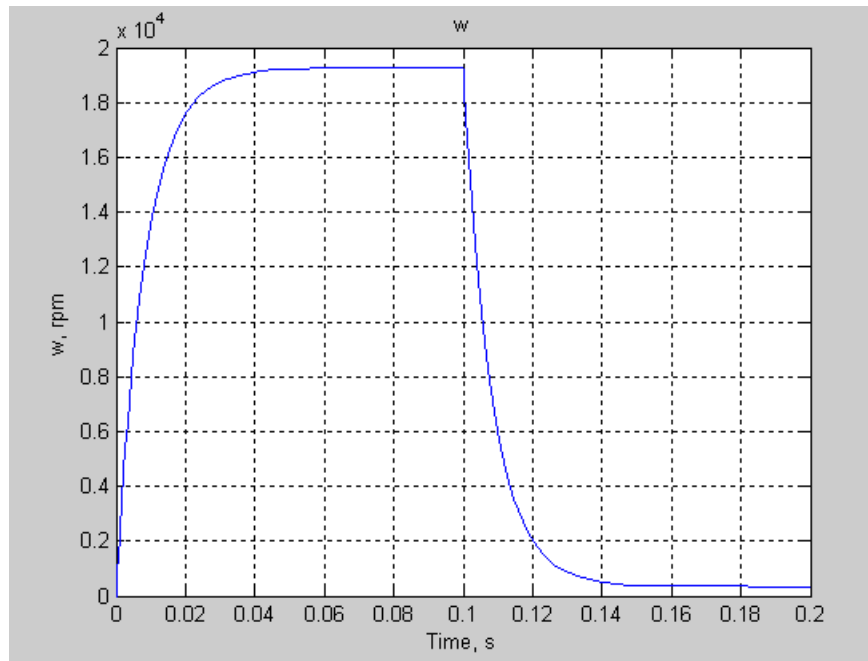
- 5** Every node that represents an Across, Through, or internal block variable contains series data. To get to the series, you have to specify the complete path to it through the tree, starting with the top-level variable name. For example, to get a handle on the series representing the angular velocity of the motor, type:

```
s1 = simlog.Rotational_Electromechanical_Converter.R.w.series;
```

From here, you can access the values and time vectors for the series and analyze them.

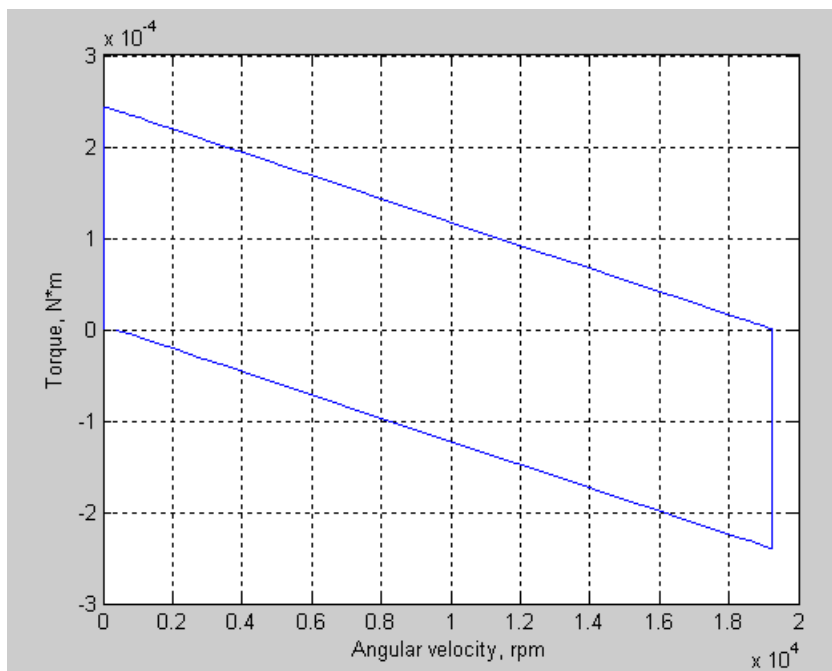
- 6** You do not have to isolate series data to plot its values against time, or against another series. For example, to see how the motor speed (in revolutions per minute) changes with time, type:

```
plot(simlog.Rotational_Electromechanical_Converter.R.w, 'units', 'rpm')
```



- 7** Compare this figure to the RPM scope display in the Permanent Magnet DC Motor example. The results are exactly the same.
- 8** To plot the motor torque against its angular velocity, in rpm, and add descriptive axis names, type:

```
plotxy(simlog.Rotational_Electromechanical_Converter.R.w, simlog.Motor_Inertia_J.t, ...  
       'xunit', 'rpm', 'xname', 'Angular velocity', 'yname', 'Torque')
```



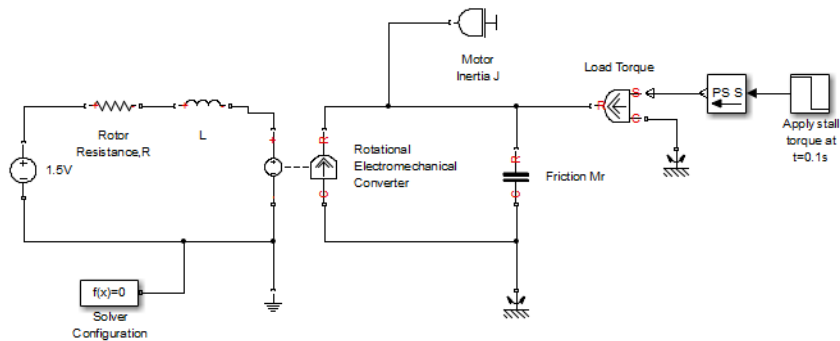
For more information on plotting logged simulation data, see the `simscape.logging.plot` and `simscape.logging.plotxy` reference pages.

Note Plotting simulation data for a high-level node and its children can generate a large number of plots. By default, the plots are not docked in the desktop, which results in a multitude of separate figure windows. To avoid this inconvenience, you can issue a command to make figures automatically dock in the desktop. For more information, see “Docking Figures Automatically” in the MATLAB documentation.

Log Simulation Statistics

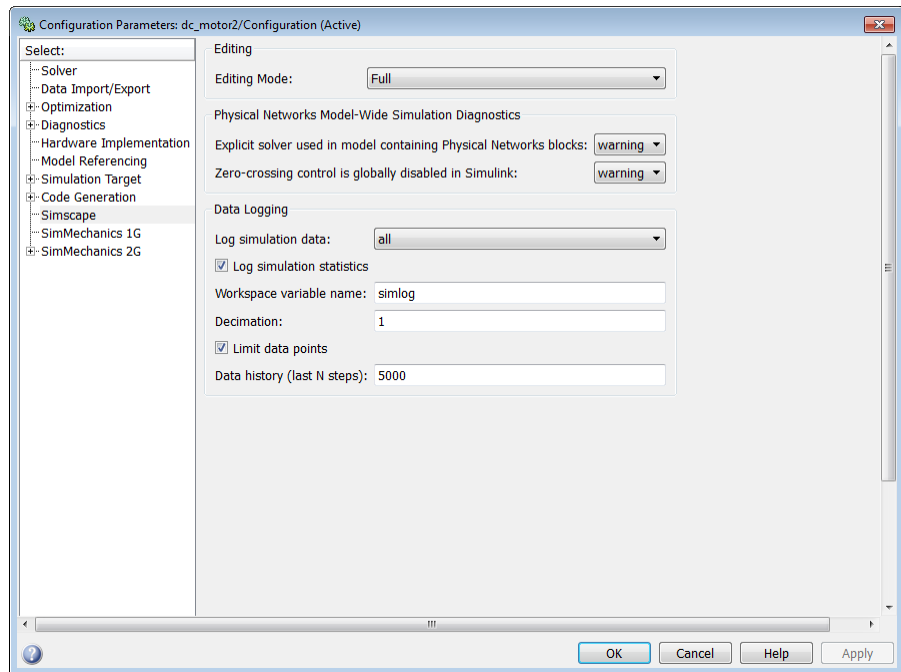
This example shows how you can access and analyze information on zero crossings during simulation. By default, the zero-crossing data is not logged. If you select the **Log simulation statistics** checkbox, the simulation log variable contains an additional `SimulationStatistics` node for each block that can produce zero crossings, at the price of slower simulation speed and heavier memory consumption.

The model shown represents a permanent magnet DC motor.



This model is the same as the one used in the “Log and Plot Simulation Data” on page 4-7 example.

- 1 Build the model, as shown in the preceding illustration.
- 2 To enable data logging, open the Configuration Parameters dialog box, in the left pane, select **Simscape**, then set the **Log simulation data** parameter to **all**, select the **Log simulation statistics** checkbox, and click **OK**.



3 Simulate the model. This creates a workspace variable named `simlog` (as specified by the **Workspace variable name** parameter), which contains the simulation data. Because you selected the **Log simulation statistics** checkbox, the workspace variable contains additional nodes that represent zero-crossing data.

4 The `simlog` variable has the same hierarchy as the model. To see the whole variable structure, at the command prompt, type:

```
simlog.print
```

This command prints the whole data tree.

```
dc_motor2
+-Electrical_Reference2
| +-V
| | +-v
| +-i
+-Friction_Mr
```

```
| +-C
| | +-w
| +-R
| | +-w
| +-SimulationStatistics
| | +-zc_0
| | | +-crossings
| | | +-values
| | +-zc_1
| | | +-crossings
| | | +-values
| | +-zc_2
| | | +-crossings
| | | +-values
| +-t
| +-w
+-L
| +-i
| +-i_L
| +-n
| | +-v
| +-p
| | +-v
| +-v
+-Load_Torque
| +-C
| | +-w
| +-R
| | +-w
| +-S
| +-t
| +-w
+-Mechanical_Rotational_Reference
| +-W
| | +-w
| +-t
+-Mechanical_Rotational_Reference1
| +-W
| | +-w
| +-t
```

```
+Motor_Inertia_J
| +-I
| | +-w
| +-t
+Rotational_Electromechanical_Converter
| +-C
| | +-w
| +-R
| | +-w
| +-i
| +-n
| | +-v
| +-p
| | +-v
| +-t
| +-v
| +-w
+Rotor_ResistanceR
| +-i
| +-n
| | +-v
| +-p
| | +-v
| +-v
+-x1_5V
  +-i
  +-n
  | +-v
  +-p
  | +-v
  +-v
```

- 5** If you compare this tree to the one used in the “Log and Plot Simulation Data” on page 4-7 example, you can see that under the `Friction_Mr` node there is now an additional node called `SimulationStatistics`. The rest of the tree is unchanged. This means that `Friction Mr` is the only block in the model that can generate zero-crossings during simulation.

- 6 You can access and analyze this data similar to other data logged to workspace during simulation. For more information, see `simscape.logging.Node` and `simscape.logging.Series` reference pages.

Model Statistics

- “Simscape Model Statistics” on page 5-2
- “1-D Physical System Statistics” on page 5-4
- “3-D Multibody System Statistics” on page 5-6
- “View Model Statistics” on page 5-9

Simscape Model Statistics

Viewing Simscape model statistics is a good way to evaluate the model prior to simulation. Model statistics provide feedback on the model complexity, so that you can make informed choices about whether you want to simulate the model in its current configuration or make changes to it. This approach helps you achieve the desired simulation performance and goals.

Unlike other derived data (such as data logging or simulation statistics), which is generated during simulation, model statistics is compile-time data that is generated before the model is simulated. When you generate model statistics, the model must be in a compilable state, that is, it must satisfy the requirements described in “Model Validation” on page 3-7.

Use model statistics as part of the iterative model building process. For example, after you make a change to the model, you can view model statistics to answer the following questions:

- Did the change increase the number of variables?
- Does the model have redundant constraints or have I resolved them?
- How many potential zero-crossing signals does the model have?

The Statistics Viewer analysis tool is available for models containing Simscape blocks and blocks from add-on products. Depending on the types of blocks in the model, the analysis can produce one or both statistics categories:

- **1-D Physical System** — This node represents aggregate statistics generated from all physical networks that are associated with blocks from Simscape, SimDriveline™, SimHydraulics, SimElectronics®, and SimPowerSystems™ Third Generation libraries.
- **3-D Multibody System** — This node represents aggregate statistics generated from all physical networks that are associated with blocks from SimMechanics™ Second Generation library.

Each statistic is generated separately from each topologically distinct physical network of these blocks and then aggregated to appear as a single statistic in the Statistics Viewer.

**Related
Examples**

- “View Model Statistics” on page 5-9

Concepts

- “1-D Physical System Statistics” on page 5-4
- “3-D Multibody System Statistics” on page 5-6

1-D Physical System Statistics

This node represents aggregate statistics generated from all physical networks that are associated with blocks from Simscape, SimDriveline, SimHydraulics, SimElectronics, and SimPowerSystems Third Generation libraries.

Each statistic is generated separately from each topologically distinct physical network of these blocks and then aggregated to appear as a single statistic.

The individual statistics are:

- **Number of variables** — This statistic represents the number of variables associated with all 1-D physical systems in the model. Variables are categorized further as continuous and discrete variables.

This statistic represents the number of variables in the system after variable elimination. If a system is truly input-output with no dynamics, it is possible to completely eliminate all variables and, in that case, the number of variables is zero.

- **Number of continuous variables** — This statistic represents the number of continuous variables associated with all 1-D physical systems in the model. Continuous variables are those variables whose values vary continuously with time, although some continuous variables can change values discontinuously after events. Continuous variables are categorized further as algebraic and differential variables. This statistic represents the number of continuous variables in the system after variable elimination.
- **Number of algebraic variables** — This statistic represents the number of algebraic variables associated with all 1-D physical systems in the model. Algebraic variables are continuous system variables whose time derivative does not appear in any system equations. These variables appear in algebraic equations but add no dynamics, and this typically occurs in physical systems due to conservation laws, such as conservation of mass and energy. This statistic represents the number of algebraic variables in the model after variable elimination.
- **Number of differential variables** — This statistic represents the number of differential variables associated with all 1-D physical systems in the model. Differential variables are continuous variables whose time derivative appears in one or more system equations. These variables add dynamics to the system and require the solver to use numerical integration

to compute their values. This statistic represents the number of differential variables in the model after variable elimination.

- **Number of discrete variables** — This statistic represents the number of discrete, or event, variables associated with all 1-D physical systems in the model. Discrete variables are those variables whose values can change only at specific events. Discrete variables are categorized further as integer-valued and real-valued discrete variables.
- **Number of integer-valued variables** — This statistic represents the number of integer-valued discrete variables associated with all 1-D physical systems in the model. Integer-valued discrete variables are system variables that take on integer values only and can change their values only at specific events, such as sample time hits. These variables are typically generated from blocks that are sampled and run at specified sample times.
- **Number of real-valued variables** — This statistic represents the number of real-valued discrete variables associated with all 1-D physical systems in the model. Real-valued discrete variables are system variables that take on real values and can change their values only at specific events.

If you select a local solver in the Solver Configuration block, then all continuous variables associated with that system are discretized and represented as real-valued discrete variables.

- **Number of zero-crossing signals** — This statistic represents the number of scalar signals that are monitored by the Simulink zero-crossing detection algorithm. Zero-crossing signals are scalar functions of states, inputs, and time whose crossing zero indicates discontinuity in the system. These signals are typically generated from operators and functions that contain discontinuities, such as comparison operators, `abs`, `sqrt` functions, and so on. Times when these signals cross zero are reported as zero-crossing events. During simulation it is possible for none of these signals to produce a zero-crossing event or for one or more of these signals to have multiple zero-crossing events.

3-D Multibody System Statistics

This node represents aggregate statistics generated from all physical networks that are associated with blocks from SimMechanics Second Generation library.

Each statistic is generated separately from each topologically distinct physical network of these blocks and then aggregated to appear as a single statistic.

The individual statistics are:

- **Number of rigidly connected components (excluding ground)** — This statistic provides the number of rigid components present in a mechanical system. Rigid components are subsets of rigidly connected blocks that represent rigid bodies or rigid frame networks in a model. These subsets generally include blocks from the Body Elements library as well as Rigid Transform blocks.

Rigid connections within a rigid component can include Rigid Transform blocks but not Weld Joint blocks. Rigid Transform blocks provide rigid connections between blocks in the same rigid component. Weld Joint blocks, like all joint blocks, provide connections between blocks in different rigid components.

This statistic excludes from the count any rigid component that rigidly connects to the World Frame block.

- **Number of joints (total)** — This statistic provides the total number of joints present in a mechanical system. This number equals the sum of three types of joints: explicit tree, cut, and implicit 6-DOF joints. For more information, see the statistic descriptions for these joints.

The kinematic graph provides a practical means to understand the topology of a model. This graph is a connected, undirected diagram in which each vertex corresponds to a rigid component and each edge corresponds to a joint. The total number of joints equals the total number of edges present in this graph.

The kinematic tree is a spanning tree of the kinematic graph in which each closed loop is opened by cutting one of its edges. If the kinematic graph contains no closed loops, it is identical to the kinematic tree.

- **Number of explicit tree joints** — This statistic provides the number of joints in the kinematic tree of a mechanical system that correspond to explicit joint blocks. Each tree joint corresponds to an edge in the kinematic tree. The number of explicit tree joints excludes joints cut from the kinematic graph to generate the kinematic tree.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **Number of implicit 6-DOF tree joints** — This statistic provides the number of 6-DOF joints in the kinematic tree of a mechanical system that do not correspond to explicit joint blocks. SimMechanics adds implicit 6-DOF joints when the kinematic graph of a model is not fully connected. These implicit joints connect previously disconnected portions of the graph to the ground body, adding the edges required to fully connect the graph. Implicit joints are always tree joints and do not create loops.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **Number of cut joints** — This statistic provides the number of joints that are cut from the kinematic graph of a mechanical system to generate the associated kinematic tree. The number of cut joints equals the number of closed loops present in the kinematic graph.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **Number of constraints** — This statistic provides the total number of constraint blocks in a mechanical system.
- **Number of tree degrees of freedom** — This statistic provides the total number of degrees of freedom in the kinematic tree of a mechanical system. This number equals the sum of all degrees of freedom that the tree joints provide. It excludes degrees of freedom associated with cut joints.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **Number of position constraint equations (total)** — This statistic provides the number of scalar equations that impose position constraints on a mechanical system. Constraint equations arise from two types of blocks: Constraints and Joints. Joint blocks contribute constraint equations only if the joints are cut in the kinematic tree. The number of position constraint

equations that a cut joint contributes equals six minus the number of degrees of freedom that joint provides.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

- **Number of position constraint equations (non-redundant)** — This statistic provides the number of unique position constraint equations associated with a model. This number is smaller than or equal to the total number of position constraint equations. The difference between the two is the number of redundant position constraint equations, which are satisfied whenever the unique position constraint equations are satisfied. SimMechanics attempts to remove redundant equations to improve simulation performance.

- **Number of mechanism degrees of freedom (minimum)** — This statistic provides a lower bound on the number of degrees of freedom in a mechanical system. It equals the difference between the number of tree degrees of freedom and the number of non-redundant position constraint equations. The actual number of degrees of freedom can exceed this lower bound if SimMechanics fails to detect a position constraint equation.

Some position constraint equations become redundant only in certain configurations. If an equation becomes redundant during simulation, the actual number of degrees of freedom in a model can change. However, that number must still equal or exceed the lower bound that this statistic provides.

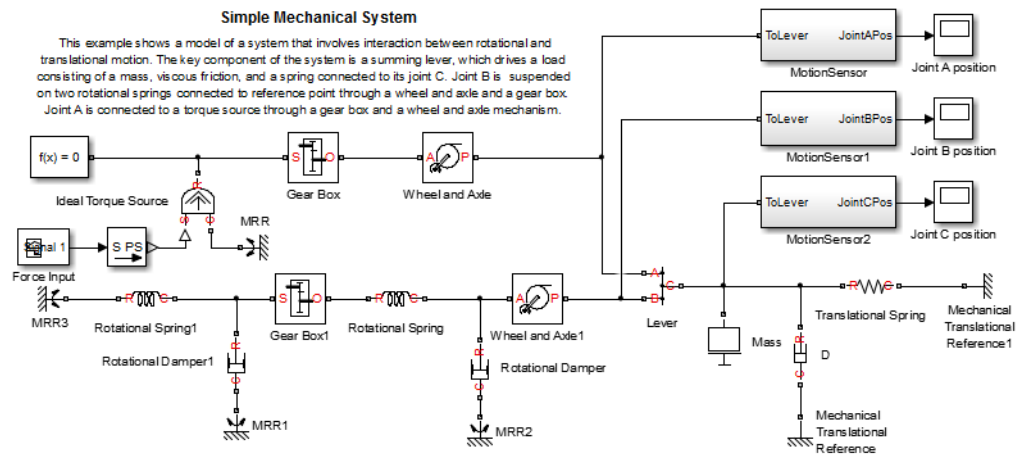
- **State vector size** — This statistic provides the number of scalar values in the state vector of a mechanical system.
- **Average kinematic loop length** — This statistic provides the average number of edges—or, equivalently, vertices—in the closed loops of a kinematic graph. The average number is taken over all loops in the graph. If the graph has no kinematic loops, this number equals zero.

For more information about kinematic graphs and trees, see the statistic description for **Number of joints (total)**.

View Model Statistics


This example shows how you can use model statistics to determine the effect of a change on model complexity.

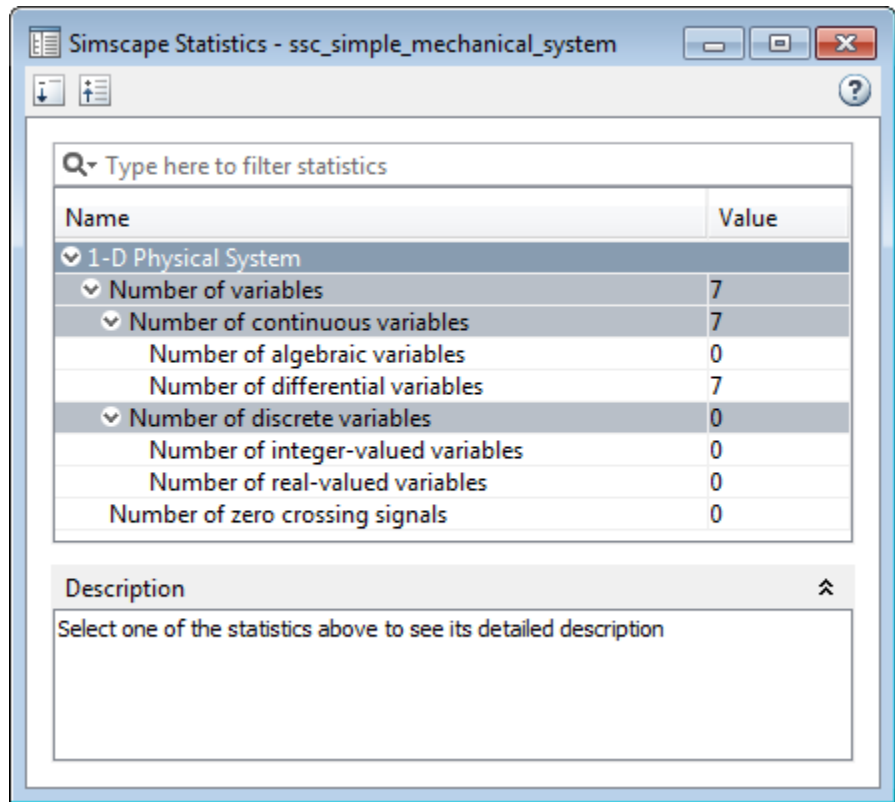
- 1 Open the Simple Mechanical System example model.



- 2 To view model statistics, in the top menu bar of the model window, select **Analysis > Simscape > Statistics Viewer**.

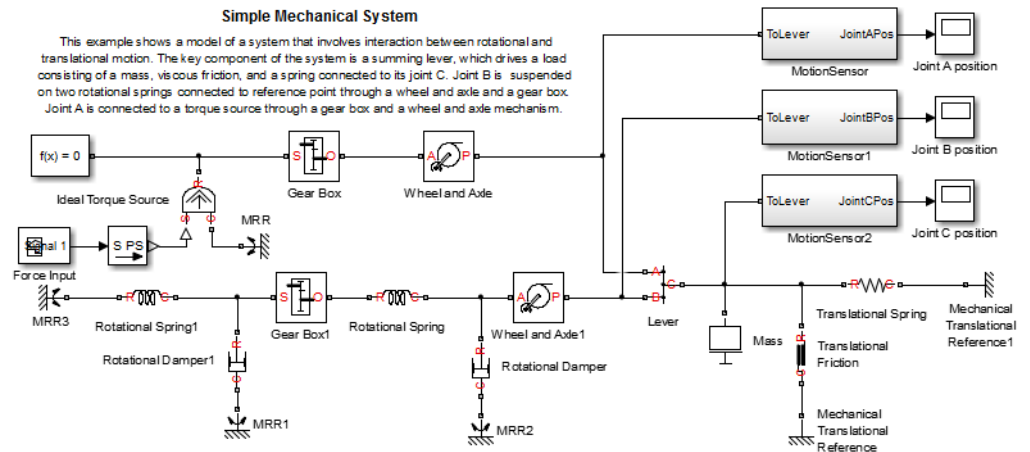
The Simscape Statistics window opens, displaying the name of the model and an overview of the models statistics in a collapsed state.

- 3 Click  to expand all nodes.

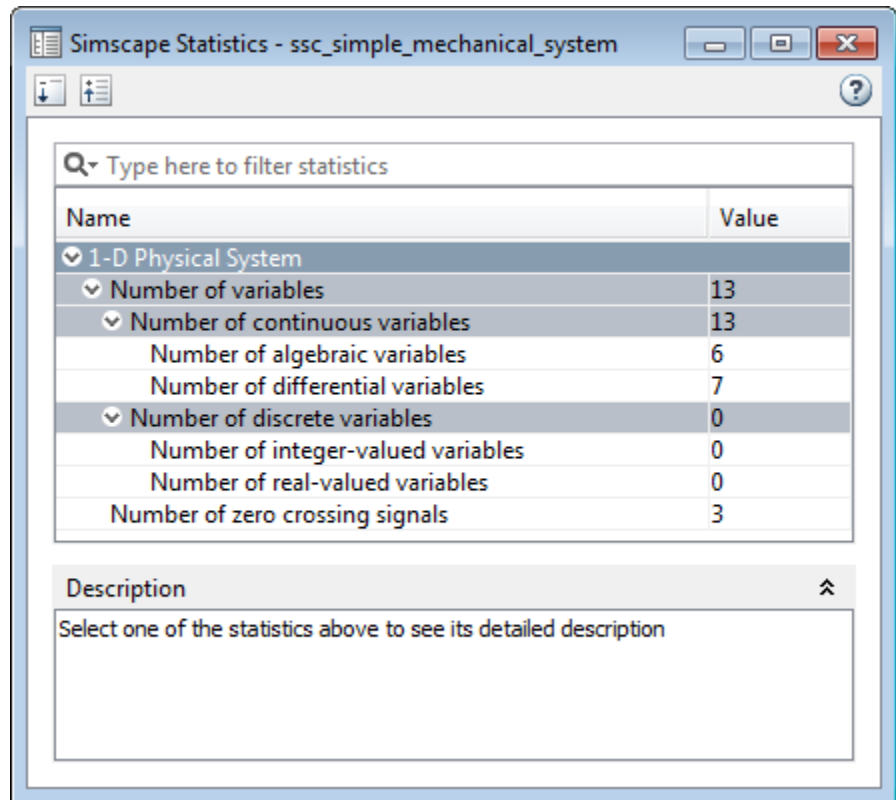


You can see that the model contains seven continuous differential variables, no algebraic variables, no discrete variables, and no zero-crossing signals.

- 4** Replace the Translational Damper block D in the model diagram with a Translational Friction block, as shown in the following figure.



- 5 Select **Analysis > Simscape > Statistics Viewer** to refresh the model statistics.



Name	Value
1-D Physical System	
Number of variables	13
Number of continuous variables	13
Number of algebraic variables	6
Number of differential variables	7
Number of discrete variables	0
Number of integer-valued variables	0
Number of real-valued variables	0
Number of zero crossing signals	3

Description ⬆

Select one of the statistics above to see its detailed description

The revised model contains six algebraic variables, seven differential variables, and three zero-crossing signals. This happened because you replaced a linear block (Translational Damper) with a nonlinear one (Translational Friction). Therefore the linear optimization that the solver initially performed on the model no longer applies.

Physical Units

- “How to Work with Physical Units” on page 6-2
- “Unit Definitions” on page 6-4
- “How to Specify Units in Block Dialogs” on page 6-10
- “Thermal Unit Conversions” on page 6-12
- “Angular Units” on page 6-16
- “Units for Angular Velocity and Frequency” on page 6-17

How to Work with Physical Units

Unlike Simulink signals, which are essentially unitless, physical signals can have units associated with them. You specify the units along with the parameter values in the block dialogs, and Simscape unit manager performs the necessary unit conversion operations when solving a physical network. Simscape blocks support standard measurement systems. The default block units are meter-kilogram-second or MKS (SI).

Simscape software comes with a library of standard units, and you can define additional units as needed (see “Unit Definitions” on page 6-4). You can use these units in your block diagrams:

- To specify the units of an input physical signal, type a unit name, or a mathematical expression with unit names, in the **Input signal unit** field of the Simulink-PS Converter block dialog. You can also select a unit from a drop-down list, which is prepopulated with some common input units. Signal units that you specify in a Simulink-PS Converter block must match the input type expected by the Simscape block connected to it. For example, when you provide the input signal for an Ideal Angular Velocity Source block, specify angular velocity units, such as rad/s or rpm, in the Simulink-PS Converter block, or leave it unitless. If you leave the block unitless, with the **Input signal unit** parameter set to 1, then the physical signal units are inferred from the destination block.
- Simscape block dialogs have drop-down combo boxes of units next to a parameter value, letting you either select a unit from the drop-down list, or type a unit name (or a mathematical expression with unit names) directly into the box. These drop-down lists are automatically populated by those units that are commensurate with the unit of the parameter, based on the current list of unit definitions. For example, if a parameter is specified, by default, with the units of meters per second, m/s, the drop-down list of units contains commensurate units, such as mm/s, in/s, fps (feet per second), fpm (feet per minute), and so on, including any other linear velocity units currently defined in your unit registry.
- To specify the units of an output physical signal, type a unit name, or a mathematical expression with unit names, in the **Output signal unit** field of the PS-Simulink Converter block dialog. You can also select a unit from a drop-down list, which is prepopulated with some common output units. The system compares the units you specified with the actual units

of the input physical signal coming into the converter block and applies a gain equal to the conversion factor before outputting the Simulink signal. The default value is 1, which means that the unit is not specified. If you do not specify a unit, or if the unit matches the actual units of the input physical signal, no gain is applied.

For more information, see “How to Specify Units in Block Dialogs” on page 6-10.

Note Currently, the blocks in the Physical Signals library (such as PS Add, PS Gain, and so on) ignore the physical unit of the input signal and just perform calculations on the value. The output signals of the Physical Signals library blocks are unitless.

Unit Definitions

Simscape unit names are defined in the `pm_units.m` file, which is shipped with the product. You can open this file to see how the physical units are defined in the product, and also as an example when adding your own units. This file is located in the directory `matlabroot\toolbox\physmod\common\units\mli\m`.

Default registered units and their abbreviations are listed in the following table. Use the `pm_getunits` command to get an up-to-date list of units currently defined in your unit registry. Use the `pm_adddimension` and `pm_addunit` commands to define additional units.

Physical Unit Abbreviations Defined by Default in the Simscape Unit Registry

Quantity	Abbreviation	Unit
Acceleration	gee	Earth gravitational acceleration (9.80665 m/s ²)
Amount of substance	mol	Mole
Angle	rad deg rev	Radian Degree Revolution
Angular velocity	rpm	Revolutions/minute
Capacitance	F pF nF uF	Farad Picofarad Nanofarad Microfarad
Charge	c	Coulomb

Physical Unit Abbreviations Defined by Default in the Simscape Unit Registry (Continued)

Quantity	Abbreviation	Unit
Conductance	S	Siemens
	nS	Nanosiemens
	uS	Microsiemens
	mS	Millisiemens
Current	A	Ampere
	pA	Picoampere
	nA	Nanoampere
	uA	Microampere
	mA	Milliampere
	kA	Kiloampere
Energy	J	Joule
	Btu	British thermal unit
	eV	Electronvolt
Flow rate	lpm	Liter/minute
	gpm	Gallon/minute
Force	N	Newton
	dyn	Dyne
	lbf	Pound-force
	mN	Millinewton
Frequency	Hz	Hertz
	kHz	Kilohertz
	MHz	Megahertz
	GHz	Gigahertz

Physical Unit Abbreviations Defined by Default in the Simscape Unit Registry (Continued)

Quantity	Abbreviation	Unit
Inductance	H	Henry
	uH	Microhenry
	mH	Millihenry
Length	m	Meter
	cm	Centimeter
	mm	Millimeter
	km	Kilometer
	um	Micrometer
	in	Inch
	ft	Foot
	mi	Mile
yd	Yard	
Magnetic flux	Wb	Weber
Magnetic flux density	T	Tesla
	G	Gauss
Mass	kg	Kilogram
	g	Gram
	mg	Milligram
	lbm	Pound mass
	oz	Ounce
	slug	Slug

Physical Unit Abbreviations Defined by Default in the Simscape Unit Registry (Continued)

Quantity	Abbreviation	Unit
Pressure	Pa	Pascal
	kPa	Kilopascal
	MPa	Megapascal
	GPa	Gigapascal
	bar	Bar
	kbar	Kilobar
	atm	Atmosphere
Power	psi	Pound/inch ²
	W	Watt
	uW	Microwatt
	mW	Milliwatt
	kW	Kilowatt
	MW	Megawatt
	HP	Horsepower
Resistance	Ohm	Ohm
	kOhm	Kiloohm
	MOhm	Megaohm
	GOhm	Gigaohm
Temperature	K	Kelvin
	C	Celsius
	Fh	Fahrenheit
	R	Rankine

Physical Unit Abbreviations Defined by Default in the Simscape Unit Registry (Continued)

Quantity	Abbreviation	Unit
Time	s	Second
	min	Minute
	hr	Hour
	ms	Millisecond
	us	Microsecond
	ns	Nanosecond
Velocity	mph	Miles/hour
	fpm	Feet/minute
	fps	Feet/second
Viscosity absolute	Poise	Poise
	cP	Centipoise
	reyn	Reyn
Viscosity kinematic	St	Stokes
	cSt	Centistokes
	Newt	Newt
Volume	l	Liter
	gal	US liquid gallon
	igal	Imperial (UK) gallon
Voltage	V	Volt
	mV	Millivolt
	kV	Kilovolt

Note This table lists the unit abbreviations defined in the product. For information on how to use the abbreviations above, or mathematical expressions with these abbreviations, to specify units for the parameter values in the block dialogs, see “How to Specify Units in Block Dialogs” on page 6-10.

How to Specify Units in Block Dialogs

Simscape block dialogs have drop-down combo boxes for units next to a parameter value. For example, in the Constant Volume Chamber block dialog box, the drop-down list for the **Chamber volume** parameter contains l, gal, in³, ft³, mm³, cm³, m³, and km³, and the drop-down list for the **Initial pressure** parameter contains Pa, bar, psi, and atm.

You can either select a unit from the drop-down list, or type a commensurate unit name (or a mathematical expression with unit names) directly into the unit combo box of the block dialog. You can use the abbreviations for the units defined in your registry, or any valid mathematical expressions with these abbreviations. For example, you can specify torque in newton-meters (N*m) or pound-feet (lbf*ft). To specify velocity, you can use one of the defined unit abbreviations (mph, fpm, fps), or an expression based on any combination of the defined units of length and time, such as meters/second (m/s), millimeters/second (mm/s), inches/minute (in/min), and so on.

Note Affine units (such as Celsius or Fahrenheit) are not allowed in unit expressions. For more information, see “About Affine Units” on page 6-12.

The following operators are supported in the unit mathematical expressions:

- * Multiplication
- / Division
- ^ Power
- + Plus — for exponents only
- Minus — for exponents only
- () Brackets to specify evaluation order

Metric unit prefixes, such as *kilo*, *milli*, or *micro*, are not supported. For example, if you want to use milliliter as a unit of volume, you have to add it to the unit registry:

```
pm_addunit('ml', 0.001, 'l');
```

The drop-down lists next to parameter names are automatically populated by those units that are commensurate with the unit of the parameter. If you specify the units by typing, it is your responsibility to enter units that are commensurate with the unit of the parameter. The unit manager performs error checking when you click **Apply** or **OK** in the block dialog box, and issues an error if you type an incorrect unit.

In the Simulink-PS Converter and the PS-Simulink Converter block dialog boxes, the drop-down lists are prepopulated with some common input and output units, and it is your responsibility to select or type a unit expression commensurate with the expected input or output units. The error checking for the converter blocks is performed at the time of simulation. See “Model Validation” on page 3-7 for details.

Thermal Unit Conversions

In this section...

“About Affine Units” on page 6-12

“When to Apply Affine Conversion” on page 6-12

“How to Apply Affine Conversion” on page 6-13

About Affine Units

Thermal units often require an affine conversion, that is, a conversion that performs both multiplication and addition. To convert from the old value T_{old} to the new value T_{new} , we need a linear conversion coefficient L and an offset O :

$$T_{new} = L * T_{old} + O$$

For example, to convert a temperature reading from degrees Celsius into degrees Fahrenheit, the linear term equals 9/5, and the offset equals 32:

$$T_{Fahr} = 9 / 5 * T_{Cels} + 32$$

Simscape unit manager defines kelvin (K) as the fundamental temperature unit. This makes Celsius (C) and Fahrenheit (Fh) affine units because they are both related to kelvin with an affine conversion. Rankine (R) is defined in terms of kelvin with a zero linear offset and, therefore, is not an affine unit.

The following are the default Simscape unit registry definitions for temperature units:

```
pm_adddimension('temperature', 'K'); % defines kelvin as fundamental temperature unit
pm_addunit('C', [1 273.15], 'K'); % defines Celsius in terms of kelvin
pm_addunit('Fh', [5/9 -32*5/9], 'C'); % defines Fahrenheit in terms of Celsius
pm_addunit('R', [5/9 0], 'K'); % defines rankine in terms of kelvin
```

When to Apply Affine Conversion

In dealing with affine units, sometimes you need to convert them using just the linear term. Usually, this happens when the value you convert represents relative, rather than absolute, temperature, $\Delta T = T_1 - T_2$.

$$\Delta T_{new} = L * \Delta T_{old}$$

In this case, adding the affine offset would yield incorrect conversion results.

For example, the outdoor temperature rose by 18 degrees Fahrenheit, and you need to input this value into your model. When converting this value into kelvin, use linear conversion

$$\Delta T_{kelvin} = 5 / 9 * \Delta T_{Fahr}$$

and you get 10 K, that is, the outdoor temperature changed by 10 kelvin. If you apply affine conversion, you will get a temperature change of approximately 265 kelvin, which is incorrect.

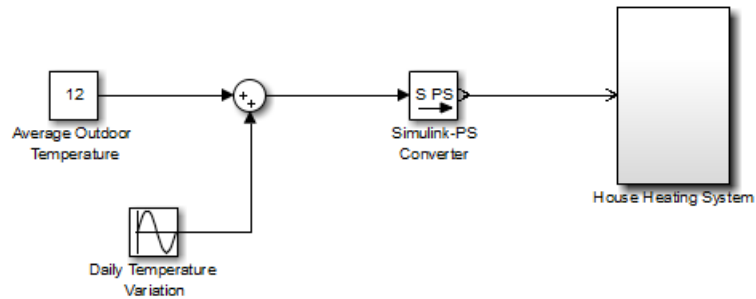
This is even better illustrated if you use degrees Celsius for the input units because the linear term for conversion between Celsius and kelvin is 1:

- If the outdoor temperature *changed* by 10 degrees Celsius (relative temperature value), then it changed by 10 kelvin (do not apply affine conversion).
- If the outdoor temperature *is* 10 degrees Celsius (absolute temperature value), then it is 283 kelvin (apply affine conversion).

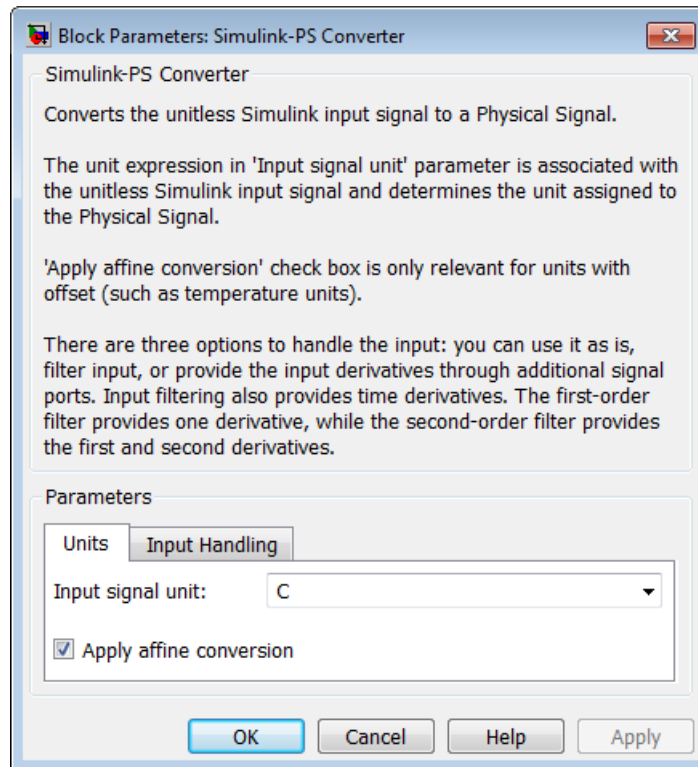
How to Apply Affine Conversion

When you specify affine units for an input temperature signal, it is important to consider whether you need to apply affine conversion. Usually this decision depends on whether the signal represents absolute or relative temperature (see “When to Apply Affine Conversion” on page 6-12).

For example, you model a house-heating system, and you need to input the outdoor temperature. In the following diagram, the Constant source block represents the average outdoor temperature, in degrees Celsius, and the Sine source block adds the daily temperature variation. The average outdoor temperature, in this case, is 12 degrees Celsius. Daily variation with an amplitude of 8 makes the input outdoor temperature vary between 4 and 20 degrees Celsius.



This signal is an absolute temperature reading. Therefore, when the signal converts into kelvin for further computations, you need to specify that it should use affine conversion. Double-click the Simulink-PS Converter block, type C in the **Input signal unit** field, and select the **Apply affine conversion** check box.



As a result, the Simulink-PS Converter block outputs a value varying between 277 K and 293 K.

Angular Units

Simscape implementation of angular units relies on the concept of angular units, specifically radians, being a unit but dimensionless. The notion of angular units being dimensionless is widely held in the metrology community. The fundamental angular unit, radian, is defined in the Simscape unit registry as:

```
pm_addunit('rad', 1, 'm/m');
```

which corresponds to the SI and NIST definition [1]. In other words, Simscape unit manager does not introduce a separate dimension, 'angle', with a fundamental unit of 'rad' (similar to dimensions for length or mass), but rather defines the fundamental angular unit in terms of meter over meter or, in effect, 1.

The additional angular units, degree and revolution, are defined respectively as:

```
pm_addunit('deg', pi/180, 'rad');  
pm_addunit('rev', 2*pi, 'rad');
```

As a result, forward trigonometric functions, such as `sin`, `cos`, and `tan`, work directly with arguments expressed in angular units. For example, `cosinus` of 90 degrees equals the `cosinus` of $(\pi/2)$ radians and equals the `cosinus` of $(\pi/2)$. Expansion of forward trigonometric functions works in a similar manner.

Another effect of dimensionless implementation of angular units is the convenience of the work-energy conversion. For example, torque (in $\text{N}\cdot\text{m}$) multiplied by angle (in rad) can be added directly to energy (in J, or $\text{N}\cdot\text{m}$). If you specify other commensurate units for the components of this equation, Simscape unit manager performs the necessary unit conversion operations and the result is the same.

References

[1] *The NIST Reference on Constants, Units, and Uncertainty*,
<http://physics.nist.gov/cuu/Units/units.html>

Units for Angular Velocity and Frequency

Angular velocity units, such as rad/s, deg/s, and rpm, can also be used to measure frequency for cyclical processes. This is consistent with frequency defined as revolutions per second in a mechanical context, or cycles per second in an electrical context, and lets you write frequency-dependent equations without requiring the 2π conversion factor. In the SI unit system, however, the unit of frequency is hertz (Hz), defined as 1/s.

Simscape software defines the unit hertz (Hz) as 1/s, in compliance with the SI unit system. This definition works well when frequency refers to a nonrotational periodic signal such as the frequency of a PWM source. For cyclical processes, however, the block equations have to contain the 2π conversion factor, to convert the numerical value specified in Hz, or s^{-1} , to angular frequency.

As a result, frequency units (based on Hz) and angular velocity units (based on rpm) are not directly convertible, and using one instead of the other may result in unexpected conversion factors applied to the numerical values by the block equations. For example, the AC Voltage Source block explicitly multiplies the value you specify for its **Frequency** parameter by 2π , to convert it to angular frequency before calculating the sine function.

Drop-down lists of suggested units in block dialogs reflect this distinction. For example, if a block has a **Frequency** parameter with the default unit of Hz, the drop-down list for this parameter contains only units directly convertible to Hz (such as kHz, MHz, and GHz) and does not contain the angular velocity units. Conversely, if you define a custom block where the **Frequency** parameter has the default unit of rpm, its drop-down list of suggested units will include deg/s and rad/s, but will not contain Hz, kHz, MHz, or GHz.

When you type a unit expression in the parameter units combo box (instead of selecting a value from the drop-down list), the Simscape unit manager considers the units of frequency and angular velocity to be commensurate. For example, when the default parameter unit is Hz, you are able to type not only 1/s, but also expressions such as deg/s and rad/s. This behavior is consistent with the Simscape implementation of angular units (see “Angular Units” on page 6-16). It is your responsibility to verify that the unit expression you typed works correctly with the block equations and reflects your design intent.

Note Prior to Release R2013a, the unit definition for Hz was rev/s . For information on how to update legacy models and custom Simscape libraries written in R2012b or earlier, see Compatibility Considerations under “Unit definition of Hz now consistent with SI”, in the R2013a Release Notes.

Add-On Product License Management

- “About the Simscape Editing Mode” on page 7-2
- “Working with Restricted and Full Modes” on page 7-9
- “Editing Mode Information” on page 7-24

About the Simscape Editing Mode

In this section...
“Suggested Workflows” on page 7-2
“What You Can Do in Restricted Mode” on page 7-3
“What You Can Do in Full Mode” on page 7-4
“Switching Between Modes” on page 7-4
“Working with Block Libraries” on page 7-7

Suggested Workflows

The Simscape Editing Mode functionality is implemented for customers who perform physical modeling and simulation using Simscape platform and its add-on products: SimDriveline, SimElectronics, SimHydraulics, SimMechanics, and SimPowerSystems. It allows you to open, simulate, and save models that contain blocks from add-on products in Restricted mode, without checking out add-on product licenses, as long as the products are installed on your machine. It is intended to provide an economical way to distribute simulation models throughout a team or organization.

Note Unless your organization uses concurrent licenses, see the Simscape product page on the MathWorks Web site for specific information on how to install add-on products on your machine, to be able to work in Restricted mode.

The Editing Mode functionality supports widespread use of Physical Modeling products throughout an engineering organization by making it economical for one user to develop a model and provide it to many other users.

Specifically, this feature allows a user, *model developer*, to build a model that uses Simscape platform and one or more add-on products and share that model with other users, *model users*. When building the model in Full mode, the model developer must have a Simscape license and the add-on product licenses for all the blocks in the model. For example, if a model combines Simscape, SimHydraulics, and SimDriveline blocks, the model developer needs to check out licenses for all three products to work with it in Full mode.

Once the model is built, model users need only to check out a Simscape license to simulate the model and fine-tune its parameters in Restricted mode. As long as no structural changes are made to the model, model users can work in Restricted mode and do not need to check out add-on product licenses.

Another workflow, available with concurrent licenses only, lets multiple users, who all have Simscape licenses, share a small number of add-on product licenses by working mostly in Restricted mode, and temporarily switching models to Full mode only when they need to perform a specific design task that requires being in Full mode.

Note MathWorks recommends that you save all the models in Full mode before upgrading to a new version of Simulink or Simscape software.

If you have saved a model in Restricted mode and, upon upgrading to a new product version, open the model and it does not run, switch it to Full mode and save. You can then again switch to Restricted mode and work without problem.

What You Can Do in Restricted Mode

When your model is open in Restricted mode, you can:

- Simulate the model.
- Inspect parameters.
- Change certain block parameters. In general, you can change numerical parameter values, but cannot change the block parameterization options. See the block reference pages for specifics.
- Generate code.
- Make data logging or visualization changes.
- Add or delete regular Simulink blocks (such as sources or scopes) and appropriate connections.

For other types of changes, listed in the following section, your model has to be in Full mode. Some of these disallowed changes are impossible to make in Restricted mode (for example, Restricted parameters are grayed out in block

dialog boxes). Other changes, like changing the physical topology of a model, are not explicitly disallowed, but if you make these changes in Restricted mode, the software will issue an error message when you try to run, compile, or save such a model.

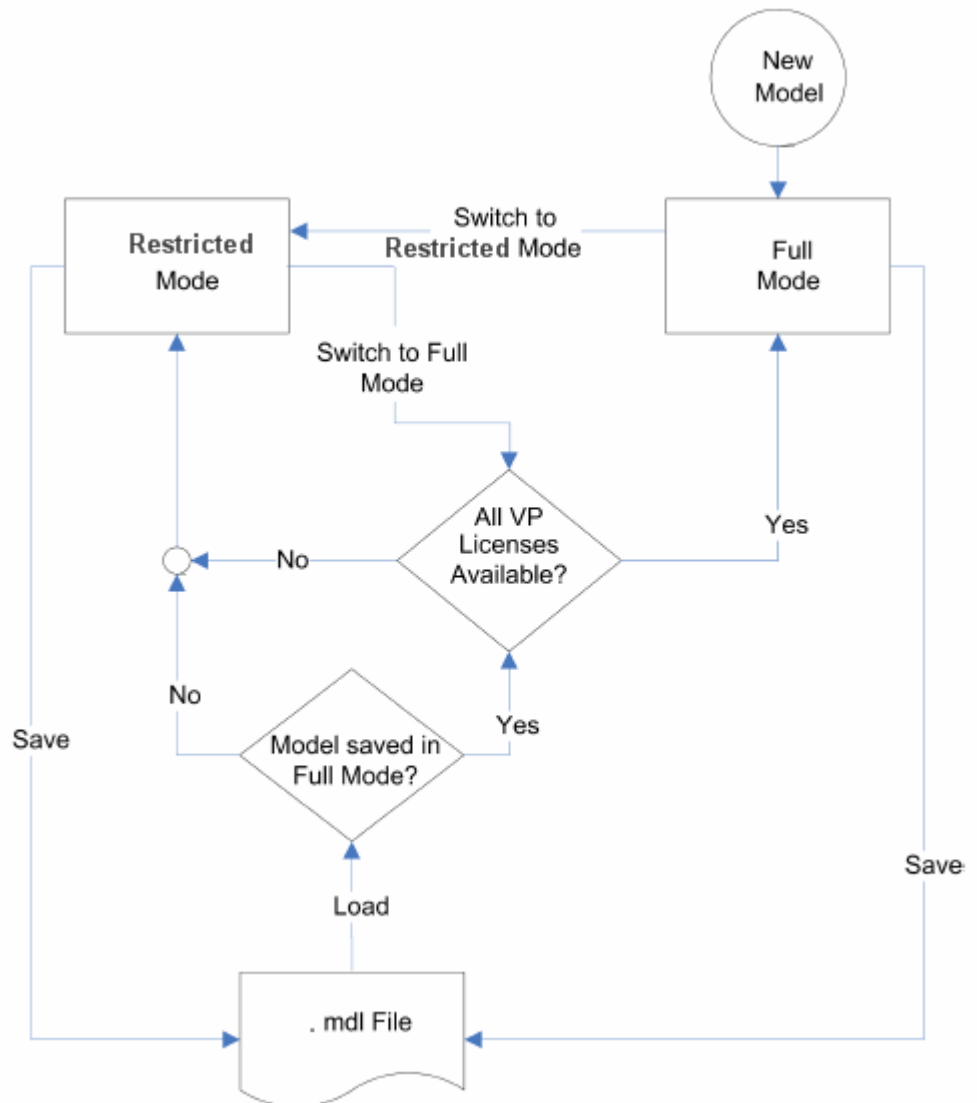
What You Can Do in Full Mode

You need to open a model in Full mode if you need to do any of the following:

- Add or delete Physical Modeling blocks (that is, Simscape blocks or blocks from the add-on product libraries).
- Make or break Physical connections (between Conserving or Physical Signal ports).
- Change the types of signals going into actuators or out of sensors (for example, from velocity to torque).
- Change configuration parameters.
- Change block parameterization options and other restricted parameters.
- Change physical units of parameters.
- Protect a referenced model containing Physical Modeling blocks (for more information, see “Protected Model”).

Switching Between Modes

The following flow chart shows what happens when you switch between modes.



New models are always created in Full mode. You can then either save the model in Full mode, or switch to Restricted mode and save the model in Restricted mode.

When you load an existing model, the license manager checks whether it has been saved in Full or Restricted mode.

- If the model has been saved in Restricted mode, it opens in Restricted mode.
- If the model has been saved in Full mode, the license manager checks whether all the add-on product licenses for this model are available and, if so, opens it in Full mode. If a add-on product license is not available, the license manager issues an error message and opens the model in Restricted mode. See also “Example with Multiple Add-On Products” on page 7-6.

Note You can set a Simulink preference to specify that the models are always to open in Restricted mode, regardless of the way they have been saved.

When a model is open, you can transition it between Full and Restricted modes at any time, in either direction:

- When you try to switch from Restricted to Full mode, the license manager checks whether all the add-on product licenses for this model are available. If a add-on product license is not available, the license manager issues an error message and the model stays in Restricted mode. See also “Example with Multiple Add-On Products” on page 7-6.
- No checks are performed when switching from Full to Restricted mode.

Note If a add-on product license has been checked out to open a model in Full mode, it remains checked out for the remainder of the MATLAB session. Switching to Restricted mode does not immediately return the license.

Example with Multiple Add-On Products

When you try to open a model in Full mode or to switch from Restricted to Full mode, the license manager scans the model and attempts to check out the required add-on product licenses as it encounters them in the model. If a license is not available, the license manager issues an error message and the model stays in Restricted mode. The licenses are checked out sequentially. As a result, if a model uses blocks from multiple add-on products, some of the

add-on product licenses may have already been checked out by the time the license manager encounters an unavailable license. In this case, these add-on product licenses stay checked out until you quit the MATLAB session, even though the model is in Restricted mode.

For example, consider a model that uses blocks from SimHydraulics and SimDriveline libraries, but the user who tries to open it has only the SimDriveline license available. It may happen that the license manager checks out a SimDriveline license first, and then tries to check out a SimHydraulics license, which is not available. The license manager then issues an error message and opens the model in Restricted mode, but the SimDriveline license stays checked out until the end of the MATLAB session.

Working with Block Libraries

This section describes the specifics of working with block libraries while using the Editing Mode functionality. These rules are applicable to any physical modeling blocks, that is, blocks from all Simscape libraries, including the add-on products. In general, you need to work in Full mode when you modify a library block. However, when you open a model that references the modified block, you may work in Restricted mode, under certain conditions. The following summary details the Editing Mode rules for modifying and using library blocks:

- To add physical modeling blocks to a library block, you need to work in Full mode.
 - If this library block had not previously contained physical modeling blocks, you need to work in Full mode to load a preexisting model that uses this library block or to drag this block to a model.
 - If this library block had previously contained physical modeling blocks, you can work in Restricted mode when loading a preexisting model that uses this library block. However, you have to work in Full mode to drag this block from the library to a model.
- To add external physical ports to a library block, you need to work in Full mode.
 - You can work in Restricted mode when loading a preexisting model that uses this library block.

- However, to connect these additional ports, you need to work in Full mode because you are changing the model topology.
- To delete external physical ports from a library block, you need to work in Full mode. If these ports were connected in a model saved in Restricted mode, loading the model causes the topology to change, so you need to switch to Full mode to save or compile the model.

Resolving Block Library Links

All Simscape blocks in your models, including the add-on products' blocks, must have resolved block library links. You can neither disable nor break these library links. This is a global requirement of Simscape platform, which is necessary to enforce the Editing Mode rules for modifying and using library blocks, listed above. A model with broken library links will neither compile nor save. You must restore all the broken block library links for your model to be valid.

If you want to customize certain blocks and use them in your models, you must add these modified blocks to your own custom library, then copy the block instances that you need to your model.

Working with Restricted and Full Modes

In this section...

“Set the Model Loading Preference” on page 7-9

“Save a Model in Restricted Mode” on page 7-10

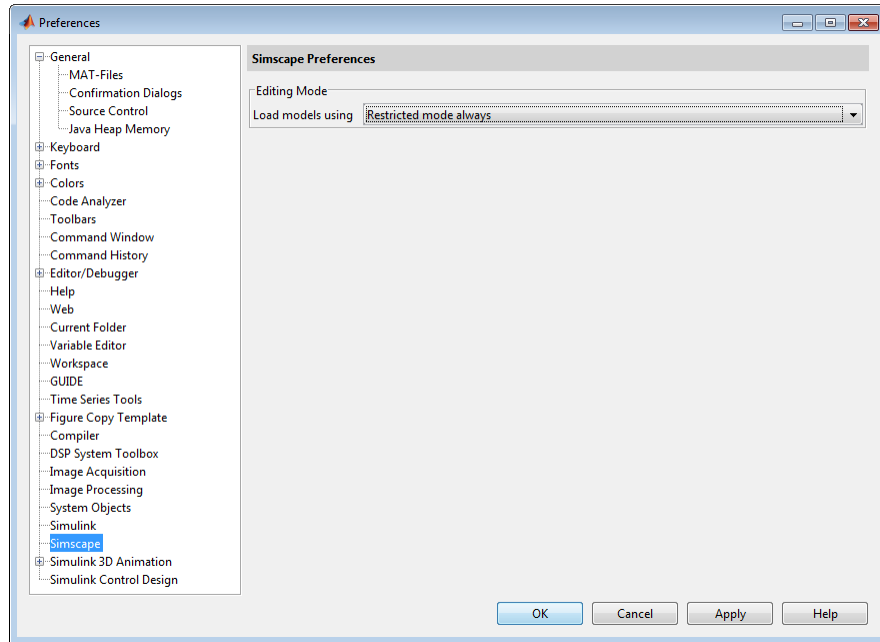
“Work with a Model in Restricted Mode” on page 7-13

“Switch from Restricted to Full Mode” on page 7-22

Set the Model Loading Preference

By default, when you load an existing model, the license manager checks whether it has been saved in Full or Restricted mode and tries to open it in this mode. However, you can set your preferences so that the models are always open in Restricted mode, regardless of the way they have been saved.

- 1** On the MATLAB Toolstrip, click **Preferences**. The Preferences dialog box opens.
- 2** In the left pane of the Preferences dialog box, select **Simscape**. The right pane displays the **Editing Mode** group box. By default, the **Load models using** option is set to Editing mode specified in models.
- 3** Select **Restricted mode** always from the drop-down list, as shown, and click **OK**.

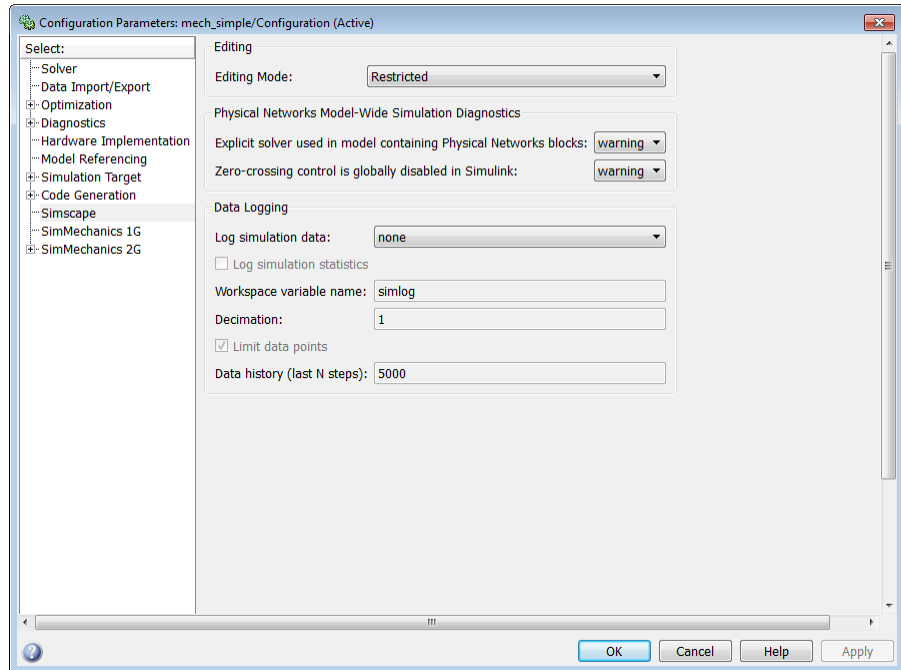


Now, when you open a model, the license manager does not attempt to check out add-on product licenses and always opens the model in Restricted mode.

Save a Model in Restricted Mode

Rather than setting your preferences so that all the models always open in Restricted mode, you can switch an individual model to Restricted mode before saving it. Such a model will then, by default, open in Restricted mode.

- 1 From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. The Configuration Parameters dialog box opens.
- 2 In the left pane of the Configuration Parameters dialog box, select **Simscape**. The right pane displays the **Editing Mode** option, which is by default set to Full.
- 3 Select **Restricted** from the drop-down list, as shown, and click **OK**.



4 Save the model.

Note The **Simscape** entry does not appear in the left pane of the Configuration Parameters dialog box until you add at least one Physical Modeling block to your model. If you create an additional configuration set for a model, the **Simscape** entry does not appear in it until you either activate it or perform a Physical Modeling operation, such as adding or deleting a Physical Modeling block or connection, opening a Physical Modeling block dialog box, and so on.

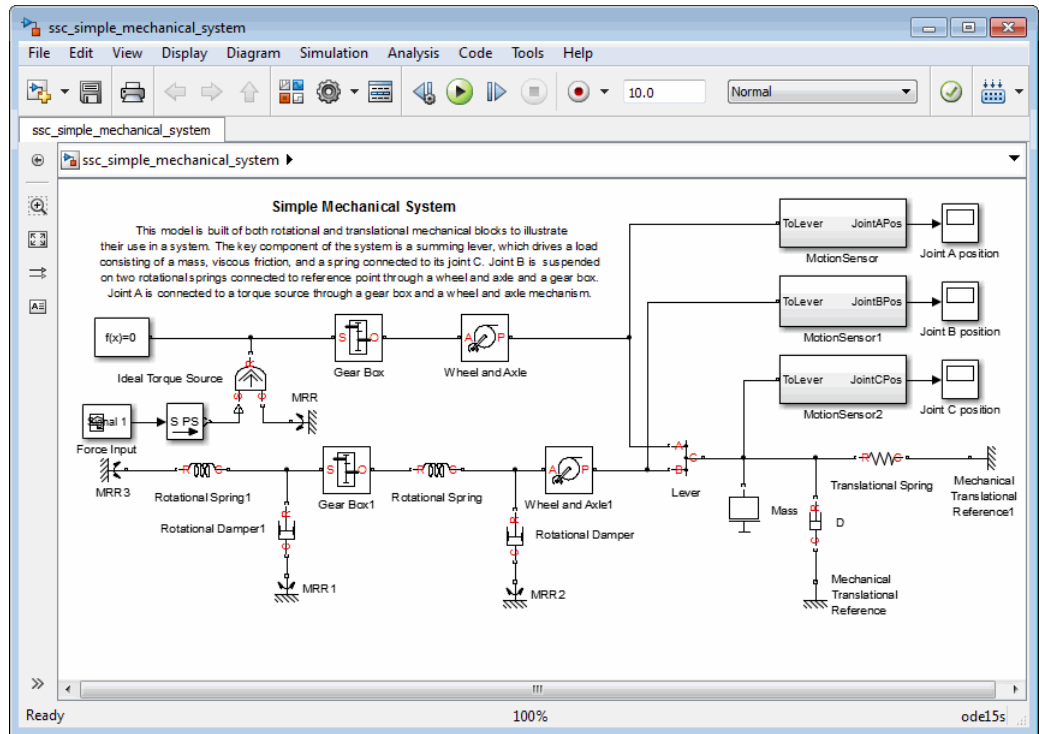
Once you have switched a model to Restricted mode, working with it follows the rules described in “Work with a Model in Restricted Mode” on page 7-13. Note, however, that the add-on product licenses for this model stay checked out until you quit the MATLAB session.

When you open a model that has been saved in Restricted mode, the license manager opens it in Restricted mode and does not check out the add-on product licenses.

Example of Saving a Model in Restricted Mode

In this example, you switch a model to Restricted mode and save it.

- 1 Open the Simple Mechanical System example model (ssc_simple_mechanical_system).



- 2 From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. The Configuration Parameters dialog box opens.

- 3 In the left pane of the Configuration Parameters dialog box, select **Simscape**. The right pane displays the **Editing Mode** option, which is set to **Full** by default.
- 4 Select **Restricted** from the drop-down list and click **OK**.
- 5 Save the model as `model_test_edit_mode`.

Work with a Model in Restricted Mode

When you open a model in Restricted mode, you can perform a variety of tasks: simulate the model, inspect and fine-tune block parameters, add and delete basic Simulink blocks, and so on. For a complete list of allowed operations, see “What You Can Do in Restricted Mode” on page 7-3.

When you open a block dialog box in Restricted mode, some of the block parameters may be grayed out. These are the so-called *restricted* parameters that can be modified only in Full mode. In general, you can change numerical parameter values in Restricted mode, but you cannot change the block parameterization options. See the block reference pages for specifics. Note also that when a restricted parameter defines the block parameterization schema, nonrestricted parameters available for fine-tuning in Restricted mode depend on the value of this restricted parameter. For example, in a Constant Volume Chamber block, the **Chamber specification** parameter is restricted. If, at the time the model entered Restricted mode, this parameter was set to **By volume**, then the nonrestricted parameters available for fine-tuning would be **Chamber volume**, **Specific heat ratio**, and **Initial pressure**. If, however, it was set to **By length and diameter**, you will have a different set of parameters available in Restricted mode.

You cannot change physical units in Restricted mode. When you open a block dialog box in Restricted mode, the drop-down lists of units next to a parameter name and value are grayed out. When you open a PS-Simulink Converter or Simulink-PS Converter block dialog box, the **Unit** parameter is grayed out.

The following examples illustrate operations allowed and disallowed in Restricted mode:

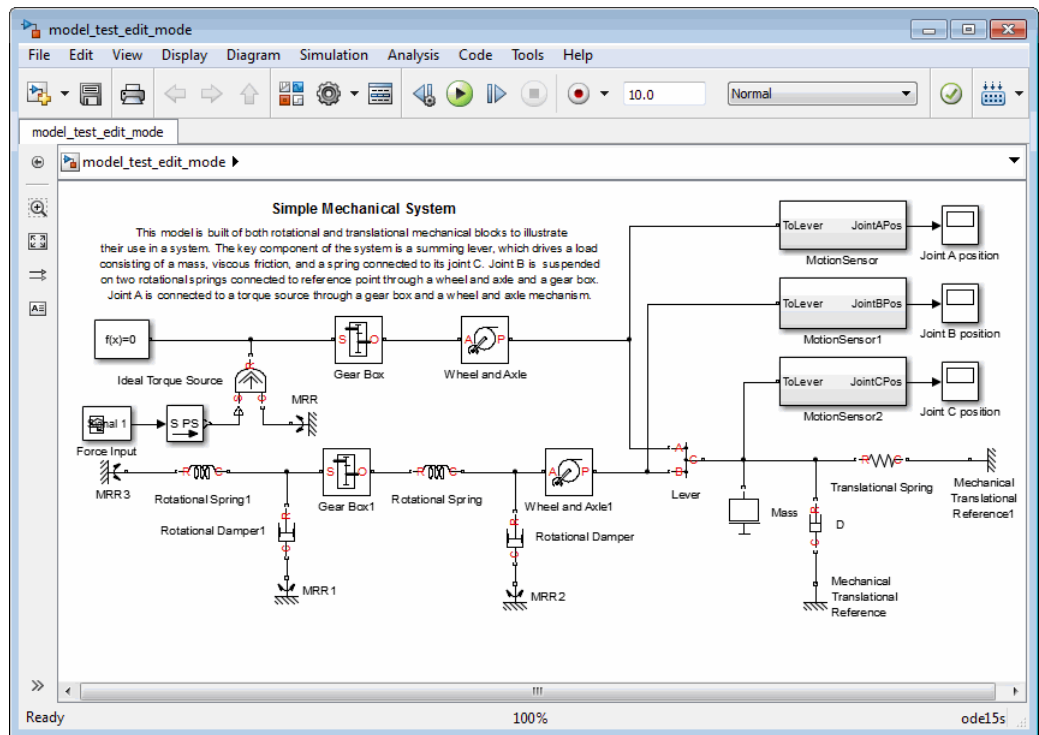
- “How to Simulate and Fine-Tune a Model in Restricted Mode” on page 7-14
- “How to Add and Delete Simulink Blocks in Restricted Mode” on page 7-17

- “Performing an Operation Disallowed in Restricted Mode” on page 7-20

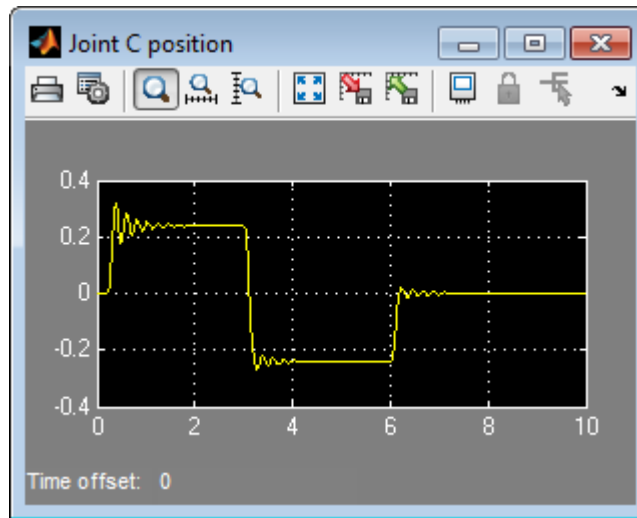
How to Simulate and Fine-Tune a Model in Restricted Mode

This example shows how you can work with a model in Restricted mode by changing certain parameter values and observing the simulation results.

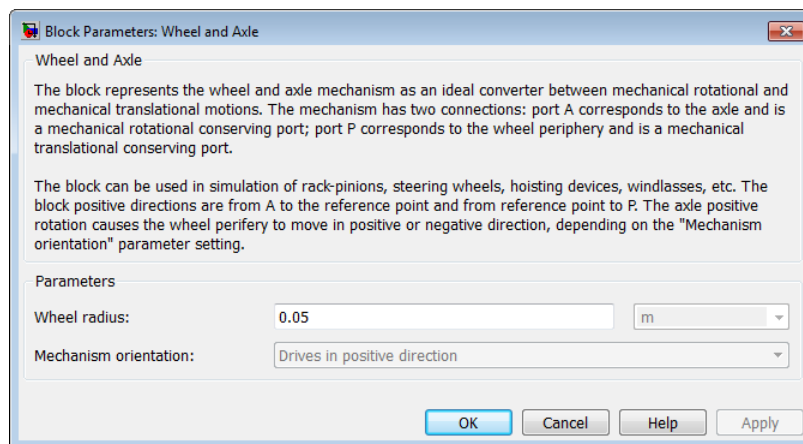
- 1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in “Example of Saving a Model in Restricted Mode” on page 7-12. The model opens in Restricted mode.



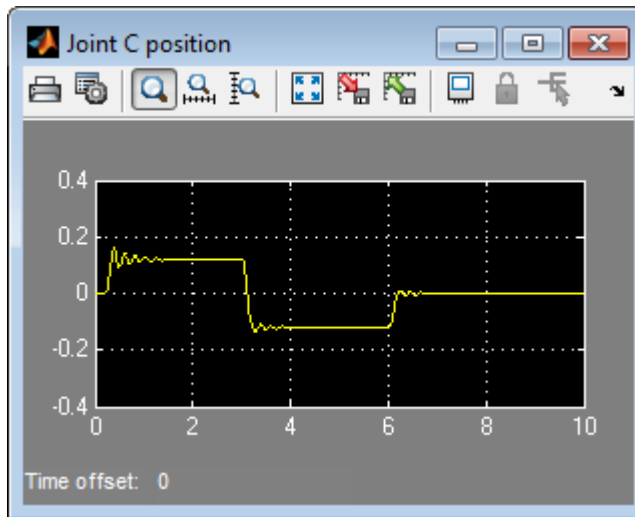
- 2 Open the Joint C Position scope and simulate the model. The models runs and simulates in Restricted mode.



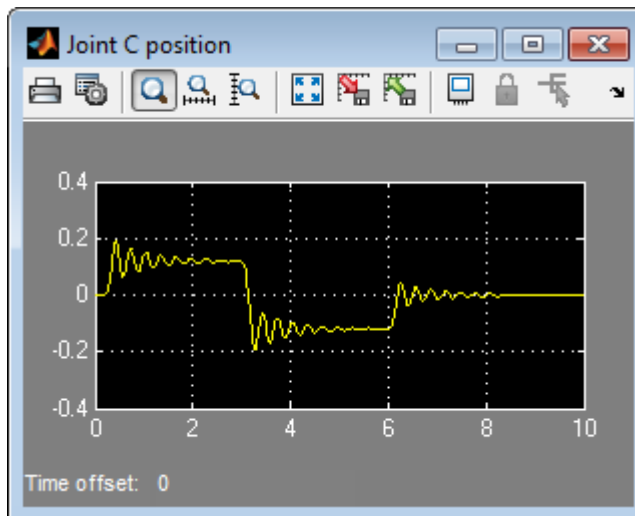
- 3 Double-click the Wheel and Axle block to open its dialog box. Notice that the **Mechanism orientation** parameter is grayed out, because you cannot modify the block driving direction in Restricted mode.



- 4 Change the **Wheel radius** parameter value to 0.1.
- 5 Simulate the model again. Notice that the motion amplitude of node C became smaller as a result of the wheel radius change.



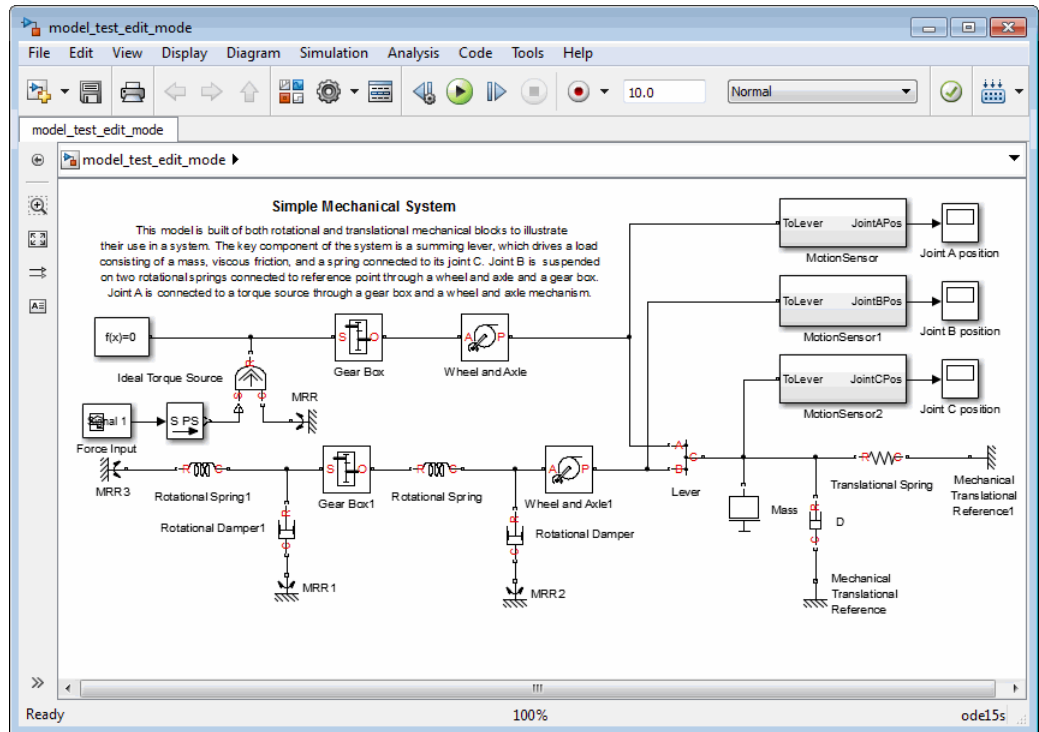
- 6** Double-click the Mass block and change the **Mass** parameter value to 24.
- 7** Simulate the model. Notice that doubling the mass resulted in increased vibrations.



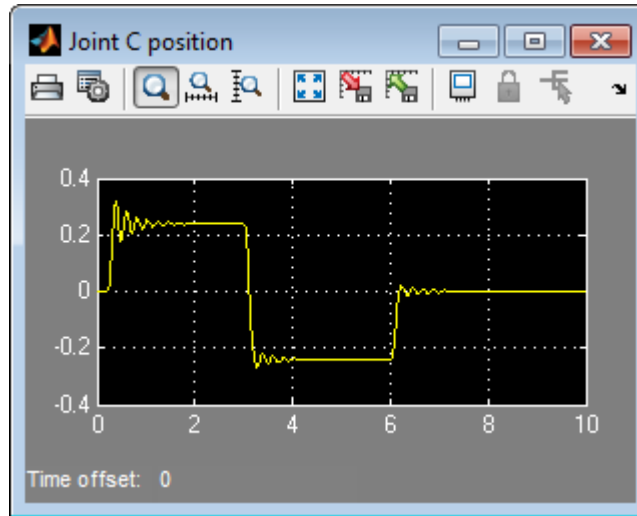
How to Add and Delete Simulink Blocks in Restricted Mode

This example shows how you can change the model input signal in Restricted mode by adding and deleting basic Simulink blocks.

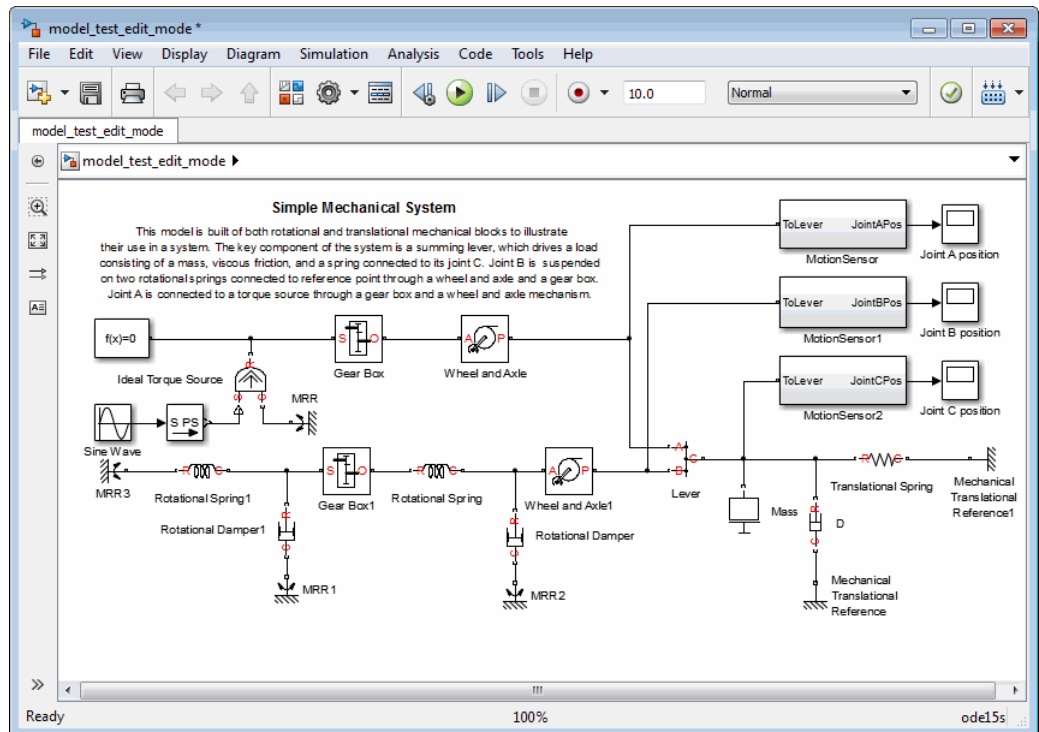
- 1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in “Example of Saving a Model in Restricted Mode” on page 7-12. The model opens in Restricted mode.



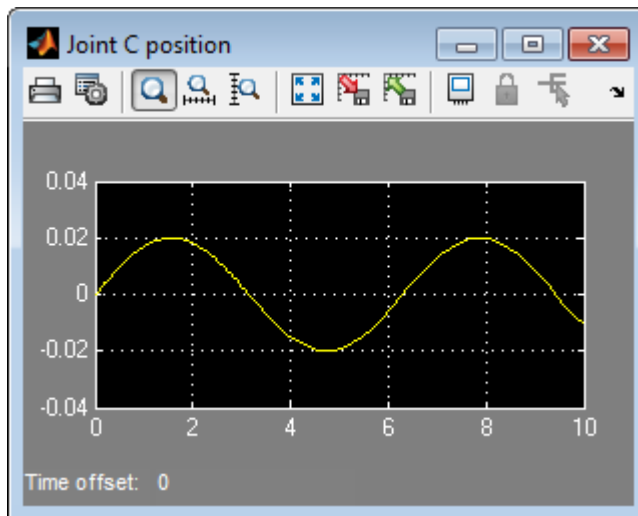
- 2 Open the Joint C Position scope and simulate the model.



- 3** Delete the Signal Builder block named Force Input. Replace it with a Sine Wave block from the Simulink Sources library, as shown below.



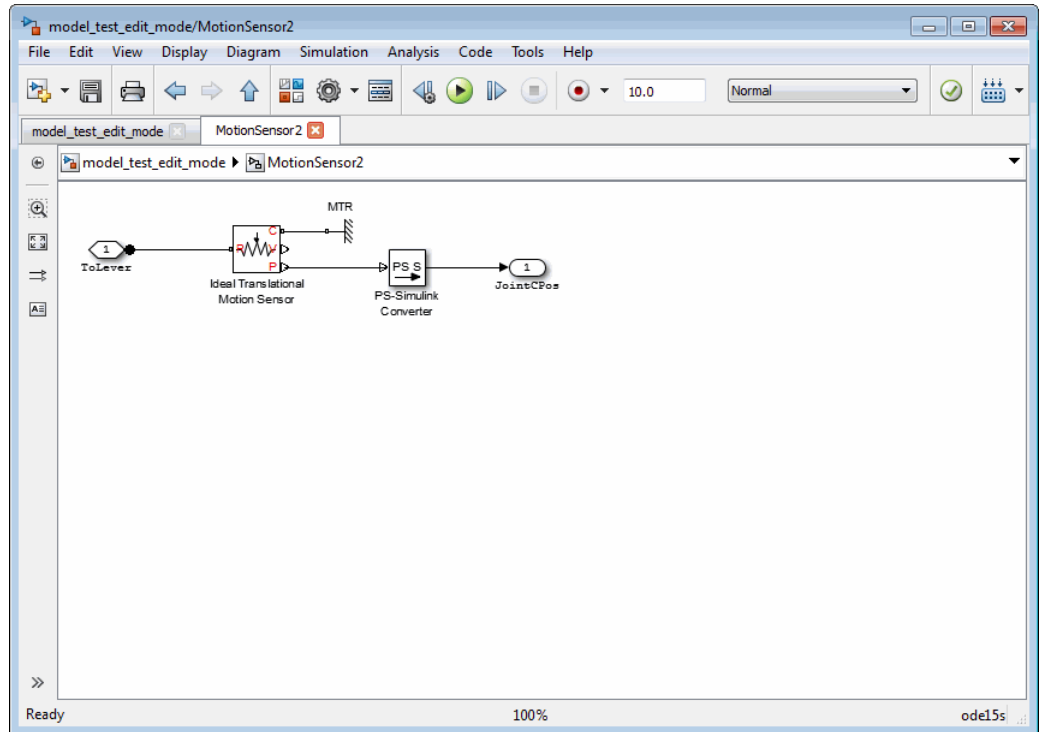
- 4** Simulate the model again. The model successfully compiles and simulates in Restricted mode.



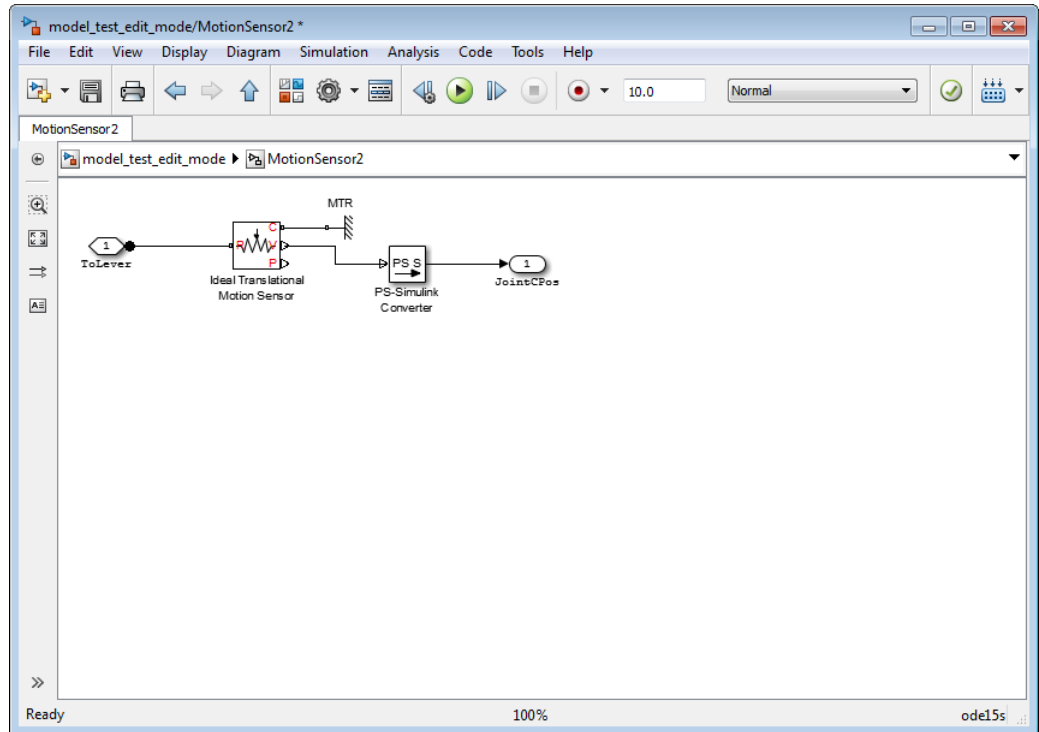
Performing an Operation Disallowed in Restricted Mode

This example shows what happens when you perform an operation that is disallowed in Restricted mode.

- 1 Open the `model_test_edit_mode` model, which you saved in Restricted mode in “Example of Saving a Model in Restricted Mode” on page 7-12. The model opens in Restricted mode.
- 2 Double-click the `MotionSensor2` block to open the subsystem.



- 3 Delete the connection line between port P of the Ideal Translational Motion Sensor block and the PS-Simulink Converter block. Instead, connect port V of the Ideal Translational Motion Sensor block to the input port of the PS-Simulink Converter block, to measure the velocity on node C of the lever.



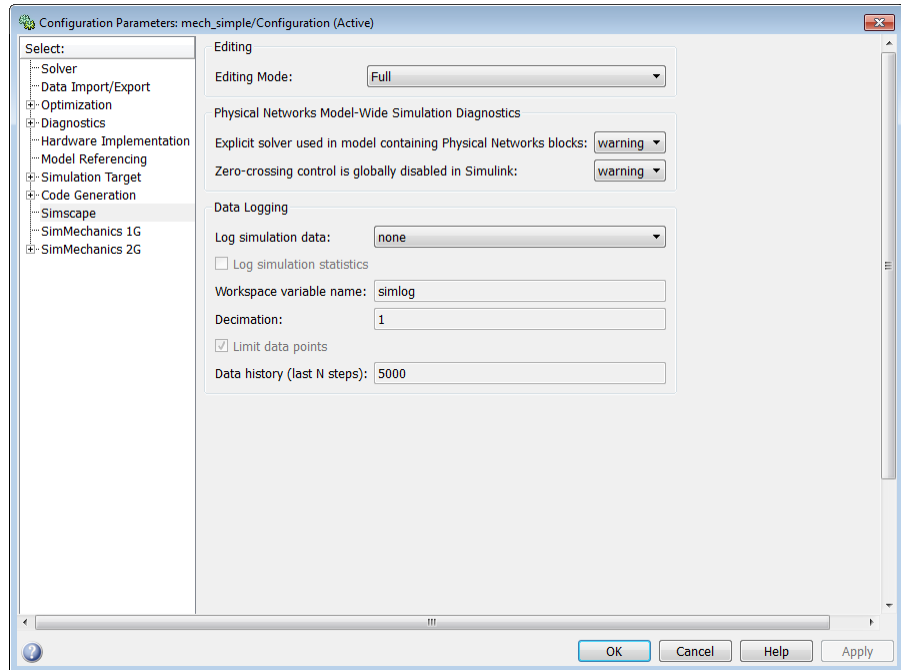
- 4 Try to simulate the model. An error message appears saying that the model cannot be compiled because its topology has been changed while in Restricted mode. You can either undo the changes, or switch to Full mode, as described in “Switch from Restricted to Full Mode” on page 7-22.

Switch from Restricted to Full Mode

If you need to perform a task that is disallowed in Restricted mode, you can try to switch the model to Full mode.

- 1 From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. The Configuration Parameters dialog box opens.
- 2 In the left pane of the Configuration Parameters dialog box, select **Simscape**. The right pane displays the **Editing Mode** option.

3 Select **Full** from the drop-down list, as shown, and click **OK**.



The license manager checks whether all the add-on product licenses for this model are available. If yes, it checks out the add-on product licenses and switches the model to Full mode. If a add-on product license is not available, the license manager issues an error message and the model stays in Restricted mode.

Note If the switch to Full mode fails but some of the add-on product licenses have already been checked out, they stay checked out until you quit the MATLAB session. For more information, see “Example with Multiple Add-On Products” on page 7-6.

Once the model is switched to Full mode, you can perform the needed design and simulation tasks, and then either save it in Full mode, or switch back to Restricted mode and save it in Restricted mode.

Editing Mode Information

In this section...
“What Is the Current Mode?” on page 7-24
“Which Licenses Are Checked Out?” on page 7-24

What Is the Current Mode?

If you are unsure whether the model is currently open in Restricted or Full mode, you can check by following these steps.

- 1** From the top menu bar in the model window, select **Simulation > Model Configuration Parameters**. The Configuration Parameters dialog box opens.
- 2** In the left pane of the Configuration Parameters dialog box, select **Simscape**. The right pane displays the **Editing Mode** option, which is either **Full** or **Restricted**.
- 3** At this point, you can either try switching the mode by selecting a different option from the drop-down list, or click **Cancel** to stay in the current mode.

Which Licenses Are Checked Out?

Use the MATLAB `license` command to get a list of all the licenses currently in use. In the MATLAB Command Window, type

```
license('inuse')
```

This command returns a list of licenses checked out in the current MATLAB session. In the list, products are listed alphabetically by their license feature names.

E

electrical ground
specifying 1-36

L

linearizing
Simscape™ models 3-55

N

numerical simulation issues
avoiding 1-40

O

operating points
finding in Simscape™ models 3-48
linearizing Simscape™ models at 3-55

P

ports
physical conserving 1-9
physical signal 1-10

S

Simscape Editing Mode 7-2
Full mode 7-4

information 7-24
Restricted mode 7-3
saving in Restricted mode 7-10
switching between modes 7-4
workflows 7-2
working in Restricted mode 7-13
working with block libraries 7-7

Simscape software
block library structure 1-11
editing modes 7-2
logging simulation data 4-2

T

trimming
Simscape™ models 3-48

U

units
defining physical units 6-4

V

variables
across 1-4
direction 1-6
through 1-4
using in model equations 1-5