

# SimMechanics

For Use with Simulink<sup>®</sup>

- Modeling
- Simulation
- Implementation

## How to Contact The MathWorks:



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



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



508-647-7000	Phone
--------------	-------



508-647-7001	Fax
--------------	-----



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

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

### *SimMechanics User's Guide*

© COPYRIGHT 2001-2004 by The MathWorks, Inc.

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

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

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

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

Printing History:	December 2001	Online only	Version 1 (Release 12.1+)
	July 2002	First printing	Revised for Version 1.1 (Release 13)
	November 2002	Online only	Revised for Version 2.0 (Release 13+)
	October 2004	Online only	Revised for Version 2.0 (Release 13SP2)

## About This Guide

---

<b>Using This Guide</b> .....	<b>x</b>
<b>Getting Online Help</b> .....	<b>xii</b>
For Further Help and Feedback .....	<b>xii</b>
<b>Opening SimMechanics Demos</b> .....	<b>xiii</b>
<b>Typographical Conventions</b> .....	<b>xiv</b>
<b>Mechanical Units</b> .....	<b>xv</b>
<b>Mechanical Conventions and Abbreviations</b> .....	<b>xvi</b>
Right-Hand Rule Is Assumed .....	<b>xvi</b>
Vector Multiplication .....	<b>xvi</b>
Common Abbreviations .....	<b>xvi</b>
Glossary Terms .....	<b>xvi</b>

## Introducing SimMechanics

---

**1**

<b>Welcome to SimMechanics</b> .....	<b>1-2</b>
SimMechanics and Physical Modeling .....	<b>1-2</b>
What's in This Chapter .....	<b>1-2</b>
<b>Related Products</b> .....	<b>1-3</b>
Requirements for SimMechanics .....	<b>1-3</b>
Other Related Products .....	<b>1-3</b>
<b>Running a Demo Model</b> .....	<b>1-5</b>
What This Demo Illustrates .....	<b>1-6</b>

Opening the Model .....	1-6
Starting the Demo .....	1-9
Modifying the Model .....	1-10
<b>What Can You Do with SimMechanics?</b> .....	<b>1-17</b>
Modeling Machines with SimMechanics .....	1-17
Bodies, Coordinate Systems, Joints, and Constraints .....	1-18
Sensors, Actuators, and Force Elements .....	1-19
Simulating Mechanical Motion .....	1-19
Visualizing and Animating Machines .....	1-21

## Building and Visualizing Simple Machines

# 2

<b>Introducing the SimMechanics Block Libraries</b> .....	<b>2-2</b>
Viewing the Blocks .....	2-2
<b>Creating SimMechanics Models</b> .....	<b>2-7</b>
Essential Steps to Build a Model .....	2-7
Essential Steps to Configure and Run a Model .....	2-9
<b>Building a Simple Pendulum</b> .....	<b>2-11</b>
The World Coordinate System and Gravity .....	2-12
Configuring a Ground Block .....	2-12
Configuring a Body Block .....	2-14
Configuring a Joint Block .....	2-20
Adding Sensors and Starting the Simulation .....	2-24
<b>Visualizing a Simple Pendulum</b> .....	<b>2-30</b>
Representing the Bodies .....	2-31
Visualizing with Handle Graphics .....	2-32
Visualizing with Virtual Reality .....	2-34
<b>A Four Bar Mechanism</b> .....	<b>2-36</b>
Configuring the Mechanical Environment Settings .....	2-38
Setting Up the Block Diagram .....	2-40
Configuring the Ground and Joint Blocks .....	2-43

Configuring the Body Blocks . . . . .	2-47
Sensing Motion and Running the Model . . . . .	2-52

## Understanding Mechanical Concepts

---

### 3

<b>Representing Body Positions and Orientations . . . . .</b>	<b>3-2</b>
Reference Frames and Coordinate Systems . . . . .	3-2
Relating Moving Coordinate Systems . . . . .	3-3
Observing Motion in Different Coordinate Systems . . . . .	3-4
Observing a Translating, Rotating Rigid Body . . . . .	3-6
How SimMechanics Represents Body Rotations . . . . .	3-7
Kinematics and the Machine State . . . . .	3-9
 <b>Summary of Technical Vocabulary . . . . .</b>	 <b>3-11</b>

## Modeling Mechanical Systems

---

### 4

<b>Modeling Machines . . . . .</b>	<b>4-2</b>
About SimMechanics Models . . . . .	4-2
Creating a SimMechanics Model . . . . .	4-3
Connecting SimMechanics Blocks . . . . .	4-3
Interfacing SimMechanics Blocks to Simulink Blocks . . . . .	4-5
Setting SimMechanics Block Properties at the Command Line . . . . .	4-5
Creating SimMechanics Subsystems . . . . .	4-5
Creating Custom SimMechanics Blocks with Masks . . . . .	4-7
 <b>Modeling Bodies . . . . .</b>	 <b>4-8</b>
Modeling Grounds . . . . .	4-8
Modeling Finite Rigid Bodies . . . . .	4-9
Working with Body Coordinate Systems . . . . .	4-12

<b>Modeling Joints</b> .....	<b>4-17</b>
About Joints .....	4-17
Creating a Joint .....	4-23
Modeling with Massless Connectors .....	4-26
Modeling with Disassembled Joints .....	4-29
Cutting Closed Loops .....	4-32
<b>Modeling Constraints and Drivers</b> .....	<b>4-34</b>
Directionality of Constraints and Drivers .....	4-35
Solving Constraints .....	4-35
Restrictions on Using Constraint and Driver Blocks .....	4-35
Constraint Example: Gear Constraint .....	4-35
Driver Example: Using the Angle Driver .....	4-37
<b>Modeling Actuators</b> .....	<b>4-40</b>
Actuating a Body .....	4-40
Varying a Body's Mass and Inertia Tensor .....	4-43
Actuating a Joint .....	4-44
Actuating a Driver .....	4-49
Specifying Initial Positions and Velocities .....	4-50
<b>Modeling Force Elements</b> .....	<b>4-55</b>
Inserting a Linear Force Between Bodies .....	4-55
Inserting a Linear Force or Torque Through a Joint .....	4-57
Customizing Force Elements with Sensor-Actuator Feedback	4-58
<b>Modeling Sensors</b> .....	<b>4-60</b>
Sensing Body Motions .....	4-60
Sensing Joint Motions and Forces .....	4-62
Sensing Constraint Reaction Forces .....	4-62
<b>Checking Model Validity</b> .....	<b>4-64</b>
Checking Schematic Topology .....	4-64
Counting Degrees of Freedom .....	4-67

<b>Choosing Simulation Options</b> .....	<b>5-2</b>
Choosing an ODE Solver .....	<b>5-2</b>
Choosing an Analysis Mode .....	<b>5-4</b>
Setting Assembly Tolerances .....	<b>5-5</b>
Choosing a Constraint Solver .....	<b>5-5</b>
<b>Mechanical Environment Settings Dialog Box</b> .....	<b>5-7</b>
Parameters Pane .....	<b>5-7</b>
Constraints Pane .....	<b>5-9</b>
Linearization Pane .....	<b>5-10</b>
Visualization Pane .....	<b>5-12</b>
<b>How SimMechanics Works</b> .....	<b>5-14</b>
Model Validation .....	<b>5-14</b>
Machine Initialization .....	<b>5-14</b>
Force Analysis and Motion Integration .....	<b>5-15</b>
Stiction Mode Iteration .....	<b>5-15</b>
<b>Troubleshooting Simulation Errors</b> .....	<b>5-16</b>
Data Validation Errors .....	<b>5-16</b>
Ground and Body Geometry Errors .....	<b>5-16</b>
Joint Geometry Errors .....	<b>5-17</b>
Block Connection and Topology Errors .....	<b>5-17</b>
Motion Singularity and Inconsistency Errors .....	<b>5-18</b>
Analysis Mode Errors .....	<b>5-21</b>
<b>Generating Code</b> .....	<b>5-22</b>
Using Code-Related Products and Features .....	<b>5-22</b>
How SimMechanics Code Generation Differs from Simulink ..	<b>5-23</b>
<b>Limitations</b> .....	<b>5-25</b>
Restrictions with Generated Code .....	<b>5-25</b>

<b>Choosing Visualization Options in SimMechanics</b> . . . . .	<b>6-2</b>
Uses of the SimMechanics Visualization Tools . . . . .	<b>6-3</b>
Setting Up Static and Dynamic Visualization . . . . .	<b>6-4</b>
<b>Rendering Body Shapes in SimMechanics</b> . . . . .	<b>6-5</b>
Choosing the Body Shape . . . . .	<b>6-5</b>
Equivalent Ellipsoids . . . . .	<b>6-5</b>
Convex Hulls . . . . .	<b>6-8</b>
<b>Viewing Machines with Handle Graphics</b> . . . . .	<b>6-10</b>
Using the Standard Handle Graphics Controls . . . . .	<b>6-12</b>
Interpreting the Special SimMechanics Symbols . . . . .	<b>6-12</b>
Using Special SimMechanics Functions . . . . .	<b>6-13</b>
Highlighting Bodies and Body Blocks . . . . .	<b>6-15</b>
Changing How the Machine Is Displayed . . . . .	<b>6-16</b>
Changing Perspective and Window Size . . . . .	<b>6-19</b>
Controlling the Simulation from the Window . . . . .	<b>6-20</b>
Recording and Playing Animations . . . . .	<b>6-20</b>
Saving and Recalling Display Settings . . . . .	<b>6-22</b>
Summary of the SimMechanics Toolbar . . . . .	<b>6-22</b>
<b>Viewing Machines in Virtual Reality</b> . . . . .	<b>6-24</b>
Interpreting Special Virtual Reality Symbols . . . . .	<b>6-25</b>
Changing the Observer's Viewpoint . . . . .	<b>6-26</b>
Learning More About Virtual Reality Toolbox . . . . .	<b>6-28</b>
<b>Creating Custom Machine Visualizations</b> . . . . .	<b>6-30</b>
Creating Virtual Worlds for SimMechanics Models . . . . .	<b>6-30</b>
Interfacing SimMechanics with Virtual Worlds . . . . .	<b>6-34</b>



7

<b>Finding Forces from Motions</b> .....	7-2
Closed-Loop Example: Four-Bar System .....	7-3
Open-Topology Example: Double Pendulum .....	7-7
<b>Trimming SimMechanics Models</b> .....	7-12
Unconstrained Example: Spring-Loaded Double Pendulum ..	7-12
Constrained Example: Four-Bar System .....	7-18
<b>Linearizing SimMechanics Models</b> .....	7-23
Model Linearization Example: Double Pendulum .....	7-23

SimMechanics Block Reference

8

<b>Blocks – By Category</b> .....	8-2
Creating Bodies and Grounds .....	8-4
Creating Joints .....	8-4
Constraining and Driving Motion .....	8-6
Actuating and Sensing Motion .....	8-6
Applying Force Elements .....	8-7
Additional Useful Blocks .....	8-7

**Blocks – Alphabetical List . . . . . 8-8**

**SimMechanics Command Reference**

**9**

**Glossary**

**A**

**Selected Bibliography**

**B**

# About This Guide

---

Using This Guide (p. x)

Getting Online Help (p. xii)

Opening SimMechanics Demos (p. xiii)

Typographical Conventions (p. xiv)

Mechanical Units (p. xv)

Mechanical Conventions and  
Abbreviations (p. xvi)

Overview of how to find help

Using the online help system to view documentation

Opening demo model files online

Summary of special fonts and notations

Physical units used in SimMechanics

Mechanical assumptions and abbreviations implicit in  
this user's guide

## Using This Guide

Use this guide as you work with the software to learn about the powerful capabilities of the SimMechanics.

### **Introductory Chapters**

Beginning users with limited Simulink and/or mechanical simulation experience will especially benefit by starting with the first two chapters:

- Read “Introducing SimMechanics” for an overview of features, capabilities, and functions, and to run a model from the Demos Library.
- Use the “Building and Visualizing Simple Machines” chapter to open the block library, review essential model building steps, and build simple tutorial models.

### **General Modeling and Simulation Chapters**

- Review translational and rotational motion and their SimMechanics representation in “Understanding Mechanical Concepts.”
- Learn in “Modeling Mechanical Systems” how to represent and analyze machines with block diagrams.
- Continue with “Running Mechanical Models” to set up your simulation, generate and use code, and troubleshoot errors.

### **Advanced Features Chapters**

- The “Visualizing and Animating Machines” chapter explains the SimMechanics visualization features and interfacing to virtual worlds.
- The “Case Studies” chapter elaborates on the advanced motion analysis tools of SimMechanics, including inverse dynamics and linearization.

---

## Reference Chapters

- Consult “SimMechanics Block Reference” and “SimMechanics Command Reference” for a description of each block’s operation, parameters, and characteristics, and for command line functions.
- Entries in the “Glossary” define technical terms used in this guide.
- The “Selected Bibliography” lists additional resources for the theory applicable to SimMechanics.

---

**Note** This SimMechanics User’s Guide assumes that you already have some experience with building and running models in Simulink.

---

## Getting Online Help

There are a number of easy ways to get online help as you work with SimMechanics:

- *Help Browser* – There are several ways to open the Help browser:
  - Select **Full Product Family Help** from the **MATLAB Help** menu.
  - **Select Help** from the **MATLAB View menu**.
  - Enter `doc mech` at the command line.

Use the **Contents** pane on the left of the Help browser to find a section or chapter. Use the **Search** and **Index** features to find specific words.

- *Block Library Browser* – Click **Help** on the SimMechanics library menu bar to open online Help on Simulink, blocks, shortcuts, S-functions, and demos.
- *Context-sensitive help* – To access the help for a block, right-click the block or click **Help** on the block's dialog box.
- *Command line* – Enter `doc('block name')` at the command line to access the help for a block with the name `block name`. Spaces and capitalization in the block name are ignored.

If the same block name appears in other blocksets, MATLAB returns an `Overloaded methods` warning in the Command Window to flag those instances.

- *Help desk (via the Web)* – Use a Web browser or the Help browser to connect to the MathWorks Web site at URL `http://www.mathworks.com`. Follow the **Documentation** link on the **Support** Web page for remote access to the documentation.

## For Further Help and Feedback


The MathWorks hopes that you find SimMechanics powerful and easy to use. Your suggestions and comments are welcome.

<code>support@mathworks.com</code>	Technical support
<code>suggest@mathworks.com</code>	Product enhancement suggestions
<code>bugs@mathworks.com</code>	Bug reports
<code>doc@mathworks.com</code>	Documentation error reports

For more contact and program information, visit the MathWorks Web site at `www.mathworks.com`.

## Opening SimMechanics Demos

The SimMechanics documentation makes extensive use of demo models. To open a SimMechanics demo from the Help browser, click the **Demos** tab in the **Help Navigator** pane on the left. Locate the demo in the list and open it. You can also open demos by entering the model name at the command line.

Many demos have Help links represented by the information symbol . Click on these symbols to open that demo's documentation in the Help browser.

## Typographical Conventions

This User's Guide assumes some or all of these conventions.

Item	Convention	Example
Example code	Monospace font	To assign the value 5 to A, enter <code>A = 5</code>
Function names, syntax, filenames, directory/folder names, and user input	Monospace font	The <code>cos</code> function finds the cosine of each array element. Syntax line example is <code>MLGetVar ML_var_name</code>
Buttons and keys	<b>Boldface</b> with book title caps	Press the <b>Enter</b> key.
Literal strings (in syntax descriptions in reference chapters)	<b>Monospace bold</b> for literals	<code>f = freqspace(n, 'whole')</code>
Mathematical expressions	<i>Italics</i> for variables Standard text font for functions, operators, and constants	This vector represents the polynomial $p = x^2 + 2x + 3$ .
MATLAB output	Monospace font	MATLAB responds with <code>A =</code> <code>5</code>
Menu and dialog box titles	<b>Boldface</b> with book title caps	Choose the <b>File Options</b> menu.
New terms and for emphasis	<i>Italics</i>	An <i>array</i> is an ordered collection of information.
Omitted input arguments	(...) ellipsis denotes all of the input/output arguments from preceding syntaxes.	<code>[c,ia,ib] = union(...)</code>
String variables (from a finite list)	<i>Monospace italics</i>	<code>sysc = d2c(sysd, 'method')</code>



## Mechanical Units

SimMechanics accepts any mixture of MKS (SI), cgs, and English units.

<b>Quantity</b>	<b>MKS (SI)</b>	<b>cgs</b>	<b>English</b>
Length	meter (m)	centimeter (cm)	inch (in), foot (ft)
Time	second (s)	second (s)	second (s)
Mass	kilogram (kg)	gram(g)	slug (slug)
Velocity	meters/second (m/s)	centimeters/second (cm/s)	inches/second (in/sec), feet/second (ft/sec)
Acceleration (Gravity)	meters/second <sup>2</sup> (m/s <sup>2</sup> )	centimeters/second <sup>2</sup> (cm/s <sup>2</sup> )	inches/second <sup>2</sup> (in/s <sup>2</sup> ), feet/second <sup>2</sup> (ft/s <sup>2</sup> )
Force	Newton (N)	dyne (dyn)	pound (lb)
Angle	radian (rad), degree (deg)	radian (rad), degree (deg)	radian (rad), degree (deg)
Inertia	kilogram-meter <sup>2</sup> (kg-m <sup>2</sup> )	gram-centimeter <sup>2</sup> (g-cm <sup>2</sup> )	slug-foot <sup>2</sup> (slug-ft <sup>2</sup> )
Angular velocity	radians/second (rad/s), degrees/second (deg/s)	radians/second (rad/s), degrees/second (deg/s)	radians/second (rad/s), degrees/second (deg/s)
Angular acceleration	radians/second <sup>2</sup> (rad/s <sup>2</sup> ), degrees/second <sup>2</sup> (deg/s <sup>2</sup> )	radians/second <sup>2</sup> (rad/s <sup>2</sup> ), degrees/second <sup>2</sup> (deg/s <sup>2</sup> )	radians/second <sup>2</sup> (rad/s <sup>2</sup> ), degrees/second <sup>2</sup> (deg/s <sup>2</sup> )
Torque	Newton-meter (N-m)	dyne-centimeter (dyn-cm)	pound-foot (lb-ft)

## Mechanical Conventions and Abbreviations

### Right-Hand Rule Is Assumed

For rotational motion and vector cross products  $\mathbf{a} \times \mathbf{b}$ , the right-hand (RH) rule is always assumed.

### Vector Multiplication

Scalar vector products and matrix-vector multiplication are denoted by  $\mathbf{a} \cdot \mathbf{b}$  and  $M \cdot \mathbf{v}$ , respectively.

### Common Abbreviations

These are the abbreviations of mechanical terms most commonly used in this guide:

<b>Abbreviation</b>	<b>Meaning</b>
<i>CG</i>	center of gravity
<i>CS</i>	coordinate system
<i>DoF</i>	degree of freedom
<i>RF</i>	reference frame

### Glossary Terms

Special mechanical or SimMechanics terms are frequently hyperlinked to entries in the “Glossary”

# Introducing SimMechanics

---

SimMechanics is a tool to model mechanical systems for use with Simulink® and MATLAB®.

Welcome to SimMechanics (p. 1-2)	Introduction to SimMechanics and the Physical Modeling environment
Related Products (p. 1-3)	Products you might want to use with SimMechanics and requirements for virtual reality visualization
Running a Demo Model (p. 1-5)	A simple conveyor showing how to use SimMechanics blocks in the Simulink environment
What Can You Do with SimMechanics? (p. 1-17)	Survey of what SimMechanics does, including special simulation features

## Welcome to SimMechanics

SimMechanics is a block diagram modeling environment for the engineering design and simulation of rigid body machines and their motions, using the standard Newtonian dynamics of forces and torques.

With SimMechanics, you can model and simulate mechanical systems with a suite of tools to specify bodies and their mass properties, their possible motions, kinematic constraints, coordinate systems, and to initiate and measure body motions. You represent a mechanical system by a connected block diagram, like other Simulink models, and you can incorporate hierarchical subsystems.

The internal visualization tools of SimMechanics display and animate simplified representations of 3-D machines, before and during simulation, in these environments:

- MATLAB Handle Graphics® window
- Virtual world rendered in a virtual reality viewer

## SimMechanics and Physical Modeling

SimMechanics is part of Simulink Physical Modeling, encompassing the modeling and design of systems according to basic physical principles. Physical Modeling runs within the Simulink environment and interfaces seamlessly with the rest of Simulink and with MATLAB. Unlike other Simulink blocks, which represent mathematical operations or operate on signals, Physical Modeling blocks represent physical components or relationships directly.

## What's in This Chapter

This chapter introduces you to the capabilities of SimMechanics and its relationship to other MathWorks products:

- “Related Products” on page 1-3
- “Running a Demo Model” on page 1-5
- “What Can You Do with SimMechanics?” on page 1-17

## Related Products

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with SimMechanics.

### Requirements for SimMechanics

You must have the following products installed to use SimMechanics:

- MATLAB 6.5
- Simulink 5.0

### Virtual Reality-Based Visualization

The optional virtual reality-based visualization in SimMechanics requires the Virtual Reality Toolbox Version 3.0. The toolbox ships with its own virtual reality viewer.

You can improve virtual reality speed and graphics resolution by adding a graphics accelerator hardware card to your system. Animation of simulations is sensitive to central processor and graphics card speed and memory.

Experiment to find a reasonable compromise between quality and speed for your system.

### AVI Support for Recorded Handle Graphics Animations

You can record simulation animations in Microsoft Audio Video Interleave® (AVI) format using the built-in Handle Graphics visualization feature of SimMechanics. To play these recorded AVI files, you need an AVI-compatible video player. MATLAB has a built-in movie player compatible with AVI.

You can also use an external AVI-compatible player, such as Microsoft's Windows Media Player, Apple's QuickTime, or RealNetworks' RealOne Player.

### Other Related Products

The toolboxes listed in the following table include functions that extend the capabilities of MATLAB. The blocksets include blocks that extend the capabilities of Simulink. These products will enhance your use of SimMechanics in various applications.

## The Physical Modeling Product Family

In addition to SimMechanics, the Physical Modeling product family includes SimPowerSystems, for modeling and simulating electrical power systems. Use these products together to model physical systems in Simulink.

For more information about any MathWorks software products, see either

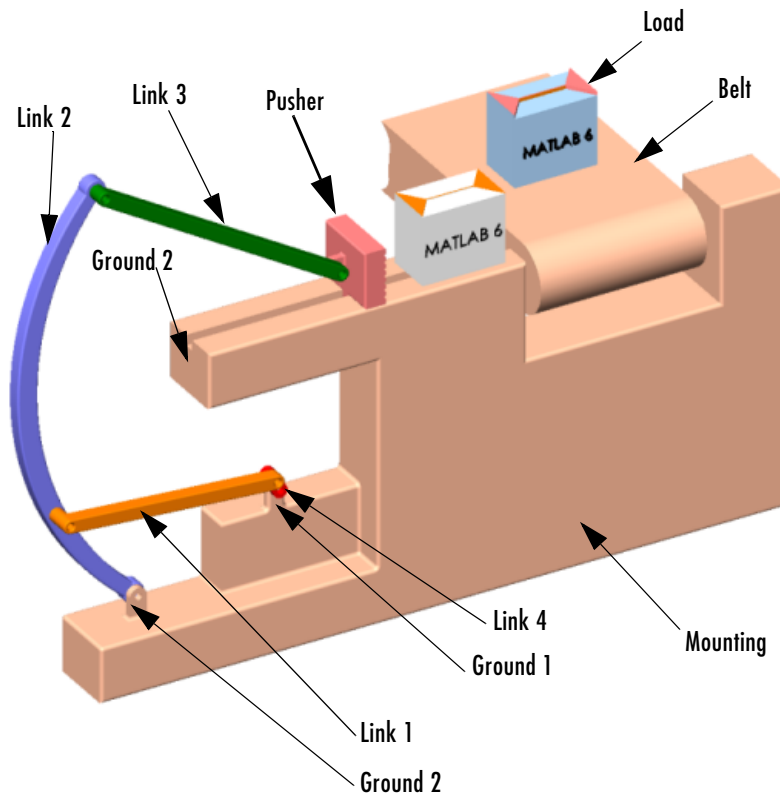
- The online documentation for that product if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site at [www.mathworks.com](http://www.mathworks.com); see the “Products” section.

<b>Product</b>	<b>Description</b>
Control System Toolbox	Design and analyze feedback control systems
Optimization Toolbox	Solve standard and large-scale optimization problems
Real-Time Workshop <sup>®</sup>	Generate C code from Simulink models
SimPowerSystems	Model and simulate electrical power systems
Simulink	Design and simulate continuous- and discrete-time systems
Stateflow <sup>®</sup>	Design and simulate event-driven systems
Virtual Reality Toolbox	Create and manipulate virtual reality worlds from within MATLAB and Simulink
xPC Target	Perform real-time rapid prototyping using PC hardware

## Running a Demo Model

This demo model uses a few blocks in the library to simulate a simple machine with feedback control. You will see how SimMechanics implements the model in conjunction with standard Simulink features.

The demo model simulates a conveyor belt loading mechanism. A simple controller (not shown), with a sensor and an actuator, guides the mechanism with a saturation limit and anti-windup logic for the applied torque. The controller is user-adjustable and sets the stopping point for the pusher.



**Conveyor Loader Mechanism**

## What This Demo Illustrates

The conveyor mechanism demo illustrates some important features of SimMechanics:

- Representing bodies and degrees of freedom with Body and Joint blocks, respectively
- Using SimMechanics blocks with normal Simulink blocks
- Feeding in and feeding out Simulink signals to and from SimMechanics blocks with Actuator and Sensor blocks, respectively
- Encapsulating groups of blocks into subsystems
- Visualizing and animating a machine by its component bodies

---

**Caution** You might want to make modifications to this demo model. To avoid errors, be sure

- Not to attempt connecting Simulink signal lines directly to SimMechanics blocks other than Actuators and Sensors
- To keep the collocation of the Body coordinate system origins on either side of each assembled Joint to within assembly tolerances

Saving modified demo models in a different directory from the demos is recommended.

---

## Opening the Model

You can open the demo model in several ways. Here is the general procedure for starting SimMechanics demos in the Launch Pad area of the MATLAB desktop:

- 1 Open the SimMechanics entry.
- 2 Double-click the **Demos** subentry in the tree. This opens the MATLAB Help browser with **Demos** selected in the left **Help Navigator** pane.
- 3 Double-click **Conveyor Mechanism** from the list of models in the list on the left or on the right.



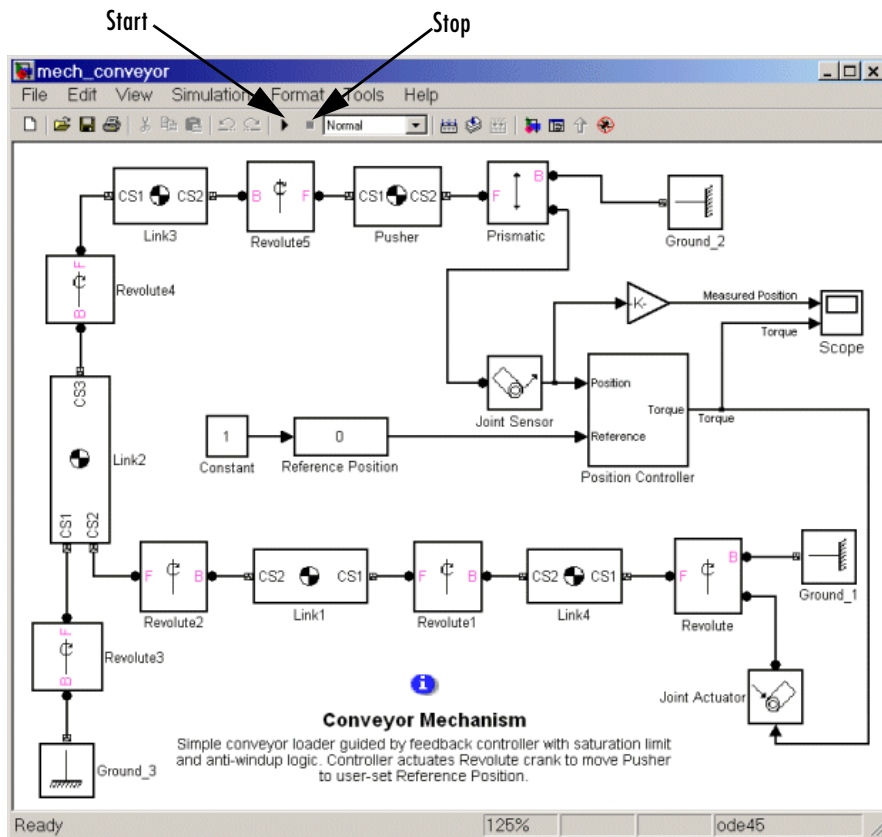
Alternatively, you can open the same MATLAB **Demos** window by entering demos at the MATLAB command line.

To get started quickly with this specific demo, you can use either of these steps:

- Enter mech\_conveyor at the MATLAB command line.
- Online Help users can click mech\_conveyor here.

## The Block Diagram Model

The block diagram model opens in a model window.

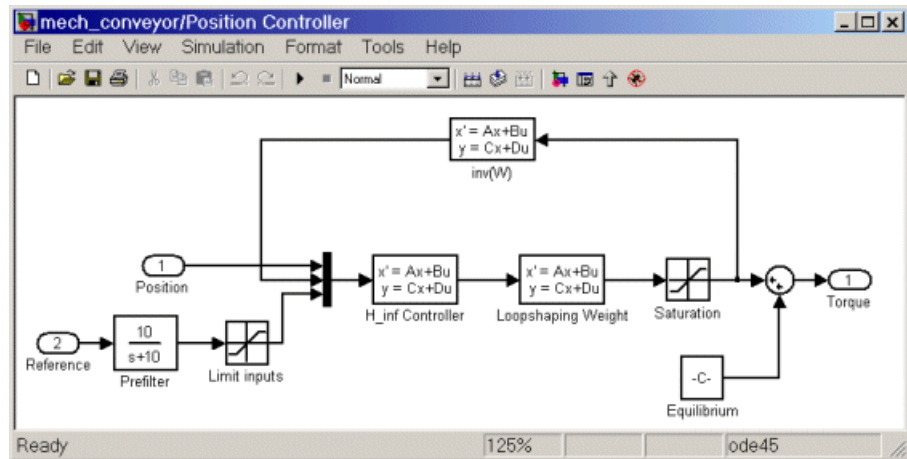


At the same time, a Scope block also opens with two plots, labeled Measured Position and Torque, respectively.

## What the Model Contains

Note some features of the model:

- Ignore the Position Controller, Joint Sensor, and Joint Actuator blocks for a moment. Note that the loading mechanism follows the tree of bodies and joints shown in the Conveyor Loader Mechanism figure on page 1-5:
  - There are four rotating link bodies and one sliding pusher body, as well as three ground points on the immobile mounting represented by Ground blocks. Double-click the Body and Ground blocks to see their dialog boxes.
  - The pusher slides and the links rotate relative to one another and to the ground points on the mounting. There are seven apparent degrees of freedom (DoFs) in the machine, represented by seven Joints, but the geometry constrains the motion to one actual DoF. Double-click the Revolute blocks to see how rotational DoFs are expressed in their dialog boxes.
  - The Prismatic block expresses the linear motion of Pusher relative to Ground\_2. The Revolute block expresses the angular motion of Link4 (the crank of the whole mechanism) relative to Ground\_1.
- The Joint Sensor detects the position of Pusher via the Prismatic block. The Joint Actuator applies torque to Link4 via the Revolute block. Double-click the Sensor and Actuator blocks to view how the machine motions and forces/torques are transformed into Simulink signals.
- The Position Controller subsystem converts the Pusher position information into a feedback signal to actuate Revolute and thus Link4. You can open the Position Controller block to view this subsystem, which is made of normal Simulink blocks.



- The Reference Position block gives you control over the stopping position of the pusher by modulating the control signal that actuates Revolute. Maintaining the initial pusher position requires a fixed torque on Revolute.
- In the Scope block, you can view both the Pusher position in millimeters (mm) relative to Ground\_2 as the Measured Position plot and the torque in Newton-meters (N-m) applied to Link4 relative to Ground\_1 as the Torque plot.

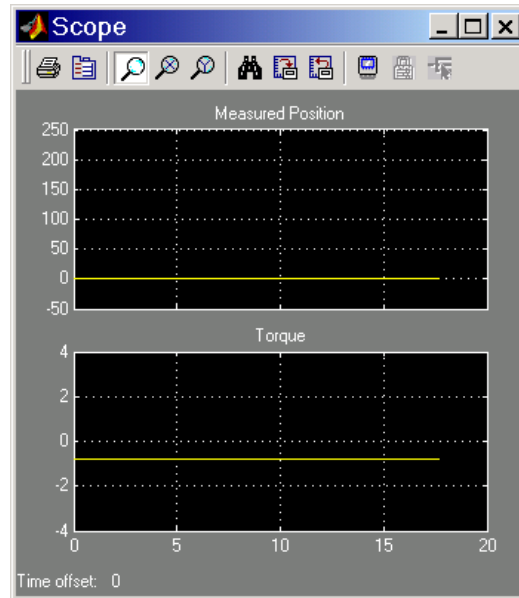
## Starting the Demo

You can now run the model as it is when you first open it:

- 1 In the **Simulation** menu, select **Simulation parameters**. The **Simulation parameters** dialog box appears:
  - a The default **Stop time** is  $\text{inf}$ , so the simulation keeps running once you start it. You should leave it at  $\text{inf}$  and stop the simulation manually the first few times you run it.  
 Later you can apply a finite stop time (in seconds) if you want.
  - b Leave the **Solver options** entries at default values and close the box.

- 2 From the **Simulation** menu, select **Start**. In Microsoft Windows, you can also click the **Start** button in the model window toolbar.

The measured position of the pusher and the torque applied to maintain that position start and remain essentially constant in the Scope plots. The applied torque is adjusted to maintain the initial pusher position.



- 3 To see greater detail at the simulation start, stop the simulation before the time passes 20 seconds and zoom in on the Scope plots.

## Modifying the Model

Here are two modifications of the demo you can try. One illustrates the simple user-driven controller you can adjust to change the motion of the pusher. The other illustrates a powerful feature of SimMechanics, visualization of a machine and animation of its simulated motion.

To make these modifications, it is best to close and restart the demo.

## Changing the Pusher Reference Position

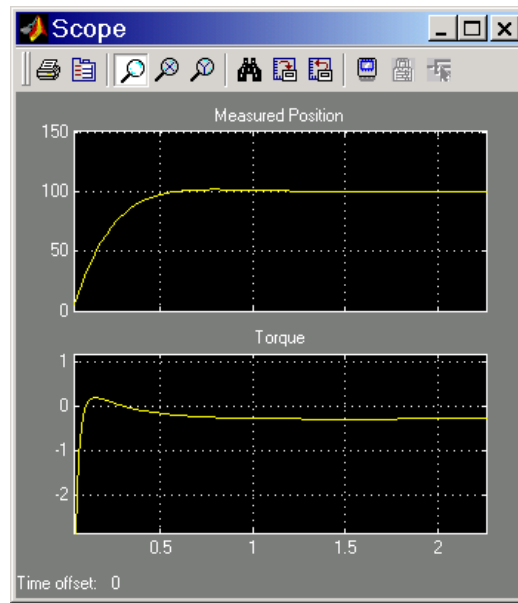
The Reference Position block is actually a Simulink Slider Gain block (from the Simulink Math Library) and controls where the pusher comes to rest.

You can adjust the Reference Position block to change where the pusher stops:

- 1** Open the Reference Position block. You see an adjustable slider to set the position of the pusher's rest point.
- 2** Enter values in the **Low** and **High** fields to set the lower and upper limits of the allowed slider range. The defaults in this demo are 0 and 0.2, with implied units of meters (m).
- 3** Enter a value in the central field to set the pusher stopping point, which you can also adjust by clicking and dragging the slider between the lower and upper limits. The default is 0 (meters).

You can apply changes to the reference position to the simulation in two ways:

- Reset the Reference Position block first, then start the demo. You see the pusher trajectory track differently now, toward the new stopping point. For example, resetting the Reference Position to 0.1 and restarting the demo produces these Scope plots, with **Autoscale** and zooming applied. The asymptotic measured position now tends to 100 mm (0.1 m), and the torque applied to keep the pusher there has changed:



- Start the demo with the Reference Position block open and move the slider up and down as the simulation runs. Watch the Scope. The measured position and the necessary torque adjust to follow the new reference position.

## Visualizing and Animating the Conveyor

You can visualize the conveyor mechanism as a static machine and animate the simulation as well with the two visualization tools:

- One tool, based on MATLAB Handle Graphics, is built into SimMechanics.
- The other tool requires the Virtual Reality Toolbox.

With either visualization tool, you can display the bodies of the machine in two possible abstract representations:

- *Equivalent ellipsoids* use the inertia tensors and masses of the bodies. Each body has a unique homogeneous ellipsoid equivalent to it in mass and inertia tensor.
- *Convex hulls* use the attached Body coordinate systems (CSs) of the bodies. A body must have at least four non-coplanar Body CS origins to enclose a

convex hull with nonzero volume. If the Body has fewer than four Body CS origins or if the origins are coplanar or collinear or coincident, the visualization tools render it with simpler figures (triangle, line, or point).

**Handle Graphics Visualization.** First try visualizing the conveyor with Handle Graphics:

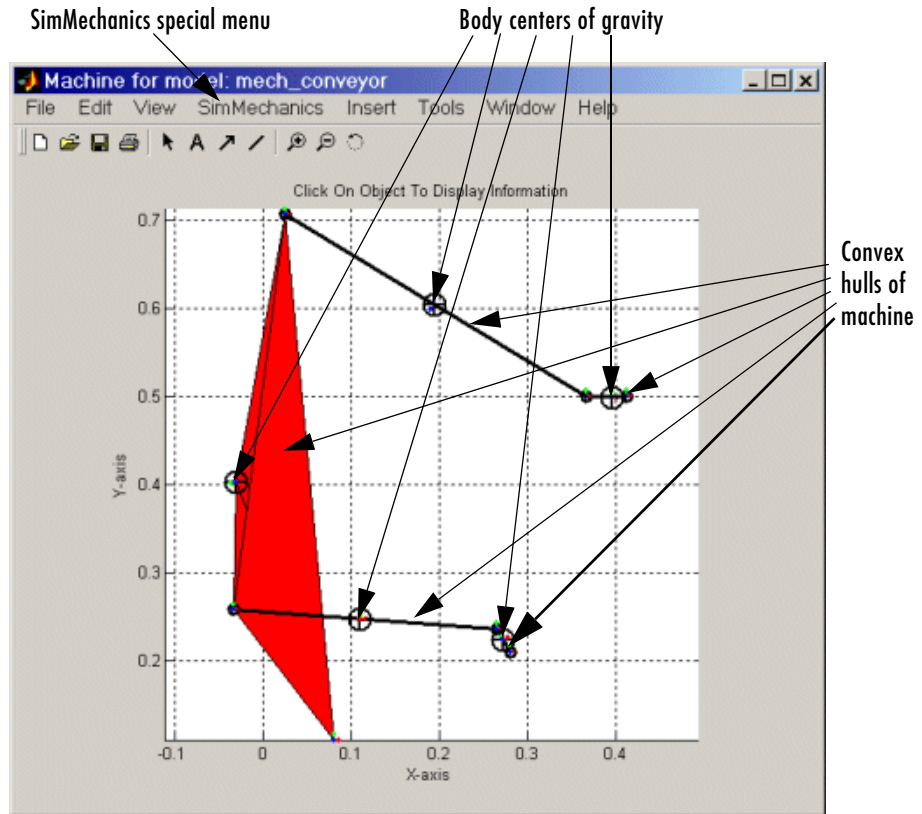
- 1** From the **Simulation** menu, select **Mechanical environment**. The **Mechanical Environment Settings** dialog box appears.
- 2** Click the **Visualization** tab. Select the **Draw machine in initial state** and **Animate machine during simulation** check boxes.
- 3** Leave the other defaults as they are and close the dialog.

<b>Visualization Control</b>	<b>Default Value</b>
<b>Draw machine using</b>	MATLAB Graphics
<b>Represent bodies as</b>	Convex hulls
<b>Update machine</b>	When diagram changes

A MATLAB Handle Graphics window appears, displaying the conveyor machine at rest in its initial state.

The bodies are rendered in the default representation, as convex hulls. The bodies and Body coordinate system axis triads are also displayed as defaults.

- 4** Change Reference Position to a nonzero value such as 0.1 or 0.2.
- 5** Restart the simulation. The Handle Graphics window animates the machine in motion. You can compare this motion to the plots in the scope.



6 Click a body in the Handle Graphics window. The model window comes back into focus with the corresponding Body block highlighted in color.

7 Open the special **SimMechanics** menu in the Handle Graphics menu bar.

Here, you can reconfigure the special display properties for machines: bodies, Body CS axis triads, colored fill-in patches connecting Body CSs on the same body, and user viewpoint orientation.

8 Close the Handle Graphics window when you are finished.

**Virtual Reality Visualization.** Now visualize the conveyor in virtual reality scene:



- 1** From the **Simulation** menu, select **Mechanical environment**. The **Mechanical Environment Settings** dialog box appears.
- 2** Click the **Visualization** tab. Select the **Draw machine in initial state** and **Animate machine during simulation** check boxes.
- 3** Using the pull-down menus, change **Draw machine using** to Virtual Reality Toolbox and **Represent bodies as** to Equivalent ellipsoids.

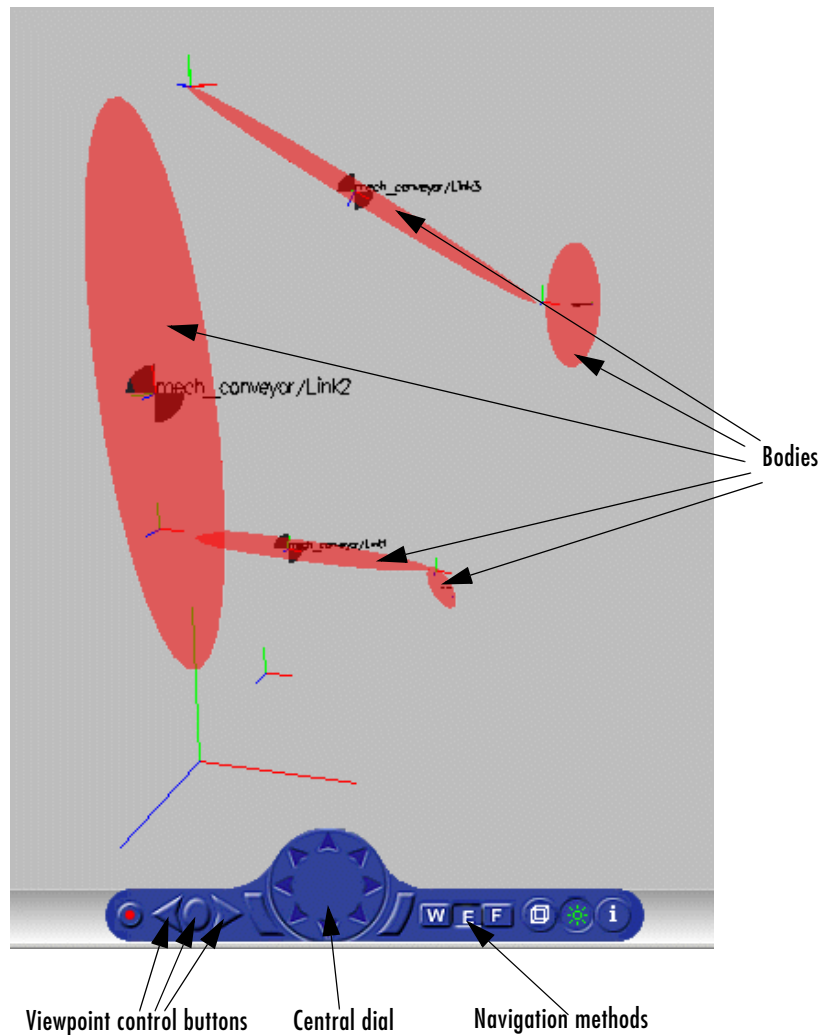
Leave **Update machine** as When diagram changes.

A virtual reality viewer opens, displaying the conveyor machine at rest in its initial state.

The bodies are rendered as equivalent ellipsoids.

- 4** Restart the simulation. The viewer now animates the machine in motion.
- 5** Use the virtual reality controls at the bottom of the virtual scene to change the user viewing position and orientation:
  - Moving the central dial pointer around the outside of the dial circle, you can change the direction of your movement. Pressing the dial pointer itself, you then move in that direction.
  - Left-clicking and rolling your mouse in the scene changes your viewpoint.
  - Right-clicking in the scene opens the control menu.

See “Viewing Machines in Virtual Reality” on page 6-24 for how to control the virtual scene.



- 6 While the animation is running, open the Reference Position block and move the slider up and down. In addition to what you can see in the Scope plots, the viewer directly animates the pusher trajectory in space as the mechanism responds to your adjustment.

## What Can You Do with SimMechanics?

SimMechanics is a set of block libraries and special simulation features for use in the Simulink environment. You connect SimMechanics blocks to normal Simulink blocks through special Sensor and Actuator blocks.

The blocks in these libraries are the elements of a platform for modeling mechanical systems consisting of any number of rigid bodies, connected by joints representing translational and rotational degrees of freedom. SimMechanics can represent machines with components organized into hierarchical subsystems, as in normal Simulink models. You can impose kinematic constraints, apply forces/torques, integrate Newton's equations, and measure resulting motions. You saw some of these features at work in the Conveyor Loader demo model.

---

**Glossary Terms** For an explanation of special terms, see “Summary of Technical Vocabulary” on page 3-11 and the “Glossary.”

---

## Modeling Machines with SimMechanics

SimMechanics extends Simulink with a library of blocks for specifying a mechanical system's components and properties and solving the system's equations of motion. The blocks are similar to other Simulink blocksets, with some properties unique to SimMechanics.

These are the major steps you follow, using SimMechanics, to build and run a model representation of a machine:

- Specify body inertial properties, degrees of freedom, and constraints, along with coordinate systems attached to bodies to measure positions and velocities.
- Set up sensors and actuators to record and initiate body motions, as well as apply forces/torques.
- Start the simulation, calling the Simulink solvers to find the motions of the system, while maintaining any imposed constraints.
- Visualize the machine while building the model and animate the simulation while running, using the Handle Graphics or virtual reality-based visualization tool.

## Bodies, Coordinate Systems, Joints, and Constraints

SimMechanics supports user-defined Body blocks specified by their masses and inertia tensors. You connect the bodies to one another with joints representing the possible motions of bodies relative to one another, the system's degrees of freedom (DoFs). You can impose kinematic constraints on the allowed relative motions of the system's bodies. These constraints impose restrictions on body DoFs or drive body motions as explicit functions of time.

The SimMechanics interface gives you many ways to specify coordinate systems (CSs), constraints/drivers, and forces/torques. You can

- Attach coordinate systems (Body CSs) to different points on various Body blocks to specify local axes and origins for actuating and sensing
- Take composite Joint blocks from the SimMechanics library or extend the existing Joint library by constructing your own composite Joints from primitive Joint blocks
- Use other Simulink tools as well as MATLAB expressions in the SimMechanics environment

### User-Defined Local Coordinate Systems

SimMechanics automatically sets up a single absolute inertial reference frame and coordinate system (CS) called *World*. You can also set up your own Local CSs:

- *Grounded CSs* attached to Ground blocks at rest in the World RF but displaced from the World CS origin
- *Body CSs* fixed on the system's rigid bodies and moving rigidly with the bodies

### Constraint Solver Types

Specifying functional algebraic or kinematic relations between any two bodies, you can constrain the motion of the system by connecting Constraint blocks to pairs of Bodies. Connecting Driver blocks applies time-dependent constraints.

The applied constraints are interpreted in one of three *constraint solver types*: *tolerancing*, *machine precision*, or *stabilizing* solvers.

## Sensors, Actuators, and Force Elements

Sensors and Actuators are the blocks you use to interface between non-SimMechanics Simulink blocks and SimMechanics blocks. Force Elements represent internal forces that require no external input.

- Sensor blocks detect the motion of Bodies and Joints.
  - Sensor block outputs are Simulink signals that you can use like any other Simulink signal. You can connect a Sensor block to a Simulink Scope block and display the motions in a system, such as positions, velocities, and accelerations, as functions of time.
  - You can feed these Sensor output signals back to a SimMechanics system to specify forces/torques in the system, via Actuator blocks.
- Actuator blocks specify the motions of Bodies or Joints.
  - They accept force/torque signals from Simulink and can apply forces/torques on a body or joint from these signals. The Simulink signals can include Sensor block outputs fed back from the system itself.
  - They detect discrete locking and unlocking of Joints to implement discontinuous friction forces.
  - They specify the position, velocity and acceleration of bodies or joints as explicit functions of time.
  - They prepare a system's initial kinematic state (positions and velocities) for forward integration of the Newtonian dynamics.

Force Elements model internal forces between bodies or acting on joints between bodies. Internal forces depend only on the positions and velocities of the bodies themselves, independent of external signals.

## Simulating Mechanical Motion

### SimMechanics Analysis Modes

SimMechanics provides four modes for analyzing the mechanical systems you simulate:

- Forward dynamic analysis integrates applied forces/torques, maintaining imposed constraints, and obtains resulting motions.
- Inverse dynamic analysis finds the forces/torques necessary to produce user-specified motions in topologically open systems.

- Kinematic analysis finds the forces/torques necessary to produce user-specified motions in topologically closed (loop) systems.
- Trimming analysis searches for steady or equilibrium states of a system.

In the Conveyor Loader demo model, you analyzed the motion in the Forward Dynamics mode. The model specified forces/torques and initial conditions, then integrated Newton's laws to obtain the machine's motion.

**Mathematical Conditions for Rigid Body Motion to Be Determined.** For the Forward Dynamics problem to be mathematically solvable, the system must satisfy certain conditions:

- The masses and inertia tensors of all bodies are known.
- All forces and torques acting on each body at each instant of time are known.
- Any constraints among DoFs are specified as constraints among positions and/or velocities alone. These are kinematic constraints; that is, zeroth- and first-order differential constraints. If the constraints are mutually consistent and are fewer in number than the DoFs, the system's motion is nontrivial and can be found by integration.
- Initial conditions (initial positions and velocities) are specified and consistent with all constraints.

In Inverse Dynamics or Kinematics analysis modes, you specify the motions instead and obtain the forces/torques needed to produce those motions.

## Forward Dynamics

In the Forward Dynamics mode, SimMechanics uses the Simulink suite of ordinary differential equation (ODE) solvers to solve the mechanical ODEs (Newton's equations). The ODE solvers project the motion of the DoFs onto the mathematical manifold of the kinematic constraints and yield the forces/torques of constraint acting within the system.

You can also use the Simulink linearization tool to linearize the forward motion of a system and obtain its response to small perturbations in forces/torques, constraints, and/or initial conditions.

## Inverse Dynamics

SimMechanics can solve the reverse of the forward dynamics problem: instead of starting with given forces/torques and finding the resulting motions, the

Inverse Dynamics mode determines the forces/torques needed to produce a given set of motions that you apply to the machine. This mode only works with *open* topology systems (model diagrams without closed loops).

### **Kinematics**

You cannot analyze machines represented by model diagrams with closed topology (models with loops) using the Inverse Dynamics mode. The *Kinematics* mode analyzes the motion of *closed-loop* models, including the extra internal invisible constraints arising from loop closures.

You also use the Kinematics mode to determine the forces/torques needed to produce a given set of motions applied to a closed-loop machine model.

Constraints can only appear in closed loops, so you use the Kinematics mode to analyze constraint forces/torques as well.

### **Trimming**

Finally, the Trimming mode searches for steady or equilibrium states in a system's motion using the Simulink `trim` command. The states, once found, are the starting point for linearization analysis using the Simulink `linmod` and `dlinmod` commands.

## **Visualizing and Animating Machines**

---

**Virtual Reality Visualization** The SimMechanics virtual reality visualization feature requires the optional Virtual Reality Toolbox. Refer to the Virtual Reality Toolbox User's Guide for full information on installing this toolbox.

You can also animate more realistic renderings of bodies by creating your own virtual world with the Virtual Reality Toolbox. Animating the motion of your custom virtual bodies with sensor signals requires building your own interface from SimMechanics to your virtual world. See "Creating Custom Machine Visualizations" on page 6-30.

We recommend you allocate generous central processor power, graphics card speed, and memory for the virtual reality feature, especially for animation.

---

SimMechanics supports two internal visualization/animation tools, powerful aids in building, animating, and debugging machines that you saw in the Conveyor Loader demo:

- The Handle Graphics-based visualization tool built into SimMechanics. The machine is displayed in a Handle Graphics window. You can use all the standard Handle Graphics functions to change the viewer perspective. The window also has options special to SimMechanics.
- An optional visualization tool, called internally by SimMechanics, based on the Virtual Reality Toolbox. The machine is displayed in a virtual scene through a virtual reality viewer. You can change user perspective in the virtual scene with the viewer controls, but you can change the bodies' properties only by changing their blocks in the model window.

Both tools display the bodies and their Body coordinate systems (CSs) in an abstract, simplified form. You can render the bodies as convex hulls or as equivalent ellipsoids.

## Visualizing Bodies During Machine Building

One way to use the visualization tool is while you're building your machine:

- You can open a virtual world or Handle Graphics window before you start to build and then watch the bodies appear and be configured in the display as you create and configure them in your model window.
- You can also build a model without visualization, then open a virtual scene or Handle Graphics window when you're done to see the completed machine.

The first approach is especially useful if you're just starting to learn how to use SimMechanics and/or modeling complex machines. In that case, visualization can guide you in assembling the body geometries and connections.

## Representing Bodies

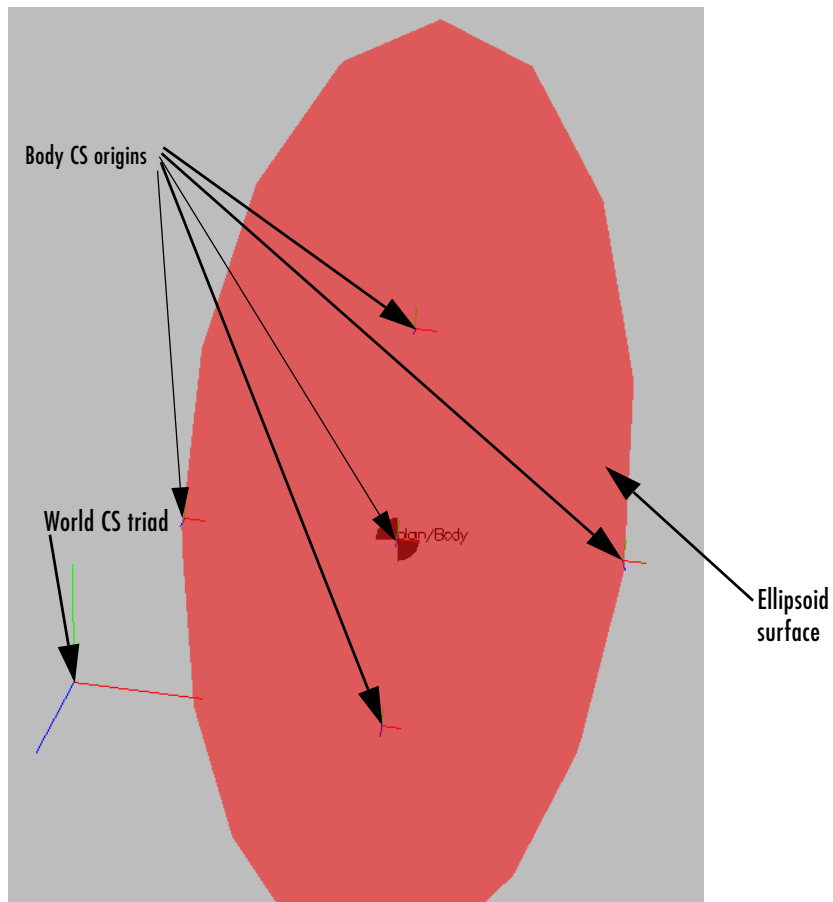
The visualization tools display your machine bodies in a virtual world or Handle Graphics window. The tools have two abstract shapes to represent the bodies, one derived from body mass properties, the other from bodies' attached Body coordinate systems (CSs). These shapes are geometric schematics, because SimMechanics accepts only limited body information:



**Mass properties.** A rigid body's dynamics are partly determined by the body's total mass and how that mass is distributed in space, as encapsulated in its *inertia tensor*. Any rigid body has a unique corresponding homogeneous ellipsoid with the same mass and inertia tensor.

Using these equivalent ellipsoids is one visualization mode of representing a body in space. The relative sizes of the ellipsoid axes indicate the relative inertial moments about each axis.

Here is a rigid body represented by its virtual reality equivalent ellipsoid.

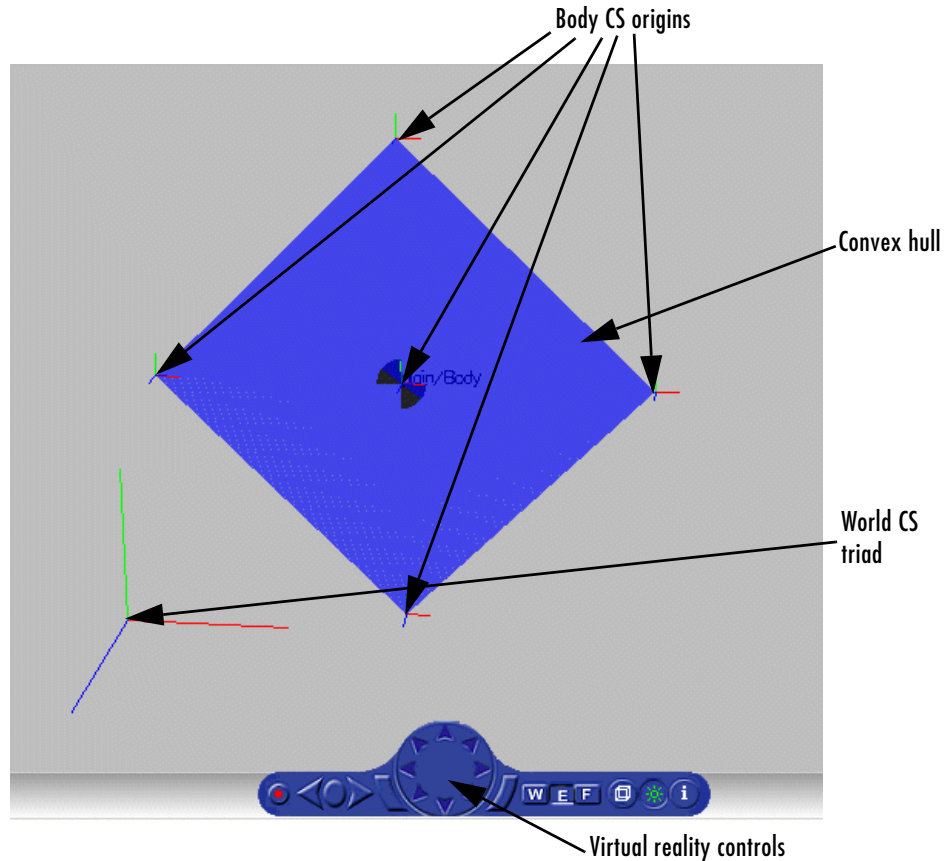


**Geometric properties.** In SimMechanics, every body is represented by a Body block with at least one attached Body CS. The minimum Body CS origin is located at the body's center of gravity (CG).

Other Body CSs can also be attached to a Body. In particular, any Joint, Constraint/Driver, Actuator, or Sensor attached to a Body must be attached at a Body CS origin.

The set of Body CS origins can be enveloped by a surface; if there are more than three non-coplanar origins, the surface encloses a volume. The minimal surface with outward-bending curvature enveloping this set is the convex hull, which becomes the other abstract shape for visualizing a body in space. Fewer than four CS origins produce simpler Body figures.

Here is the same body as a convex hull in a virtual scene. The Body CS origins are coplanar in this case, and the hull is two triangles.



### Animating Machine Motion During Simulation

Besides rendering your machine bodies either while you build a model or as a completed model, you can also keep the visualization tools open while a model is running in the Simulink model window. The Handle Graphics window or virtual reality viewer animates the simulation of the bodies' motions, whether you choose to render the bodies as ellipsoids or as convex hulls, and moves in parallel with however the model changes on the Simulink side.



# Building and Visualizing Simple Machines

---

Constructing simple mechanical models with SimMechanics is easy to learn if you already know how to make Simulink models. If you are not already familiar with Simulink, please see the Simulink documentation.

Introducing the SimMechanics Block Libraries (p. 2-2)	Overview of the SimMechanics block libraries for representing machine components
Creating SimMechanics Models (p. 2-7)	Summary of the most important steps for representing a machine by a SimMechanics model
Building a Simple Pendulum (p. 2-11)	A beginning tutorial to model and simulate a simple one-degree-of-freedom system
Visualizing a Simple Pendulum (p. 2-30)	A tutorial that shows how to visualize a simple machine with SimMechanics
A Four Bar Mechanism (p. 2-36)	A more advanced tutorial that models and simulates a closed-loop machine

### Introducing the SimMechanics Block Libraries

SimMechanics is organized into hierarchical libraries of closely related blocks. The next section, “Viewing the Blocks,” shows how to view these libraries and gives you a summary of what they contain:

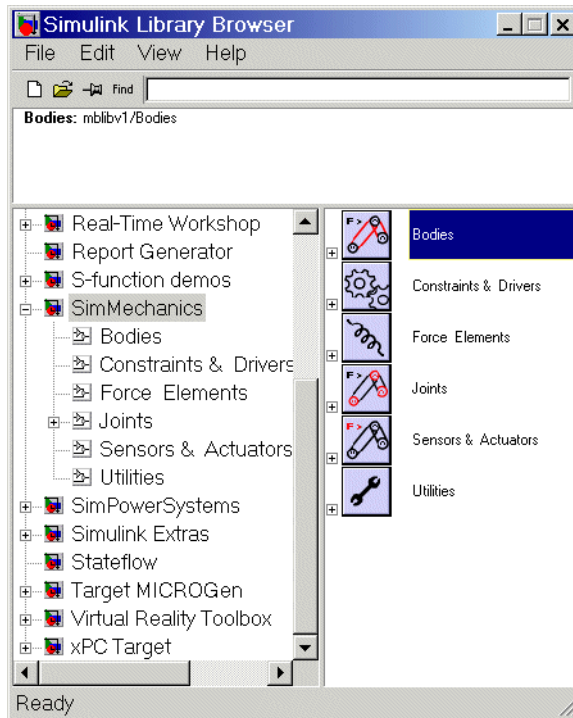
- “Bodies Library” on page 2-4
- “Joints Library” on page 2-5
- “Constraints & Drivers Library” on page 2-5
- “Sensors & Actuators Library” on page 2-5
- “Force Elements Library” on page 2-6
- “Utilities Library” on page 2-6
- “Demos Library” on page 2-6

### Viewing the Blocks

There are several ways to get to the top-level SimMechanics library on Microsoft Windows and UNIX platforms.

## Microsoft Windows Platforms

Microsoft Windows users can access the blocks through the Simulink Library Browser. Expand the SimMechanics entry in the contents tree.



You can also access the blocks directly inside the SimMechanics library in several ways:

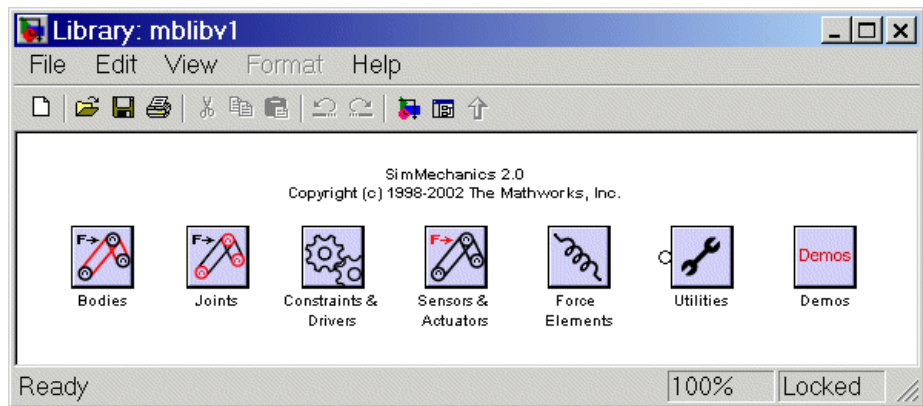
- In the Simulink Library Browser, right-click the SimMechanics entry and select **Open the 'SimMechanics' Library**. The library appears.
- Click the SimMechanics entry in the Launch Pad area to expand its tree, and then open the Block Library subentry.
- Enter `mechlib` at the MATLAB command line prompt.

### UNIX Platforms

UNIX users can click the Simulink icon on the MATLAB menu bar, open the Blocksets & Toolboxes library and then SimMechanics. You can also enter `mechlib` at the command line.

### The SimMechanics Library

Once you perform one of these preceding steps, the SimMechanics library opens.



---

**Note** This library displays six top-level block groups. You can expand each library by double-clicking its icon. The Joints library contains two second-level sublibraries. The Demos library opens the demos list in the Help browser.

---

The following sections summarize the blocks in each library. For an explanation of special terms, see “Summary of Technical Vocabulary” on page 3-11 and the “Glossary.” You can view the general SimMechanics reference in the “SimMechanics Block Reference.”

### Bodies Library

The Bodies library provides the Body block for representing user-defined bodies by their mass properties (masses and inertia tensors), their positions and orientations, and their attached Body coordinate systems (CSs). This



library also contains the Ground block representing immobile ground points, which have their own Grounded CSs.

### **Joints Library**

The Joints library provides the blocks to represent the relative motions between bodies as degrees of freedom (DoFs). The library is made up of assembled Joints listed individually and two sublibraries of specialized Joint blocks.

An assembled joint restricts the Body CSs on the two bodies to which it is connected. The assembled Joints are the primitive Prismatic, Revolute, and Spherical blocks and ready-made composite Joints. Unless it is explicitly labeled as disassembled, you can assume a generic Joint block is assembled.

**Joints/Disassembled Joints Sublibrary.** The Disassembled Joints sublibrary provides blocks for disassembled joints, special joints that do not restrict the Body CSs on the two connected bodies or the DoF axes of the two bodies. You can only use Disassembled Joints to close a loop in your machine. You cannot sense or actuate Disassembled Joints.

**Joints/Massless Connectors Sublibrary.** The Massless Connectors sublibrary provides blocks for massless connectors, composite joints whose DoFs are separated by a fixed distance. You cannot actuate or sense Massless Connectors.

### **Constraints & Drivers Library**

The Constraints & Drivers library provides blocks to specify prior restrictions on DoFs between Bodies. These restrictions can be time-independent constraints or time-dependent driving of DoFs with Simulink signals.

### **Sensors & Actuators Library**

The Sensors & Actuators library provides blocks for sensing and initiating the motions of joints and bodies. These blocks play a special role in connecting SimMechanics blocks to other Simulink blocks, as described in “Connecting SimMechanics Blocks” on page 4-3, “Modeling Actuators” on page 4-40, and “Modeling Sensors” on page 4-60.

### **Force Elements Library**

The Force Elements library provides blocks for creating forces or torques between bodies. These blocks model forces internal to your machine.

### **Utilities Library**

The Utilities library contains miscellaneous blocks useful in building models.

### **Demos Library**

The Demos library contains prewritten Simulink demonstration models using SimMechanics and other Simulink blocks, as well as Stateflow blocks. Double-clicking the Demos library icon calls the Help browser and displays the SimMechanics demos list.

In the Help browser, click the **Demos** tab in the **Help Navigator** pane to the left. To see the demos list, expand the Simulink entry, then click the SimMechanics subentry. Double-click a demo model name in the list to open it.

## Creating SimMechanics Models

To become comfortable building mechanical models, you might find it helpful to work through the guided examples in subsequent sections of how to configure and put together elements of SimMechanics to simulate simple machines. This section gives you an overview of the model-building process before you start:

- “Essential Steps to Build a Model”
- “Essential Steps to Configure and Run a Model” on page 2-9

The most important special terms used in this guide are summarized in “Summary of Technical Vocabulary” on page 3-11.

### Essential Steps to Build a Model

You use the same basic procedure for building a SimMechanics model regardless of its complexity. The steps are similar to those for building a regular Simulink model. More complex models add steps without changing these basics:

- 1 *Select Ground, Body, and Joint blocks.* From the Bodies and Joints libraries, drag and drop the Body and Joint blocks needed to represent your machine, including at least one Ground block, into a Simulink model window.

Ground blocks represent immobile ground points at rest in absolute (inertial) space.

Body blocks represent rigid bodies.

Joint blocks represent relative motions between the Body blocks to which they are connected.

- 2** *Position and connect blocks.* Place Joint and Body blocks in proper relative position in the model window and connect them in the proper order. The essential result of this step is creation of a *valid tree* block diagram made of

Ground — Joint — Body — Joint — Body — ... — Body

with an open or closed topology and where at least one of the bodies is a Ground block.

A Body can have more than two Joints attached, marking a branching of the sequence. But Joints must be attached to two and only two Bodies.

- 3** *Configure Body blocks.* Click the Body blocks to open their dialog boxes; specify their mass properties (masses and moments of inertia), then position and orient the Bodies and Grounds relative to the World coordinate system (CS) or to other CSs. You set up Body CSs here.
- 4** *Configure Joint blocks.* Click each of the Joint blocks to open its dialog box and set translation and rotation axes and spherical pivot points.
- 5** *Select, connect, and configure Constraint and Driver blocks.* From the Constraints & Drivers library, drag, drop, and connect Constraint and Driver blocks in between pairs of Body blocks. Open and configure each Constraint/Driver's dialog box to restrict or drive the relative motion between the two respective bodies of each constrained/driven pair.
- 6** *Select, connect, and configure Actuator and Sensor blocks.* From the Sensors & Actuators library, drag and drop the Actuator and Sensor blocks that you need to impart and sense motion. Reconfigure Body, Joint, and Constraint/Driver blocks to accept Sensor and Actuator connections. Connect Sensor and Actuator blocks. Specify control signals (applied forces/torques or motions) through Actuators and measure motions through Sensors.

Actuator and Sensor blocks connect SimMechanics blocks to non-SimMechanics Simulink blocks. You cannot connect SimMechanics blocks to regular Simulink blocks otherwise. Actuator blocks take inport signals from normal Simulink blocks (for example, from the Simulink Sources library) to actuate motion. Sensor block output ports generate

Simulink signals that you can feed to normal Simulink blocks (for example, from the Simulink Sinks library).

In the most straightforward model of a machine, you apply forces/torques and initial conditions, then start the simulation in the Forward Dynamics mode to obtain the resulting motions. In the Kinematics and Inverse Dynamics modes, you apply motions to all independent degrees of freedom. With these modes, you can find the forces/torques needed to produce these imposed motions.

- 7 *Encapsulate subsystems.* Systems made from SimMechanics blocks can function as subsystems of larger models, like subsystems in normal Simulink models. You can connect an entire SimMechanics model as a subsystem to a larger model by using the Connection Port block in the Utilities library.

## Essential Steps to Configure and Run a Model

After you've built your model as a connected block diagram, you need to decide how you want to run your machine, configure global settings, and set up visualization.

- SimMechanics offers four analysis modes for running a machine model. The mode you will probably use most often, at least at first, is Forward Dynamics. But a more complete analysis of a machine makes use of the Kinematics, Inverse Dynamics, and Trimming modes as well. You can create multiple versions of the model, each with the same underlying machine, but connected to Sensors and Actuators and configured differently for different modes.
- You can also use the powerful visualization and animation features of SimMechanics. You can visualize your machine as you build it or after you are finished but before you start the simulation, as a tool for debugging the machine geometry. You can also animate the machine model as you simulate.
- Choose the analysis mode and configuring visualization, as well as other important settings, in the **Mechanical Environment Settings** dialog box. You might also need to reconfigure the Simulink **Simulation Parameters** dialog for SimMechanics models. See “A Four Bar Mechanism” on page 2-36 for an example.

The tutorials of this chapter introduce you to most of these steps.

---

**Caution** You might want to make modifications to these tutorial models. To avoid errors, be sure

- Not to attempt connecting Simulink signal lines directly to SimMechanics blocks other than Actuators and Sensors
- To keep the collocation of the Body coordinate system origins on either side of each assembled Joint to within assembly tolerances

You should save multiple versions of models as you try different analysis modes and configurations.

---

The first tutorial in the next section shows you how to configure the most basic blocks in any model: Ground, Body, and a Joint, in order to create a simple pendulum model. The second tutorial explains how to visualize and animate the pendulum.

## Building a Simple Pendulum

In this first tutorial, you drag, drop, and configure the most basic blocks needed for any mechanical model, as well as add some sensors to measure motion. The tutorial guides you through these aspects of model-building:

- “The World Coordinate System and Gravity” on page 2-12
- “Configuring a Ground Block” on page 2-12
- “Configuring a Body Block” on page 2-14
- “Configuring a Joint Block” on page 2-20
- “Adding Sensors and Starting the Simulation” on page 2-24

The end result is a model of a simple pendulum. The pendulum is a swinging steel rod. We strongly recommend that users work through this tutorial first before moving on to “Visualizing a Simple Pendulum” on page 2-30.



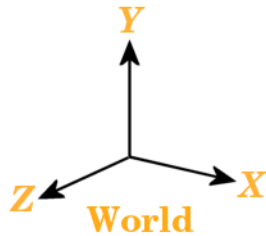
### A Simple Pendulum: A Swinging Steel Rod

#### Opening the SimMechanics Block Library

Following one of the ways described earlier in the “Viewing the Blocks” section in this chapter, open the SimMechanics library. Then from the SimMechanics library, open a new, empty Simulink model window.

### The World Coordinate System and Gravity

Before you configure a Ground block, you need to understand SimMechanics' internally defined fixed or “absolute” coordinate system (CS) called *World*. The World CS sits at rest in the inertial reference frame also called World. The World CS has an origin (0,0,0) and a triad of right-handed, orthogonal coordinate axes.



The default World coordinate axes are defined so that

- +x points right
- +y points up (gravity in  $-y$  direction)
- +z points out of the screen, in three dimensions

The vertical direction or up-and-down is determined by the gravity vector direction (acceleration  $\mathbf{g}$ ) relative to the World axes. Gravity is a background property of a model that you can reset before starting a simulation, but does not dynamically change during a simulation.

See “Mechanical Environment Settings Dialog Box” on page 5-7 for displaying global mechanical properties of SimMechanics models.

### Configuring a Ground Block

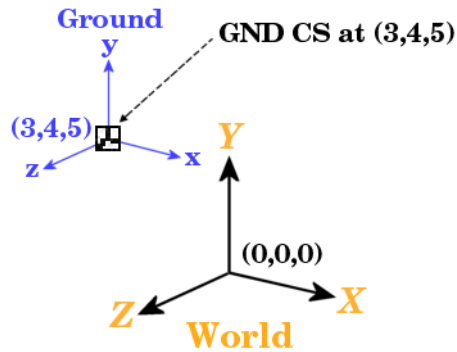
World serves as the single absolute CS that defines all other CSs. But you can create additional *ground points* at rest in World, at positions other than the World origin, by using Ground blocks. Ground blocks, representing ground points, play a dynamical role in machine models. They function as immobile bodies.



---

**Minimum Ground Block** Every machine model must have *at least one* Ground block.

---

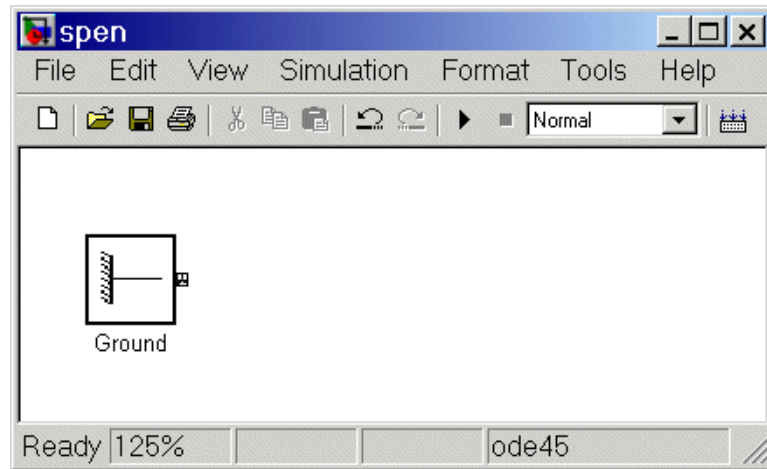


**Figure 2-1: A Ground Point Relative to World**

### Steps to Configuring the Ground Block

Now place a fixed ground point at position (3,4,5) in the World CS:

- 1 In the SimMechanics library, open the Bodies library.
- 2 Drag and drop a Ground block from the Bodies library into the model window. Close the Bodies library.
- 3 Open the Ground block dialog box. Into the **Grounded point/Location** [x y z] field, enter the vector [3 4 5]. Click **OK** to close the dialog box.



### Properties of Grounds

At every ground point, a *Grounded CS* is automatically created:

- The origin of each Grounded CS is the ground point itself.
- The Grounded CS axes are always fixed to be parallel to the World CS axes, as shown in Figure 2-1.
- You can use these Grounded CSs in setting up later Body CSs.

### Configuring a Body Block

While you need at least one Ground block to make a machine model, a real machine consists of one or more rigid bodies. So you need to translate the components of a real machine into block representations. This section explains how you use a Body block to represent each rigid body in your machine:

- “Characteristics of a Body Block” on page 2-15
- “Properties of the Simple Pendulum Body” on page 2-15
- “Configuring the Body Dialog” on page 2-17

Although the body is the most complicated component of a machine, SimMechanics does not use the full geometric shape and mass distribution of the body. SimMechanics only needs certain mass properties and simplified

geometric information about the body's center of gravity, its orientation, and the coordinate systems attached to the body.

Setting these properties sets the body's initial conditions of motion, if you do nothing else to the Body block or its connected Joints before simulating.

### Characteristics of a Body Block

The main characteristics of a Body block are its *mass properties*, its *position* and *orientation* in space, and its attached *Body coordinate systems* (CSs).

The mass properties include the *mass* and *inertia tensor*. The mass is a real, positive scalar. The inertia tensor is a real, symmetric 3-by-3 matrix. It does not have to be diagonal.

The position of the body's *center of gravity* (CG) and *orientation* relative to some coordinate system axes indicate where the body is and how it is rotated. These are the body's initial conditions during construction of the model and remain so when you start the simulation, unless you change them before starting.

The attached *Body CSs* (their origins and coordinate axes) are fixed rigidly in the body and move with it. The minimum CS set is one, the CS at the CG (the CG CS), with its CS origin at the center of gravity of the body. The default CS set is three, the CG CS and two additional CSs called CS1 and CS2 for connecting to Joints on either side. See the next section, "Configuring a Joint Block" on page 2-20.

Beyond the minimum CS at the CG, you can attach as many Body CSs on one Body as you want. You need a separate CS for each connected Joint, Constraint, or Driver and for each attached Actuator and Sensor.

The inertia tensor components are interpreted in the CG CS, setting the orientation of the body relative to the CG CS axes. The orientation of the CG CS axes relative to the World axes then determines the absolute initial orientation of the body. Since the CG CS axes remain rigidly fixed in the body during the simulation, this relative orientation of the CG CS axes and the body does not change during motion. The inertia tensor components in the CG CS also do not change. As the body rotates in inertial space, however, the CG CS axes rotate with it, measured with respect to the absolute World axes.

### Properties of the Simple Pendulum Body

The simple pendulum is a uniform, cylindrical steel rod of length 1 meter and diameter 2 cm. The initial condition is the rod lying parallel to the negative

$x$ -axis, horizontal in the gravity field. One end of the rod, the fixed pivot for the rod to swing, is located at the ground point (3,4,5). Its coordinate system is called CS1. The center of gravity and the origin of the CG CS is the geometric center of the rod. Take the CG CS axes to be parallel to the World axes as you set up the pendulum.

Uniform steel has density  $\rho = 7.93 \text{ gm/cc}$  (grams per cubic centimeter). In the CG CS here, the inertia tensor  $\mathbf{I}$  is diagonal, and  $I_{zz}$  controls the swinging about the  $z$ -axis, in the  $x$ - $y$  plane. The inertia tensor is always evaluated with the origin of coordinates at the CG. For a rod of length  $L = 1 \text{ m}$  and radius  $r = 1 \text{ cm}$ , the mass  $m = \rho\pi r^2 L = 2490 \text{ gm}$  (grams), and the inertia tensor  $\mathbf{I}$  reads

$$\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} \frac{mr^2}{2} & 0 & 0 \\ 0 & \frac{mL^2}{12} & 0 \\ 0 & 0 & \frac{mL^2}{12} \end{bmatrix} = \begin{bmatrix} 1250 & 0 & 0 \\ 0 & 2.08 \times 10^6 & 0 \\ 0 & 0 & 2.08 \times 10^6 \end{bmatrix}$$

in  $\text{gm-cm}^2$  (gram-centimeters<sup>2</sup>). The  $x$ -axis is the cylinder's symmetry axis. Thus  $I_{yy} = I_{zz}$ .

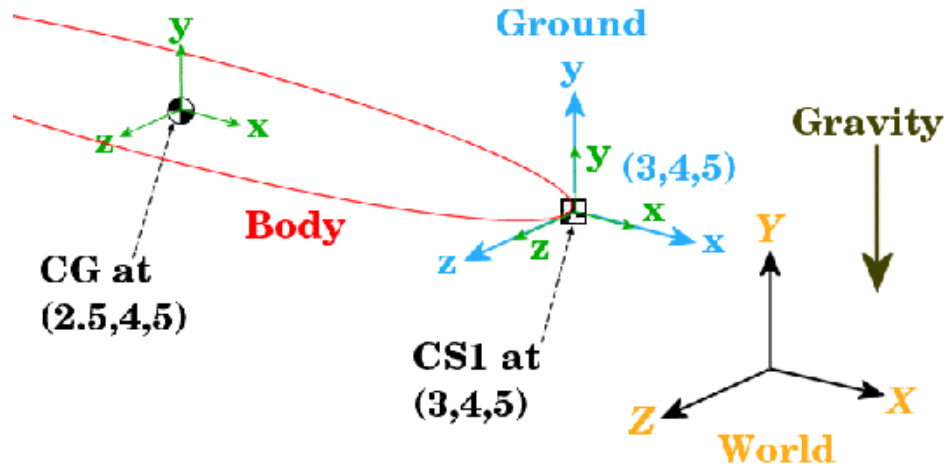
The mass and geometric properties of the body are summarized in the following table and depicted in Figure 2-2 on page 2-17.

**Body Data for the Simple Pendulum**

Property	Value
Mass (gm)	2490
Inertia tensor ( $\text{kg-m}^2$ )	[ 1.25e-4 0 0; 0 0.208 0; 0 0 0.208]
CG Position/Origin (m)	[ 2.5 4 5 ]
CS1 Origin (m)	[ 3 4 5 ]

## Configuring the Body Dialog

Take the steps to configuring a Body block dialog box in several stages.



**Figure 2-2: Equivalent Ellipsoid of Simple Pendulum with Coordinate Systems**

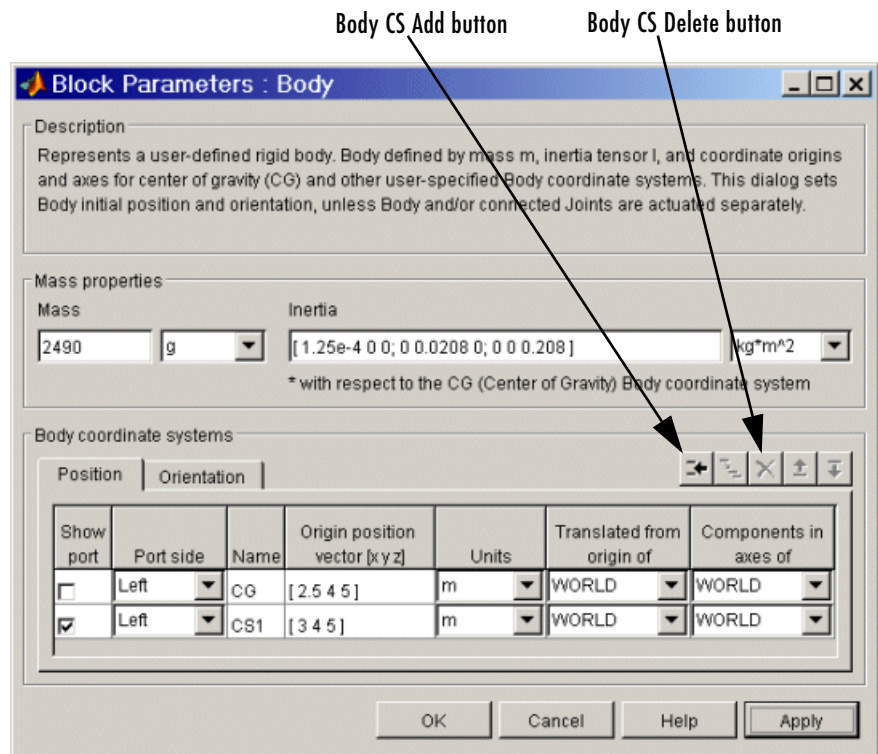
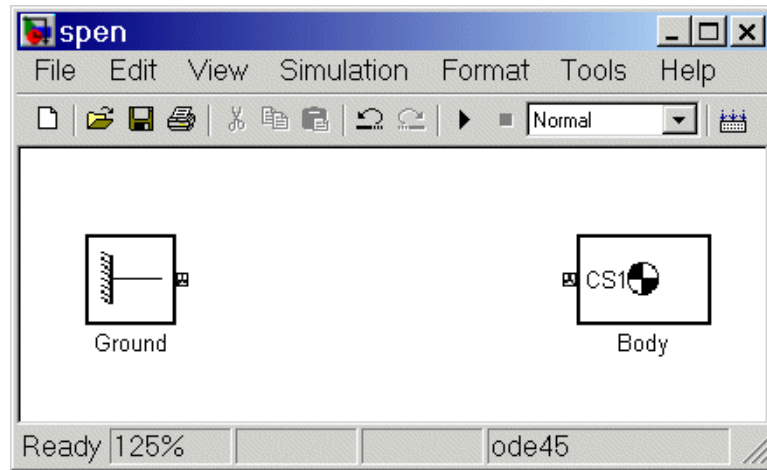
**Adding the Body Block.** To start working with the Body block:

- 1 Open the Bodies library in the SimMechanics library.
- 2 Drag and drop a Body block into your model window.
- 3 Open the Body block dialog box. Note the two main areas you need to configure:
  - **Mass properties** — These are the mass and inertia tensor.
  - **Body coordinate systems** — These are the details about the position and orientation of the Body CSs.

---

**Note** To apply your dialog entries at any time, click **Apply**. To close a dialog, click **OK**.

---



**Configuring the Body's Mass Properties.** Now enter the body's mass and inertia tensor:

- 1 Use the data from the table "Body Data for the Simple Pendulum" on page 2-16.

In the **Mass** field, enter 2490 and change the units to g (grams).

- 2 In the **Inertia tensor** field, enter [1.25e-4 0 0; 0 0.208 0; 0 0 0.208] and leave the default units as kg-m<sup>2</sup>.

**Configuring Body Coordinate Systems (Position).** Configure the translational position of the body and its Body CS origins in space:

- 1 Use the data from table "Body Data for the Simple Pendulum" on page 2-16, and work on the **Position** pane. Vectors are assumed translated from the World origin and oriented to the World axes.
- 2 Note the three default CSs in the Body dialog box. The CS at the CG is necessary for any Body, and you will connect CS1 to the Ground with a Joint shortly.

Delete CS2 by selecting its line in the Body CS list and clicking the **Delete** button in the Body CS controls.

You have two already existing CSs not on this Body that you can use to specify the positions of the Body CS origins that are on this Body:

- Preexisting World origin at [0 0 0]
- The Adjoining CS on the neighboring body, in this case the Grounded CS origin at [3 4 5]

- 3 Specify the CG and CS1 origins relative to World:
  - a In the pull-down menu under **Translated from origin of**, choose World for both CSs: CG and CS1.
  - b Under **Origin position vector**, specify the position of the origin of each CS, translated from the World origin:

[3 4 5] for CS1

[2.5 4 5] for CG

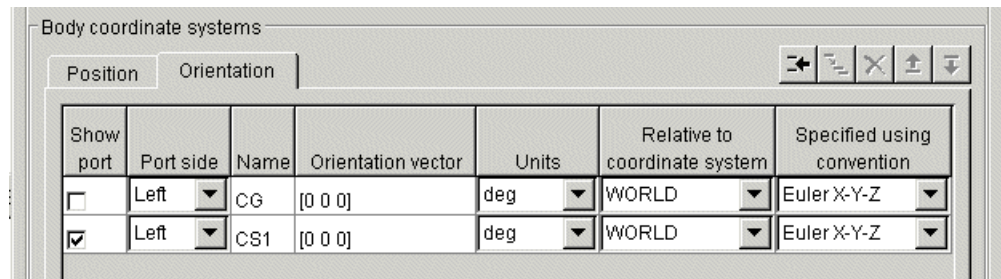
- 4 Select a CS relative to whose coordinate axes the components of the vectors in the last step are measured. You choose these CS axes in the **Components in axes of** menu. Select World for both CSs. Leave the units as m (meters).

**Configuring Body Coordinate Systems (Orientation).** Configure the rotational orientation of the body and its Body CS axes in space:

- 1 Work on the **Orientation** pane. The default orientation for all CS axes is parallel to World. The sign of all rotations is determined by the *right-hand rule*.

In Figure 2-2, the CS1 and CG axes are oriented parallel to the World axes, so the CS1 and CG axes need no rotation.

- 2 For both CSs, set the **Relative to coordinate system** menu to World.
- 3 For CG and CS1, leave the **Orientation vector** at default [0 0 0] and the **Specified using convention** at default Euler X-Y-Z. Close the Body dialog.



## Configuring a Joint Block

A machine is made up of Bodies with geometric and mass information. But Bodies carry no information of how they move. The possible directions of motion that a Body can take are called its *degrees of freedom* (DoFs), and this section explains how you represent these DoFs by Joint blocks:

- “How to Connect a Joint Between Two Bodies” on page 2-21
- “Revolute Joint for the Simple Pendulum” on page 2-21



---

**DoFs Are Relative** In SimMechanics, DoFs and the Joints that represent them are *relative* DoFs. That is, DoFs represent the possible motions between one body and another. So a DoF is defined by a pair of bodies, and you must connect every Joint to two and only two Bodies.

---

One (but not both) of the members of such a pair of Bodies can be a Ground. The other member Body of such a pair then has its motion defined relative to a fixed ground point. This fixed ground point does not have to be the same as the World origin. A machine can have many such Ground-Body pairs and *must have at least one*.

### **How to Connect a Joint Between Two Bodies**

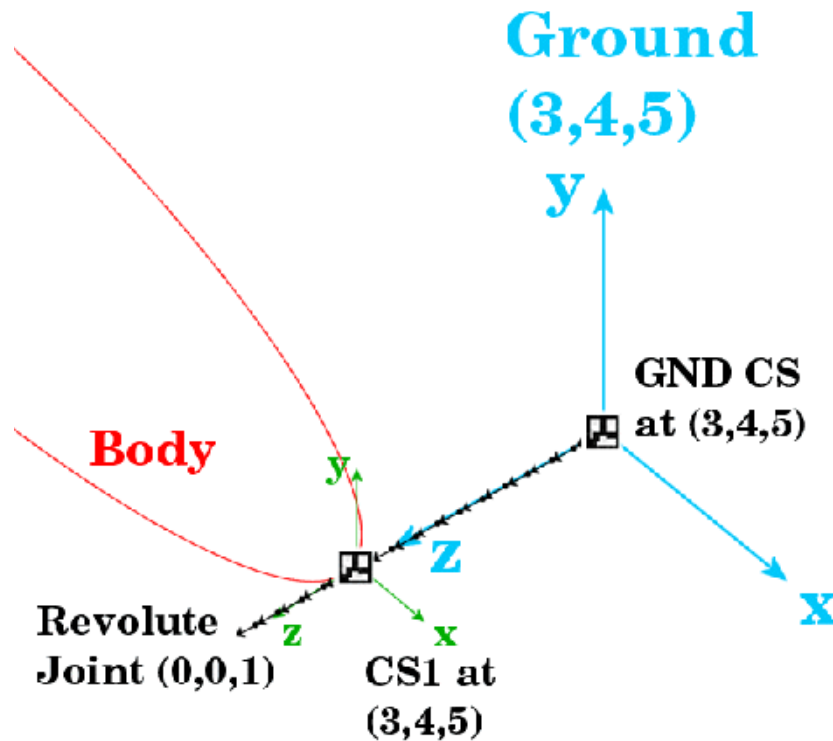
You represent relative motion of bodies with respect to one another by connecting their Body blocks with Joints. You can connect a Body to one or more Joints.

A Joint block is always connected to a specific point on the Body on either side of the Joint. The specific point for anchoring a Joint on a Body is the origin of a Body CS, and a Joint is therefore connected on one side to one Body at a Body CS origin, and on the other side to the other Body at a Body CS origin.

Usually a Body is connected to a Joint on either side, so the default you saw earlier in this tutorial for Body CSs in the Body dialog box is three Body CSs: the CS at the center of gravity (CG) and two other CSs (CS1 and CS2). But a Body at the end of a Body—Joint—...—Body chain is connected to only one Joint.

### **Revolute Joint for the Simple Pendulum**

In spite of the complexity of the concepts implicit in a Joint, the actual configuration of a Joint block is fairly simple. Here you insert and configure one revolute Joint block between the Ground and Body blocks you've already put into the model window.



**Figure 2-3: Simple Pendulum Connected to Ground via Revolute**

**Configuring the Revolute Joint Block.** The geometry of the Joint connection is shown in Figure 2-3. The ground point at (3,4,5) and the CS1 at (3,4,5) are actually the same point in space, but have been separated in the figure for clarity. The revolute rotation axis is along the +z direction:

- 1 Open the Joints library in the block library.
- 2 Drag and drop a Revolute block into your model window.
- 3 Rotate the Revolute block so that you can connect the base (B) side of the Joint to the Ground block and the follower (F) side of the Joint to the Body block. Make the two connections.

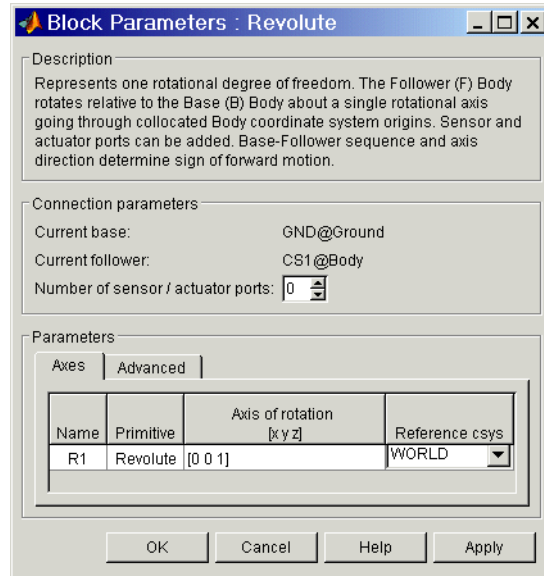
**4** Open the **Revolute** dialog box. In the **Parameters** area, on the Axes pane, configure the rotation axis to the World  $z$ -axis:

- a** Enter [0 0 1] under **Axis of rotation [x y z]**
- b** Leave the **Reference csys** at WORLD
- c** Ignore the **Advanced** tab

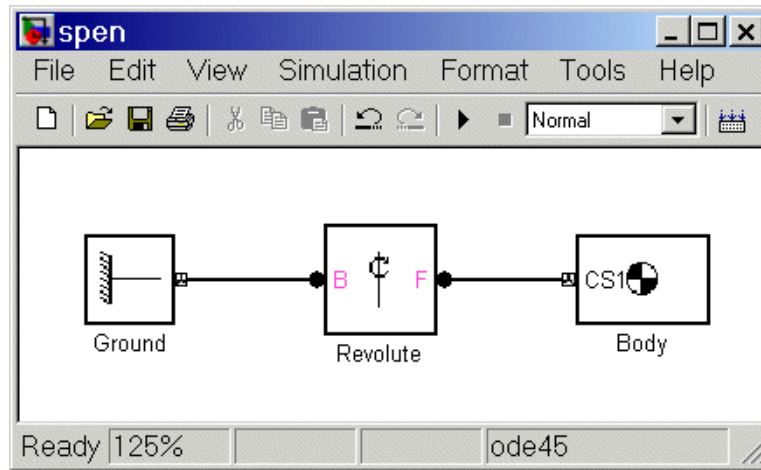
Note several important things:

- Under **Connection parameters**, the **Current base** is located at GND@Ground, which is the Grounded CS associated with the Ground block located at (3,4,5) in World.
- Under **Connection parameters**, the **Current follower** is located at CS1@Body, which is the CS1 on Body1 located at (3,4,5) in World.
- This Joint's directionality runs from Ground to Body along the  $+z$  axis.

**5** Close the **Revolute** dialog box.



Congratulations — you have now finished the simplest possible model of a machine: a connected block diagram of Ground—Joint—Body. Your model window should look like this.



### Adding Sensors and Starting the Simulation

To measure the motion of the pendulum as it swings, you need to hook one or more Simulink Scope blocks to your model. The special library of Actuators and Sensor blocks gives you the means to input and output Simulink signals to and from SimMechanics models. Sensors allow you to watch the mechanical motion once you start the simulation, as the following explain:

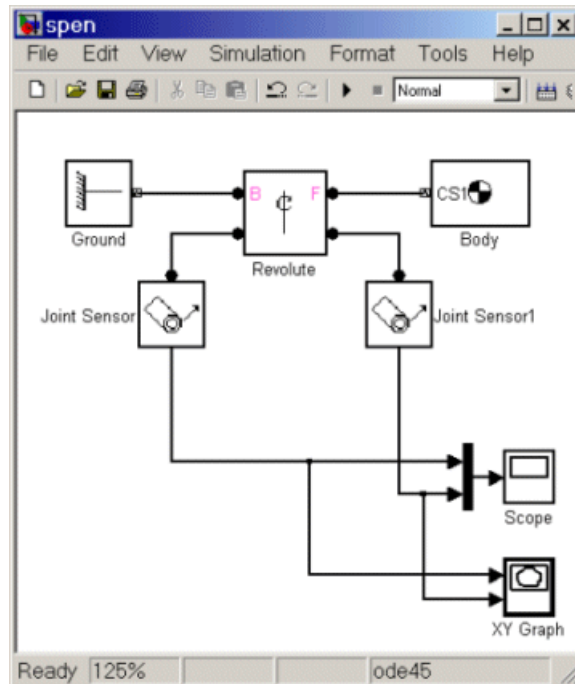
- “Connecting and Configuring Pendulum Sensors”
- “Configuring Simulation Parameters and Mechanical Environment Settings” on page 2-26
- “Starting and Interpreting the Motion” on page 2-27

#### Connecting and Configuring Pendulum Sensors

In this example, you measure the angular motion of the revolute joint:

- 1 In the block library, open the Sensors and Actuators library. Drag and drop two Joint Sensor blocks into your model window.

- 2 Open the Revolute block. Change **Number of sensor/actuator ports** from 0 to 2 using the spinner menu. Two open connector ports  $\circ$  appear on either side of Revolute. Close Revolute.
- 3 Connect these connector ports to the connector ports on the Joint Sensor blocks. The open connector ports change to solid  $\bullet$ .
- 4 Open the Simulink Library Browser. From the Sinks library, drag and drop a Scope block and an XY Graph block into your model window. From the Signal Routing library, drag and drop a Mux block as well. Connect the Simulink outputs  $>$  on the Joint Sensor blocks to the Scope and XY Graph blocks as shown.



The lines from the outputs  $>$  to the Scope and XY Graph blocks are normal Simulink signal lines and can be branched. You cannot branch the lines connecting SimMechanics blocks to each other at the round connector ports  $\bullet$ .

- 5 Open Joint Sensor. Select only the **Angle** check box. Open Joint Sensor1. Select only the **Angular velocity** check box. Leave the other defaults. Close both Sensor blocks.
- 6 Save your model for future reference as `spen.mdl`.

You now need to configure the global parameters of your model before you can run it.

### Configuring Simulation Parameters and Mechanical Environment Settings

The **Simulation Parameters** dialog box is a standard feature of Simulink. Reset its entries for this model to obtain more accurate simulation results.

- 1 In the Simulink menu bar, open the **Simulation** menu and click **Simulation parameters** to open the **Simulation Parameters** dialog.
- 2 Change **Relative tolerance** to  $1e-6$  and **Absolute tolerance** to  $1e-4$ .

If you want the simulation to stop after a finite time, change **Stop time** to a finite number of seconds. The pendulum period is approximately 1.6 sec.

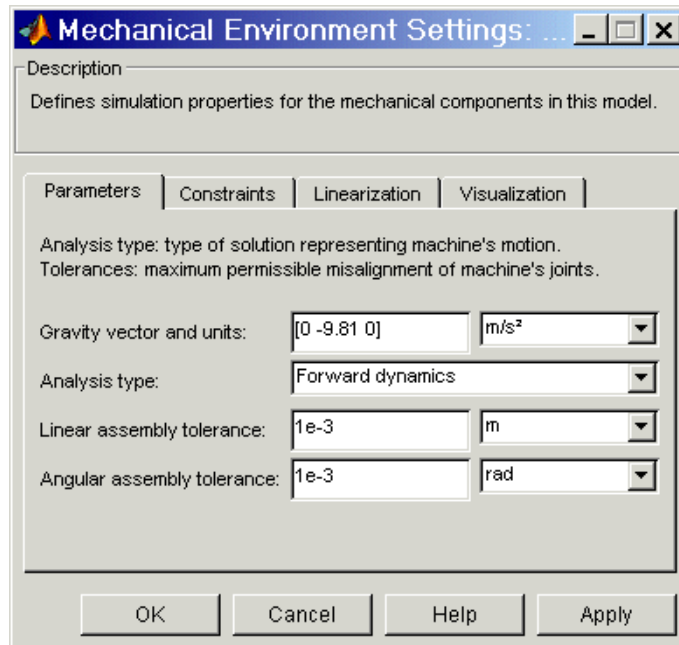
- 3 Close the **Simulation Parameters** dialog box.

A special feature of SimMechanics models is the **Mechanical Environment Settings** dialog box.

- 1 In the Simulink menu bar, open the **Simulation** menu and click **Mechanical environment** to open the **Mechanical Environment Settings** dialog.

Note the default **Gravity vector and units**:  $[0 \ -9.81 \ 0] \text{ m/s}^2$ , which points in the  $-y$  direction, as shown in Figure 2-2 on page 2-17. Gravitational acceleration  $g = 9.81 \text{ m/s}^2$ .

- 2 Close the **Mechanical Environment Settings** dialog.



### Starting and Interpreting the Motion

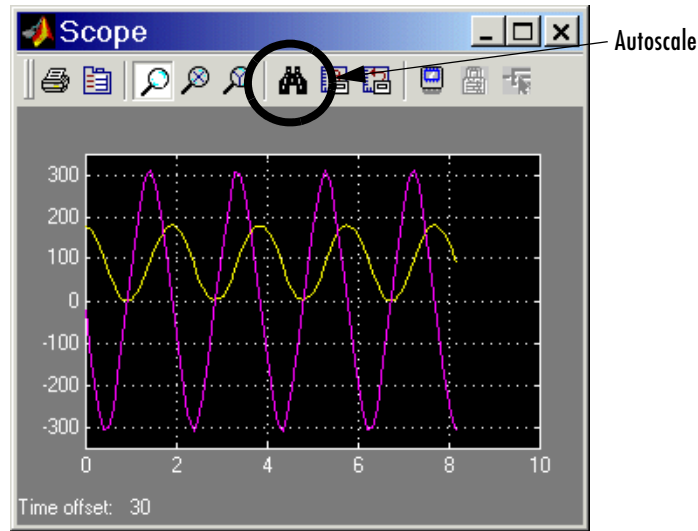
You can now start your simulation and watch the pendulum motion via the Scope and XY Graph blocks:

- 1 Open the XY Graph block dialog box. Set the following parameters.

Parameter	Value
x-min	0
x-max	200
y-min	-500
y-max	500

Leave **Sample time** at default and close the dialog.

- 2 Open the Scope block and start the model. The XY Graph opens automatically when you start the simulation.
- 3 View the full motion of both angle and angular velocity (in degrees and degrees per second, respectively) as functions of time in Scope. Click **Autoscale** if the motion is not fully visible.



### Angle and Angular Velocity of the Simple Pendulum as Functions of Time

The motion is periodic but not simple harmonic (sinusoidal), because the amplitude of the swing is so large (180 degrees from one turning point to the other). Note that the zero of angle is the initial horizontal angle, not the vertical. The zeros of motion are always the initial conditions.

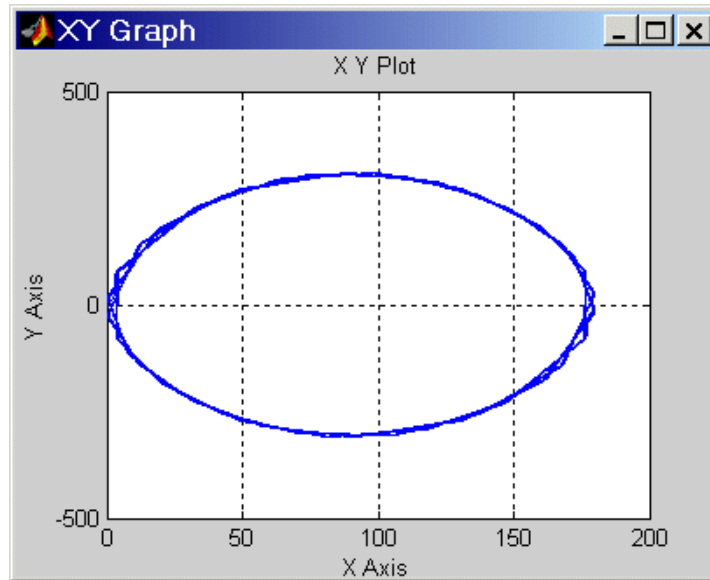
The XY Graph shows the angle versus angular velocity, with no explicit time axis. These two variables trace out a figure similar to an ellipse, because of the conservation of total energy  $E$ :

$$\frac{1}{2}J\left(\frac{d\theta}{dt}\right)^2 + mgh(1 - \sin\theta) = E = \text{const}$$

where  $J = I_{zz} + mL^2/4$  is the inertial moment of the rod about its pivot point (not the center of gravity). The two terms on the left side of this equation are



the kinetic and potential energies, respectively. The coordinate-velocity space is the *phase space* of the system.



### Phase Space Plot of Simple Pendulum Motion: Angular Velocity Versus Angle

The directionality of the Revolute Joint assumes that the rotation axis lies in the  $+z$  direction. Looking at the pendulum from the front, follow Figure 2-1 on page 2-13, Figure 2-2 on page 2-17, and Figure 2-3 on page 2-22. Positive angular motion from this perspective is counterclockwise, following the right-hand rule.

The next tutorial walks you through visualizing and animating this same simple pendulum model.

# Visualizing a Simple Pendulum

In this section, you learn how to view the swinging pendulum rod of the model introduced in the last section using the SimMechanics visualization tools. Use your saved `spen.mdl` model, or use the `mech_spn` model in the Demos library.

SimMechanics has two independent tools for visualizing a machine. One is an internal tool based on MATLAB Handle Graphics. The other is an optional virtual reality scene and requires the Virtual Reality Toolbox 3.0 to be installed. Both tools display a machine by representing its bodies. The bodies can be displayed in two ways, by equivalent ellipsoids and by closed surfaces (convex hulls) enveloping the bodies' coordinate systems.

---

**Note** You can find more on visualizing and animating machine models in “Visualizing and Animating Machines” on page 6-1

---

This section explains how to visualize your pendulum with either tool, using either body representation. You can view the pendulum before you start and, separately, choose to animate it during simulation as well:

- “Representing the Bodies” on page 2-31
- “Visualizing with Handle Graphics” on page 2-32
- “Visualizing with Virtual Reality” on page 2-34

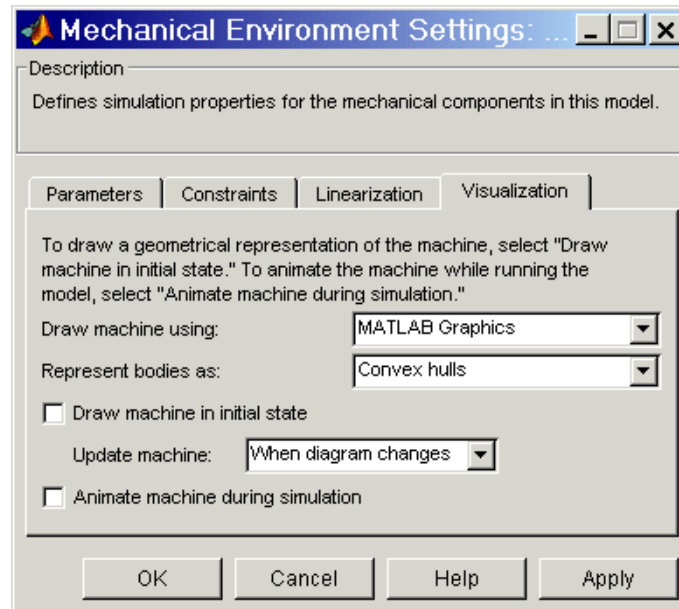
## Starting Visualization

The first step is to open the **Mechanical Environment Settings** dialog box from your model window:

- 1 On the Simulink menu bar, open the **Simulation** menu and open the **Mechanical environment** entry. The **Mechanical Environment Settings** dialog box appears. In the lower half, click the **Visualization** tab at the right.
- 2 To view the pendulum in its static initial condition, select the **Draw machine in initial state** check box.

To animate the pendulum visualization while the simulation is running, select the **Animate machine during simulation** check box as well.

- 3 Leave the dialog open until the end of this section, as you will need to configure additional settings.



## Representing the Bodies

The information that you use to specify body properties in a SimMechanics model is enough to represent each body in certain special, simplified ways. SimMechanics does not have the information about the bodies needed to represent their full geometries.

### Equivalent Ellipsoids

A rigid body has a unique equivalent ellipsoid, a homogeneous solid ellipsoid with the same inertia tensor. For more about this representation figure for rigid bodies, see “Rendering Body Shapes in SimMechanics” on page 6-5.

Because the rod has an axis of symmetry, the  $x$ -axis in this case, two of its three generalized radii are equal:  $a_y = a_z$ . The generalized radii of the equivalent ellipsoid are  $a_x = \sqrt{5/3}(L/2) = 0.646$  m and  $a_y = a_z = \sqrt{5}(r/2) = 1.12$  cm.

### Convex Hulls

Every Body has at least one Body coordinate system (CS) at the CG. A Body also has one or more extra Body CSs for the attached Joints, as well as possible Actuators and Sensors. Each Body CS has an origin point, and the collection of all these points, in general, defines a volume in space. The minimum outward-bending surface enclosing such a volume is the convex hull of the Body CSs, and this is the alternative way that SimMechanics has to represent a body.

You created the pendulum body with only two Body CSs, CG and CS1. The convex hull for the pendulum rod is thus the minimum connecting figure, a line joining the two Body CS origins.

### Choosing the Body Representation

You can choose which displayed representation of the pendulum rod or any machine bodies SimMechanics uses, by configuring the **Visualization** tab in the **Mechanical Environment Settings** dialog box. In the **Represent bodies as** menu, choose Convex hulls or Equivalent ellipsoids.

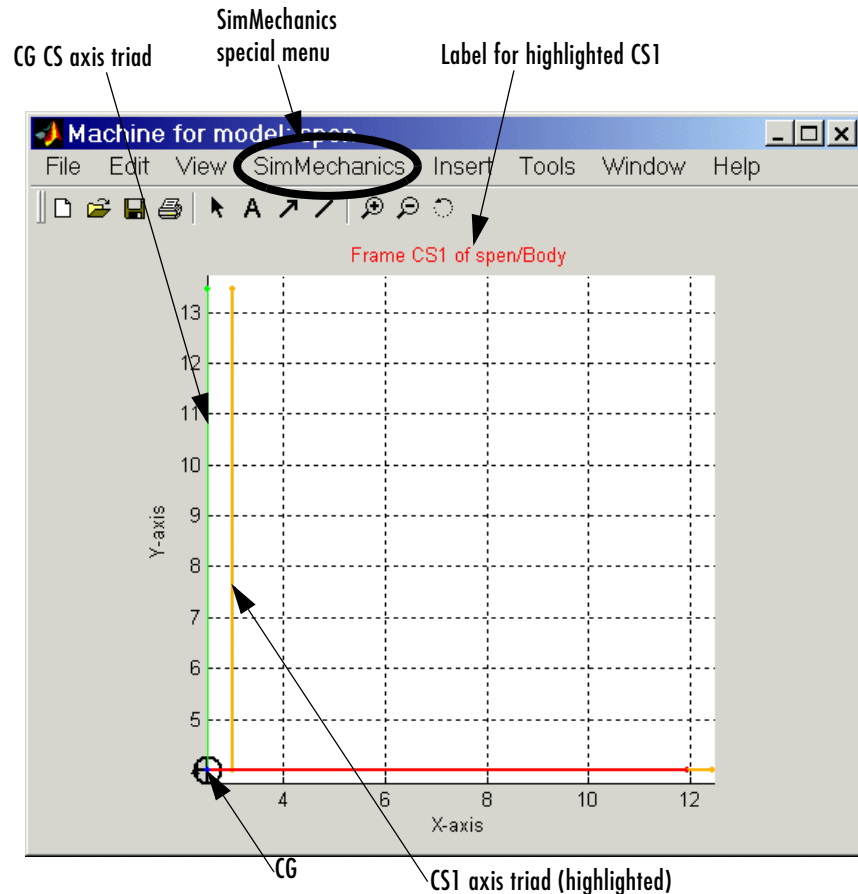
Continue to keep the **Mechanical Environment Settings** dialog box open.

### Visualizing with Handle Graphics

The Handle Graphics-based visualization tool is built into SimMechanics. To choose this visualization mode:

- 1 Use the **Draw machine using** menu to pick MATLAB Graphics.
- 2 Now click **Apply** or **OK** to apply all your visualization choices.
- 3 A MATLAB Handle Graphics window appears.

The displayed figure depends on the body representation you choose. The convex hull appears.



### Handle Graphics Window for Pendulum Rod (Convex Hull)

You can change the viewpoint and manipulate the image using standard Handle Graphics techniques. Consult “Viewing Machines with Handle Graphics” on page 6-10 as well. Experiment with the special **SimMechanics** menu’s settings to see different ways of displaying the pendulum.

When you start the model, the body in the graphics window moves in step with the simulation.

### Visualizing with Virtual Reality

---

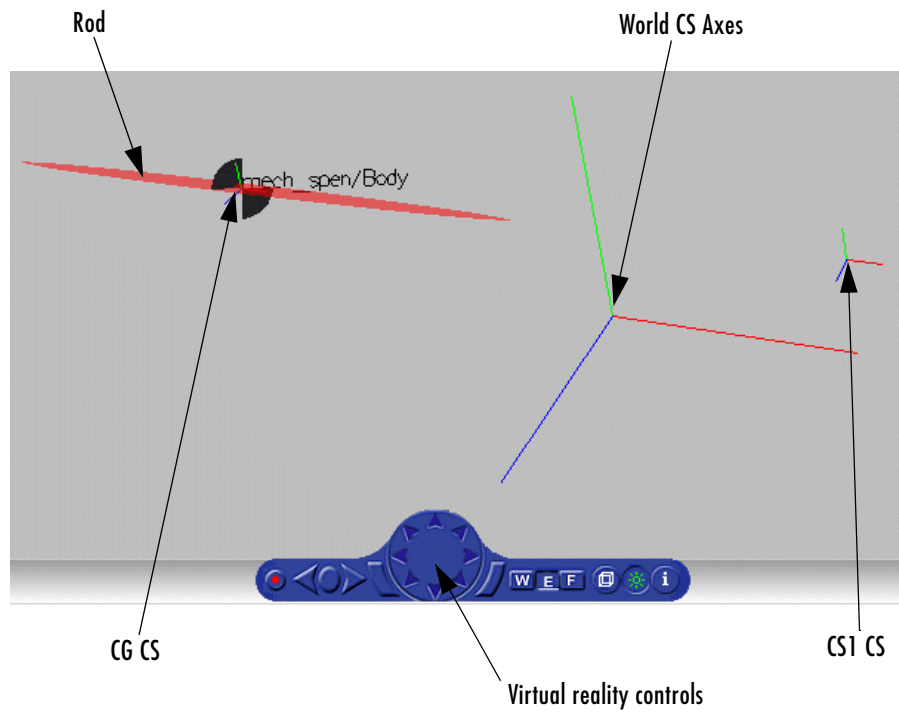
**Note** This optional feature requires the Virtual Reality Toolbox to be installed on your MATLAB path.

---

To choose the Virtual Reality-based visualization tool, follow the previous steps for Handle Graphics, but change the **Draw machine using** choice:

- 1** Open the **Mechanical Environment Settings** dialog box and click the **Visualization** tab.
- 2** Use the **Draw machine using** menu to select Virtual Reality Toolbox.
- 3** Now click **Apply** or **OK** to reapply your visualization choices.
- 4** A virtual reality viewer, displaying a virtual scene, appears.

The displayed figure depends on the body representation you choose. The equivalent ellipsoid looks like this.



The next tutorial shows how to create, run, and visualize a model for a more complex machine, a four bar mechanism. To configure Ground, Body, and Joint blocks now means repeating and expanding upon the three blocks of the first two tutorials.

# A Four Bar Mechanism

In this tutorial, you build a model of a planar four bar mechanism and practice using some of the important SimMechanics features:

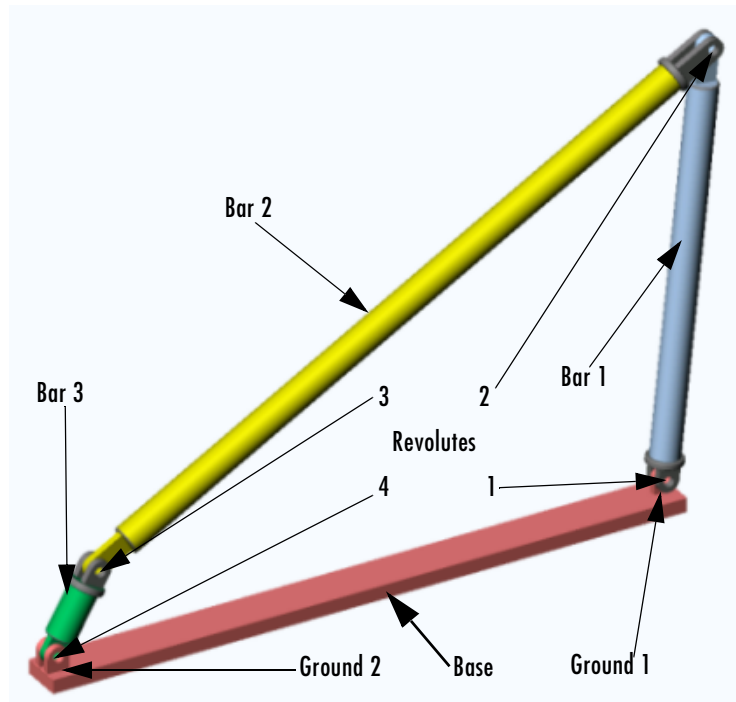
- “Configuring the Mechanical Environment Settings” on page 2-38
- “Setting Up the Block Diagram” on page 2-40
- “Configuring the Ground and Joint Blocks” on page 2-43
- “Configuring the Body Blocks” on page 2-47
- “Sensing Motion and Running the Model” on page 2-52

You are urged to go through “Building a Simple Pendulum” on page 2-11 and “Visualizing a Simple Pendulum” on page 2-30 before proceeding with this section.

The machine consists of three moving bars of homogeneous steel, two connected at one end each to ground points and a third cross bar connecting the first two. The base acts as an immobile fourth bar, with a Ground at each end. The machine forms a single closed loop, and its motion is confined to two dimensions.

The elementary parts of the machine are the bodies, while the revolute joints are the idealized rotational degrees of freedom (DoFs) at each body-to-body contact point. The bodies and the joints expressing the bodies’ relative motions must be translated into corresponding SimMechanics blocks. If you want, you can add elaborations such as Constraints, Drivers, Sensors, and Actuators to this essential block diagram.





**Figure 2-4: Four Bar Mechanism**

### Counting the Degrees of Freedom

The three moving bars are constrained to move in a plane. So each bar has two translational and one rotational DoFs, and the total number of machine DoFs, before counting constraints, is  $3 \cdot (2+1) = 9$ .

Because the motion of the bars is constrained, however, not all of these nine DoFs are *independent*:

- In two dimensions, each connection of a body with another body or with a ground point imposes two restrictions (one for each coordinate direction). Such a restriction effectively eliminates one of the two body ends as independently moving points, because its motion is determined by the next body's end.

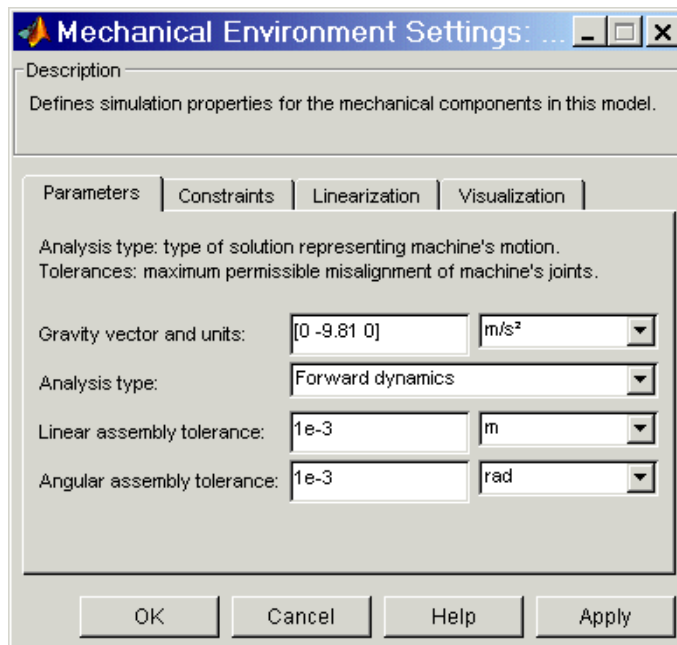
- There are four such body-body or body-ground connections and therefore eight restrictions implicit in the machine's geometry.

The eight restrictions on the nine apparent DoFs reduce the DoFs to one,  $9 - 8 = 1$ . There are four rotational DoFs between bars or between bars and grounds. But three of these are dependent. Specifying the state of one rotational DoF fully specifies the other three.

### Configuring the Mechanical Environment Settings

Open a new blank model window from the SimMechanics library. First you need to set the global mechanical environment.

On the model window toolbar, open the **Simulation** menu and click the **Mechanical environment** entry. The **Mechanical Environment Settings** dialog box appears.



## The Mechanical Environment Settings Dialog Box Panes

Click the four tabs in succession to expose each pane.

Pane	Function
<b>Parameters</b>	Controls general settings for mechanical simulations
<b>Constraints</b>	Sets constraint tolerances and how constraints are interpreted
<b>Linearization</b>	Controls how SimMechanics models are linearized with Simulink
<b>Visualization</b>	Configures the visualization tools

Note some important features of this dialog box:

- The **Gravity vector** field specifies the magnitude and direction of gravitational acceleration and sets the vertical or up-down direction.
- The **Linear** and **Angular assembly tolerance** fields are also set here. Change **Angular assembly tolerance** to 1e-3 deg (degrees). (See “Setting Assembly Tolerances” on page 5-5 and “Parameters Pane” on page 5-7.)
- Leave the other defaults, except **Visualization**.

### Setting the Visualization Pane

**Visualization** We recommend opening one of the visualization tools before building a model. With it, you can keep track of your model components and how they are connected, as well as correct mistakes.

To visualize the bodies as you build the machine, go to the Visualization pane of **Mechanical Environment Settings**:

- 1** In the **Draw machine using** menu, select either Virtual Reality Toolbox or MATLAB Graphics (Handle Graphics).

Leave **Represent bodies as** in default the Equivalent ellipsoids.

- 2 Select the **Draw machine in initial state** check box. Leave the **Update machine** menu as default. When diagram changes to see bodies appear and change in the visualized scene as you add and change them in the model.

If you want to animate the simulation later when you run the model, select the **Animate machine during simulation** check box as well.

- 3 Click **OK** or **Apply**.

If you chose Handle Graphics, an empty Handle Graphics window opens. If you chose virtual reality, an empty virtual world appears in a virtual reality viewer.

Either visualization window becomes populated with bodies as you add them to the model.

If you open visualization after the model is built, the finished machine appears.

## Setting Up the Block Diagram

In this set of steps, you create Bodies, position them, connect them with Joints, then configure the Body and Joint properties. The Body dialog boxes give you many ways to represent the same machine in the same physical state. This section explains one way.

Alternative, equivalent ways of configuring Bodies are discussed in “Body Coordinate Systems” on page 4-10.

### MAT-File Data Entry

The geometric and mass properties you need to specify for the Grounds and Bodies in this model are listed in the tables of the following two sections, “Configuring the Ground and Joint Blocks” on page 2-43 and “Configuring the Body Blocks” on page 2-47.

Instead of typing the numerical values of these properties into the dialog boxes, you can load the variable set you need into the workspace by entering

```
load fourbar_data
```

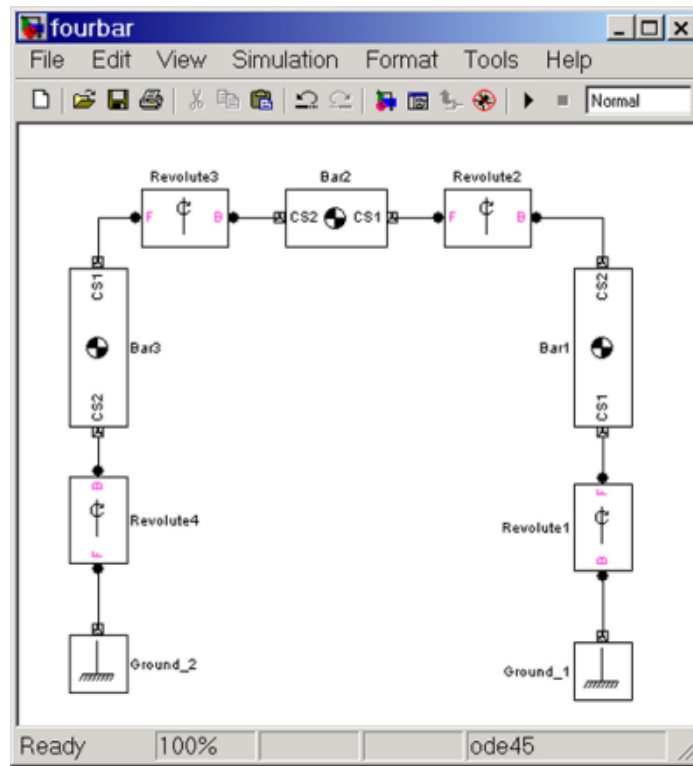
at the MATLAB command line. The variable name for each property is given in the tables. Just enter the appropriate variable names in the appropriate fields as you come to them in the dialog boxes.

### **Block Diagram Setup**

Begin assembling the model with these steps:


- 1** In the block library, open the Bodies library. Drag and drop two Ground blocks and three Body blocks into the new model window. Close the Bodies library.
- 2** From the Joints library, drag and drop four Revolute blocks into the model window.
- 3** Rotate and connect the blocks in the pattern shown in the following figure or with an equivalent block diagram topology.

Use the block names shown in this figure for later consistency.



**Figure 2-5: Connected Ground, Body, and Joint Blocks for Four Bar Model**

**Block diagram topology.** The topology of the block diagram is the connectivity of its elements. The elements are the Bodies and Grounds, connected by the Joints. Unlike the model of “Building a Simple Pendulum” on page 2-11, the four bar mechanism is a closed-loop machine. The two Ground blocks represent points on the same absolute, immobile body, and they close the loop of blocks. The simple pendulum has only one ground and does not close its block connections.

To maintain consistent Body motion direction, make sure the Body coordinate system (CS) port  pairs on each Body follow the sequence CS1 - CS2, CS1 - CS2, etc., for each bar, moving from Ground\_1 to Ground\_2, from right to left, as shown. To make the Joints consistent with the Body motion, the base-follower pairs B-F, B-F, etc., should follow the same right-to-left sequence.

## Configuring the Ground and Joint Blocks

Now configure the Ground blocks with the data from the following table. Grounded coordinate systems (CSs) are automatically created.

### Geometry of the Four Bar Base

This table summarizes the geometry of ground points.

#### Geometric Properties of the Four Bar Grounds

Property	Value	MAT-File Variable
Ground_1 point (m)	[ 0.434 0 0.04 ]	gpoint_1
Ground_2 point (m)	[ -0.433 0 0.04 ]	gpoint_2

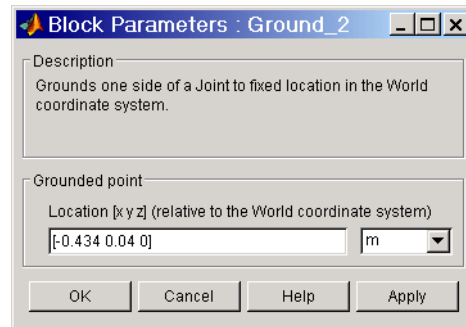
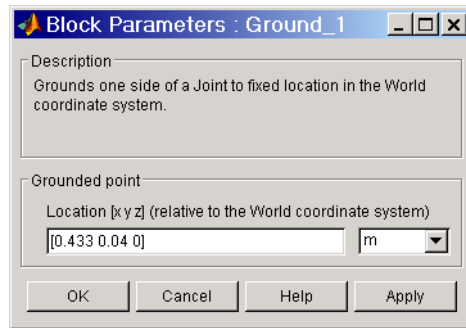
The base of the mechanism has these measurements:

- The base is horizontal, with length 86.7 cm.
- Ground\_1 represents the ground point 43.3 cm to the right of the World CS origin.
- Ground\_2 represents the ground point 43.4 cm to the left of the World CS origin.
- The bottom revolute are 4 cm above the origin ( $x$ - $z$ ) plane.

### Configuring the Ground Blocks

To represent ground points on the immobile base, configure the Ground blocks. Use the variable names if you've loaded `fourbar_data.mat` into your workspace:

- 1 Open Ground\_1 and enter [ 0.434 0 0.04 ] or `gpoint_1` in the **Location** field.
- 2 Open Ground\_2 and enter [ -0.433 0 0.04 ] or `gpoint_2` in the **Location** field.
- 3 Leave both pull-down menus for units at default `m` (meters).



### Configuring the Revolute Joints

The three nongrounded bars move in the plane of your screen ( $x$ - $y$  plane), so you need to make all the Revolute axes the  $z$ -axis (out of the screen):

- 1 Open each Revolute's dialog box in turn. In its **Parameters** area, note on the **Axes** pane that the  $z$ -axis is the default: **Axis of rotation [x y z]** is set to [0 0 1] in each, relative to **Reference csys WORLD**. Leave these defaults.

A Revolute block contains only one primitive joint, a single revolute DoF. So the **Primitive** is automatically Revolute. Its name internal to the block is R1.

- 2 Leave these Revolute joint block defaults and ignore the **Advanced** tab.

The Body CS and base-follower joint directionality should be set up as shown in the block diagram of Figure 2-5 on page 2-42. In the **Connection**

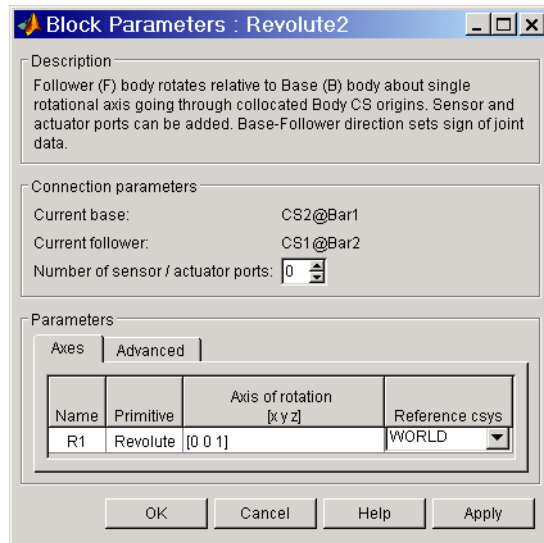
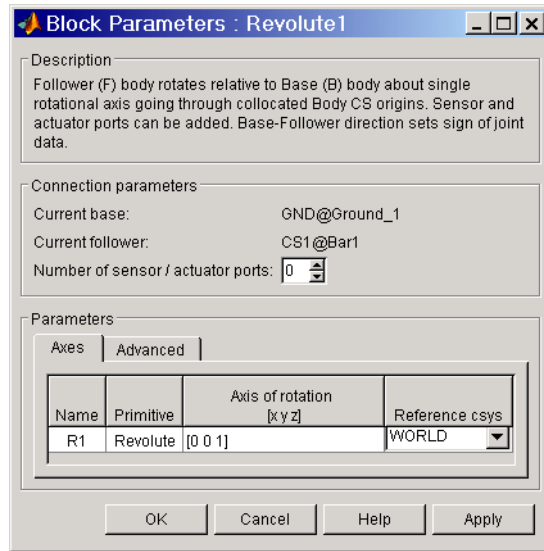


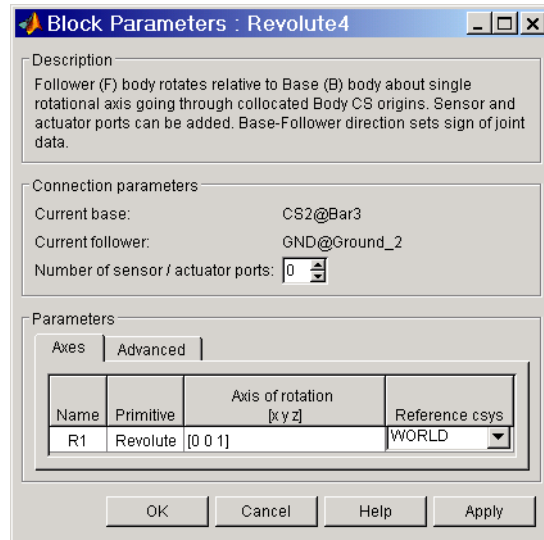
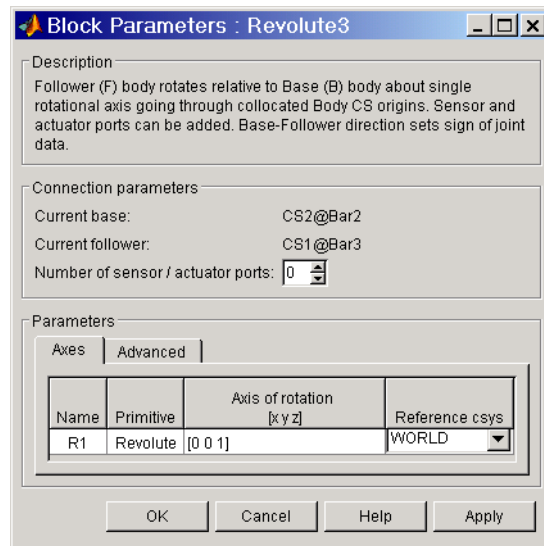
**parameters** area, the default Joint directionality for each Revolute automatically follows the right-to-left sequence of Grounded and Body CSs:

- Revolute1: Base to follower: GND@Gound\_1 to CS1@Bar1
- Revolute2: Base to follower: CS2@Bar1 to CS1@Bar2
- Revolute3: Base to follower: CS2@Bar2 to CS1@Bar3
- Revolute4: Base to follower: CS2@Bar3 to GND@Ground\_2

In this Joint directionality convention:

- At each Joint, the leftward Body moves relative to the rightward Body.
- The rotation axis points in the  $+z$  direction (out of the screen).
- Looking at the mechanism from the front in Figure 2-4 on page 2-37, the positive rotational sense is counterclockwise. All Joint Sensor and Actuator data are interpreted in this sense.





## Configuring the Body Blocks

Setting the Body properties is similar for each bar, but with different

- Mass properties
- Lengths and orientations
- Center of gravity (CG) positions
- Body coordinate systems (CSs)

to be entered into the dialog boxes.

In contrast to the first tutorial, where you specify Body CS properties with respect to the absolute World CS, in this tutorial, you specify Body CS origins on the bars in relative coordinates, displacing Bar1's CS1 relative to Ground\_1, Bar2's CS1 relative to Bar1, and so on, around the machine loop. You can refer the definition of a Body CS to three types of coordinate systems:

- To World
- To the other Body CSs on the same Body
- To the Adjoining CS (the coordinate system on a neighboring body or ground directly connected by a Joint to the selected Body CS).

The components of the displacement vectors for each Body CS origin continue to be oriented with respect to the World axes. The rotation of each Body's CG CS axes is also with respect to the World axes, in the Euler X-Y-Z convention.

The following three tables summarize the body properties for the three bars.

### Bar1 Mass and Body CS Data (MKS Units)

Property	Value	Variable Name
Mass	5.357	m_1
Inertia tensor	[1.07e-3 0 0; 0 0.143 0; 0 0 0.143]	inertia_1
CG Origin	[0.03 0.282 0] from CS1	cg_1
CS1 Origin	[0 0 0] from ADJOINING	cs1_1
CS2 Origin	[0.063 0.597 0] from CS1	cs2_1
CG Orientation	[0 0 83.1] from WORLD	orientcg_1

**Bar2 Mass and Body CS Data (MKS Units)**

Property	Value	Variable Name
Mass	9.028	m_2
Inertia tensor	[1.8e-3 0 0; 0 0.678 0; 0 0 0.678]	inertia_2
CG Origin	[-0.427 0.242 0] from CS1	cg_2
CS1 Origin	[0 0 0] from ADJOINING	cs1_2
CS2 Origin	[-0.87 0.493 0] from CS1	cs2_2
CG Orientation	[0 0 29.5] from WORLD	orientcg_2

**Bar3 Mass and Body CS Data (MKS Units)**

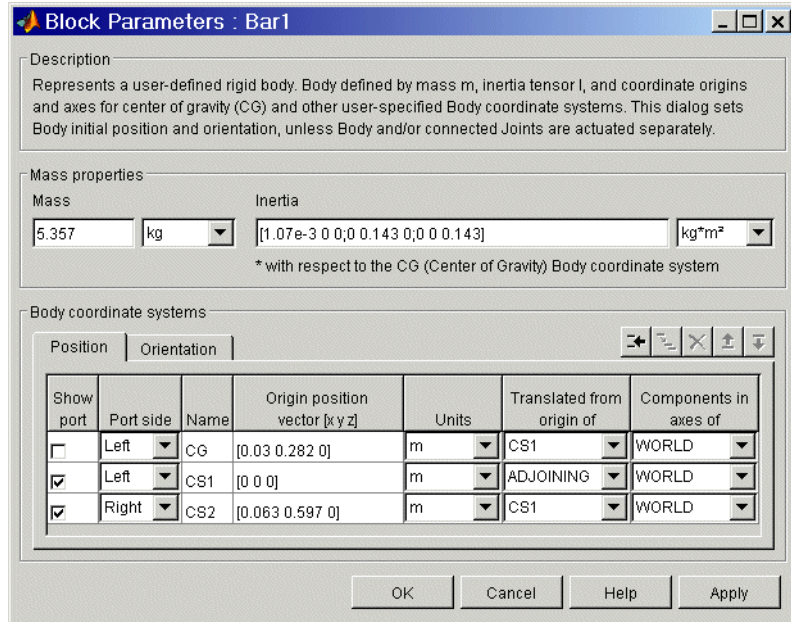
Property	Value	Variable Name
Mass	0.991	m_3
Inertia tensor	[2.06e-4 0 0; 0 1.1e-3 0; 0 0 1.1e-3]	inertia_3
CG Origin	[-0.027 -0.048 0] from CS1	cg_3
CS1 Origin	[0 0 0] from ADJOINING	cs1_3
CS2 Origin	[0 0 0] from ADJOINING	cs2_3
CG Orientation	[0 0 60] from WORLD	orientcg_3

**Configuring the Bodies**

Here are the common steps for configuring the Body dialogs of all three bars. See the three preceding tables for Body dialog box mass property (mass and inertia tensor) entries. The units are MKS: lengths in meters (m), masses in kilograms (kg), and inertia tensors in kilogram-meters<sup>2</sup> (kg-m<sup>2</sup>).

- 1 Open all three Body dialogs for each bar. Enter the mass properties for each from the tables in the **Mass** and **Inertia tensor** fields.
- 2 Now work in the **Body coordinate systems** area, the **Position** pane:

- a Set the **Components in axes of** menu, for each Body CS on each bar, to WORLD.
  - b Leave units as default m (meters).
- 3 Set the Body CS properties for each Body CS on each bar from the data of the preceding tables:
- a Enter the Body CS origin position data for CG, CS1, and CS2 on each bar from the tables or from the corresponding MAT-file variables.
  - b Set the **Translated from origin of** menu entries for each Body CS on each bar according to the “from” information in the tables.
- 4 Select the **Orientation** pane by clicking its tab:
- a Enter the **Orientation vector** for the CG on each bar from the tables or from the corresponding MAT-files variables.
  - b Choose WORLD for **Relative to coordinate system** in each case.
  - c Leave the other fields in their default values.



**Block Parameters : Bar2**

Description  
Represents a user-defined rigid body. Body defined by mass  $m$ , inertia tensor  $I$ , and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately.

Mass properties  
Mass:  kg      Inertia:  kg\*m<sup>2</sup>  
\* with respect to the CG (Center of Gravity) Body coordinate system

Body coordinate systems

Position    Orientation

Show port	Port side	Name	Origin position vector [x y z]	Units	Translated from origin of	Components in axes of
<input type="checkbox"/>	Left	CG	[-0.427 -0.242 0]	m	CS1	WORLD
<input checked="" type="checkbox"/>	Left	CS1	[0 0 0]	m	ADJOINING	WORLD
<input checked="" type="checkbox"/>	Right	CS2	[-0.87 -0.493 0]	m	CS1	WORLD

OK    Cancel    Help    Apply

**Block Parameters : Bar3**

Description  
Represents a user-defined rigid body. Body defined by mass  $m$ , inertia tensor  $I$ , and coordinate origins and axes for center of gravity (CG) and other user-specified Body coordinate systems. This dialog sets Body initial position and orientation, unless Body and/or connected Joints are actuated separately.

Mass properties  
Mass:  kg      Inertia:  kg\*m<sup>2</sup>  
\* with respect to the CG (Center of Gravity) Body coordinate system

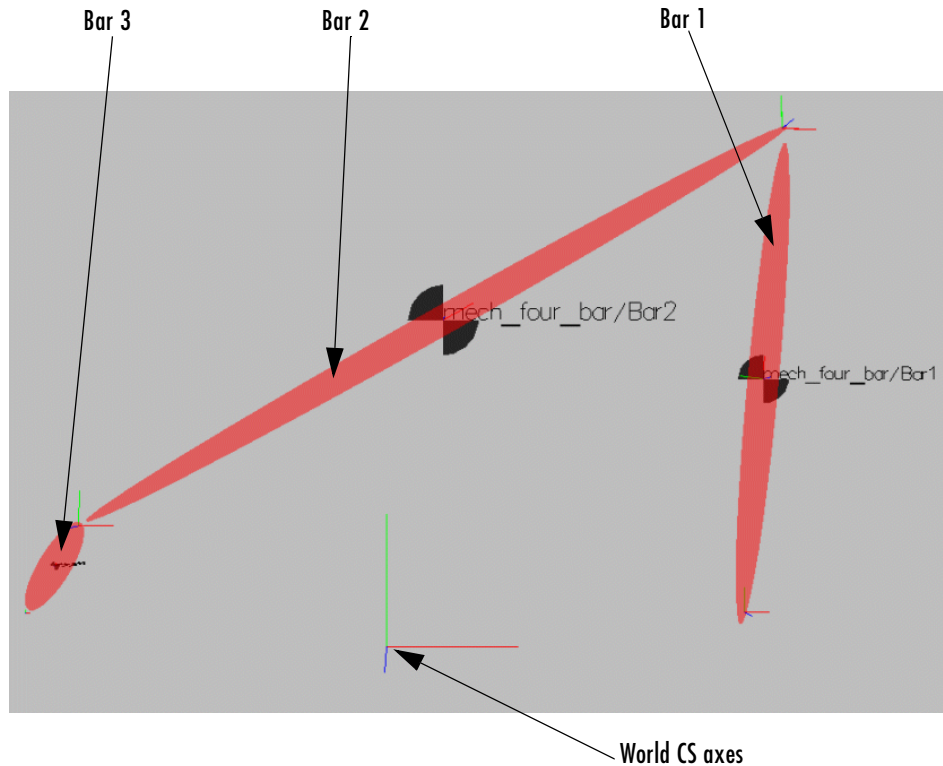
Body coordinate systems

Position    Orientation

Show port	Port side	Name	Origin position vector [x y z]	Units	Translated from origin of	Components in axes of
<input type="checkbox"/>	Left	CG	[-0.027 -0.048 0]	m	CS1	WORLD
<input checked="" type="checkbox"/>	Left	CS1	[0 0 0]	m	ADJOINING	WORLD
<input checked="" type="checkbox"/>	Right	CS2	[0 0 0]	m	ADJOINING	WORLD

OK    Cancel    Help    Apply

The front view of the four bar mechanism, rendered as equivalent ellipsoids in the virtual reality scene, looks like this.



### Sensing Motion and Running the Model

You finish building your model by setting initial conditions and inserting Sensors.

Before you start a machine simulation, you need to set its kinematic state or initial conditions. These include positions/angles and linear/angular velocities. This information, the machine's initial kinematic state, is discussed further in "Modeling Sensors" on page 4-60.



You can sense motion in any model in two basic ways: sensing bodies or sensing joints. Here you sense Joint motion, using Joint Sensor blocks and feeding their Simulink signal outputs to Scope blocks.

---

**Caution** Because they are immobile, ground points cannot be moved, nor do they have any motion to measure.

Therefore you cannot connect Ground blocks to Actuator or Sensor blocks.

---

### Connecting the Joint Sensors

To sense the motion of the Revolute2 and Revolute3 blocks:

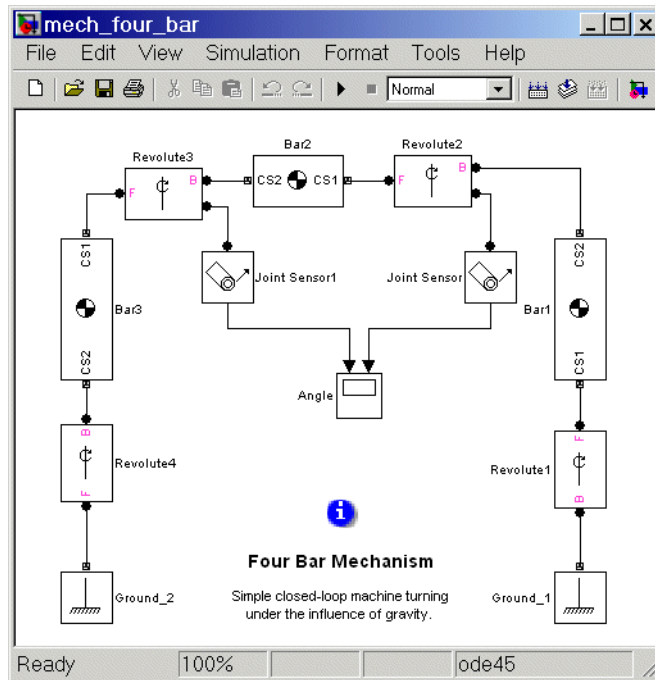
- 1 From the Sensors & Actuators library, drag and drop two Joint Sensor blocks into the model window. Drag Joint Sensor next to Revolute2 and Joint Sensor1 next to Revolute3.
- 2 Before you can attach a Joint Sensor block to a Revolute block, you need to create a new open round connector port  $\circ$  on the Revolute. Open Revolute2's dialog box:
  - a In the **Connection parameters** area in the middle, adjust the spinner menu **Number of sensor/actuator ports** to the value 1. Click **OK**.  
A new connector port  $\circ$  appears on Revolute2.
  - b Connect this connector port to the open round connector port on Joint Sensor.
- 3 Now repeat the same steps with Revolute3:
  - a Create one new connector port  $\circ$  on Revolute 3.
  - b Connect this port to Joint Sensor1.
- 4 Be sure to connect the outports  $>$  of the Sensor blocks to non-SimMechanics blocks. These ports are normal Simulink signals.

### Graphical Plot of Joint Motion with a Scope Block

Here you can view the Joint Sensor measurements of Revolute2 and Revolute3's motions using a Scope block from the Simulink Sinks library:

- 1 Open the Simulink Library Browser. From the Sinks library, drag and drop a Scope block into your model window in between Joint Sensor and Joint Sensor1 blocks. Rename the Scope block “Angle”.
- 2 Open the Angle block. In this scope window’s toolbar, open the **Parameters** box. Under **Axes**, reset **Number of axes** to 2. Click **OK**. A second inport > appears on the Angle block.
- 3 Expand the scope window for ease of viewing.
- 4 Connect the Joint Sensor and Joint Sensor1 block outputs > to the Angle block inports >.
- 5 Open Joint Sensor and Joint Sensor1:
  - a Note in the **Measurements** area, that **Connected to primitive** is set to R1 in both blocks, indicating the first and only primitive revolute inside Revolute2 and Revolute3 to which each Sensor can be connected.
  - b Select the **Angle** check box to measure just the angle. Leave the units in default as deg (degrees). The Simulink line will contain one scalar.

Your completed model should look similar to the mech\_four\_bar demo model.



**Caution** Sensor and Actuator blocks are the *only* blocks that can connect SimMechanics blocks to non-SimMechanics Simulink blocks.

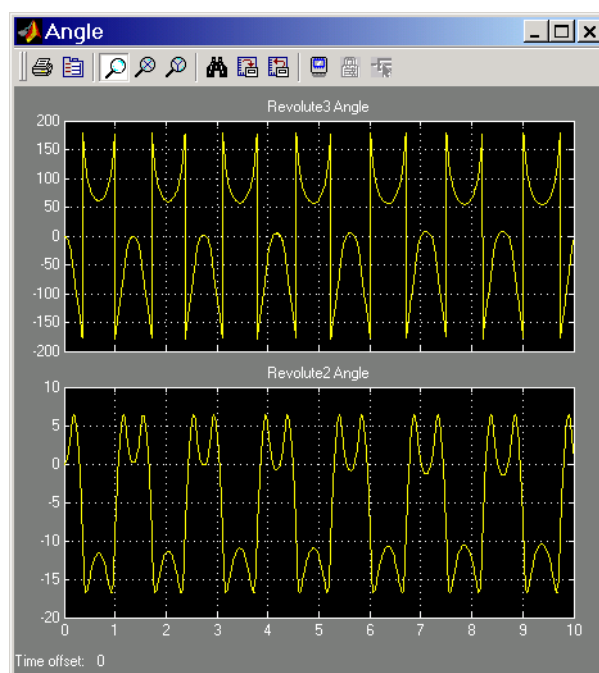
### Configuring and Running the Simulation

Now take the final steps to prepare and start the model:

- 1** In the model window **Simulation** menu, select **Simulation parameters**:
  - a** In the Solver pane, change **Absolute tolerance** to  $1e-6$ .
  - b** Leave the other defaults and click **OK**.
- 2** Now run the model by clicking **Start Simulation** in the Simulink toolbar. The four bar machine will fall under the influence of gravity.

Note some features of the simulation:

- In this example, the machine starts from rest, with the initial velocities at zero. So the initial state of the four bar machine is just the geometric state that you set up in “Setting Up the Block Diagram” on page 2-40.
- The assembly at first falls over to the right, and the Revolute2 angle decreases.
- Bar3 turns all the way around, and Bar2 and Bar1 turn back to the left. The Revolute2 angle reverses direction. Revolute3 sweeps through a complete turn. Angles are mapped to the interval  $(-180^\circ, +180^\circ]$  and exhibit discontinuities.
- The motion repeats periodically, as there is no friction.



### Animation

If you leave your visualization tool open at the time you start the simulation and select the **Animate machine during simulation** check box in the **Visualization** pane of the **Mechanical Environment Settings** dialog box, the visualized machine moves in step with the simulation.

You can now compare the animated motion with the Scope plots of the Revolute2 and Revolute3 angles.



# Understanding Mechanical Concepts

---

This chapter reviews the mathematical representations that SimMechanics uses to implement position, orientation, and motion. Special technical terms associated with mechanics and with SimMechanics are also summarized. Beginning users might find this chapter useful review after constructing a few simple machines. This chapter assumes some familiarity with vector algebra and analysis.

Representing Body Positions and  
Orientations (p. 3-2)

Summary of Technical Vocabulary  
(p. 3-11)

Review of translational and rotational motion and how  
SimMechanics implements kinematic states

Summary of special terms used in SimMechanics with  
links to the Glossary

## Representing Body Positions and Orientations

Machines are composed of bodies, which have relative degrees of freedom (DoFs). Bodies have positions, orientations, mass properties, and sets of Body coordinate systems. Joints represent the motions of the bodies.

- A machine's *geometry* consists of its static body features before starting a simulation: positions, orientations, and Body coordinate systems.
- A machine's *kinematics* consists of the positions/orientations and their derivatives of all degrees of freedom (DoFs) of all bodies at any instant during the machine's motion.

The full description of a machine's motion includes not only its kinematics, but specification of its observers, who define reference frames (RFs) and coordinate systems (CSs) for measuring the machine motion.

This section summarizes observer coordinate systems, measuring body motion, forms of body rotation, and how SimMechanics represents the state of a machine's motion.

- “Reference Frames and Coordinate Systems”
- “Relating Moving Coordinate Systems” on page 3-3
- “Observing Motion in Different Coordinate Systems” on page 3-4
- “Observing a Translating, Rotating Rigid Body” on page 3-6
- “How SimMechanics Represents Body Rotations” on page 3-7
- “Kinematics and the Machine State” on page 3-9

All vectors and tensors (matrices), unless otherwise noted, are Cartesian with three and nine, respectively, spatial components measured by orthogonal or rectangular coordinate axes. This section assumes basic knowledge of vector algebra and analysis. The books of Goldstein [2] and Murray, Li, and Sastry [11] present coordinate transformations, rotations, and rigid body kinematics in detail.

### Reference Frames and Coordinate Systems

The *reference frame* of an observer is an observer's state of motion, which has to be measured by other observers. The Newtonian dynamics of a mechanical system take their simplest form in the special set of *inertial* RFs, the set of all frames moving at uniform velocities with respect to one another. Within an RF,

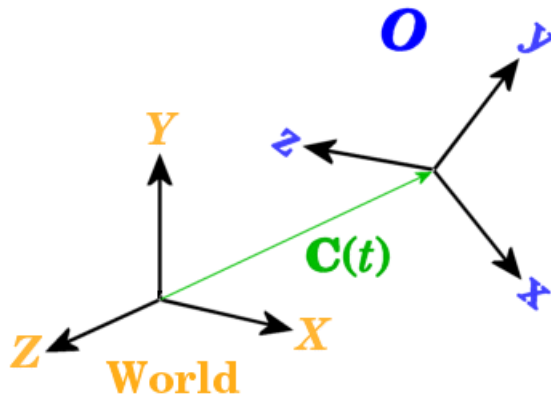


you can pick any point as a *coordinate system origin*, then set up Cartesian (orthogonal) axes there.

SimMechanics uses a master inertial RF called *World*. A CS origin and axis triad are also defined in *World*. *World* can mean either the RF or the CS, although in most contexts, the coordinate system is indicated. For SimMechanics, *World* defines absolute rest and a universal coordinate origin and axes independent of any bodies and grounds in a machine.

## Relating Moving Coordinate Systems

Add a second CS, called *O*, whose origin is translating with respect to the *World* origin and whose axes are rotating with respect to the *World* axes.



A vector  $C$  represents the origin of *O*. Its head is at the *O* origin and its tail at the *World* origin. The *O* origin translates as some arbitrary function of time  $C(t)$ .

The orthogonal unit vectors  $\{u(x), u(y), u(z)\}$  define the coordinate axes of *O*.

- This set is tilted with respect to the *World* coordinate axes  $X, Y, Z$ , with unit vectors  $\{e(x), e(y), e(z)\}$ . The tilt changes with time.

- You can express the set  $\{\mathbf{u}(x), \mathbf{u}(y), \mathbf{u}(z)\}$  as a linear combination of the basis  $\{\mathbf{e}(x), \mathbf{e}(y), \mathbf{e}(z)\}$  in terms of nine coefficients:

$$\mathbf{u}(x) = R_{xx} \mathbf{e}(x) + R_{yx} \mathbf{e}(y) + R_{zx} \mathbf{e}(z)$$

$$\mathbf{u}(y) = R_{xy} \mathbf{e}(x) + R_{yy} \mathbf{e}(y) + R_{zy} \mathbf{e}(z)$$

$$\mathbf{u}(z) = R_{xz} \mathbf{e}(x) + R_{yz} \mathbf{e}(y) + R_{zz} \mathbf{e}(z)$$

These are relationships between *vectors* (not *vector components*) and are independent of reference frame and coordinate system.

- You obtain the *components* of the  $\mathbf{u}$ 's in World by taking the scalar products of the  $\mathbf{u}$ 's with the  $\mathbf{e}$ 's:

$$u_x(x) = R_{xx}, u_y(x) = R_{yx}, u_z(x) = R_{zx}$$

$$u_x(y) = R_{xy}, u_y(y) = R_{yy}, u_z(y) = R_{zy}$$

$$u_x(z) = R_{xz}, u_y(z) = R_{yz}, u_z(z) = R_{zz}$$

The time-dependent  $R$  coefficients represent the orientation of the  $\mathbf{u}$ 's with respect to the  $\mathbf{e}$ 's. You can replace the labels  $(x,y,z)$  by  $(1,2,3)$ .

- The *components* of any vector  $\mathbf{v}$  measured in World are  $\mathbf{e}(i) \cdot \mathbf{v}$ . Represent them by a column vector  $\mathbf{v}_{\text{World}}$ . The components of  $\mathbf{v}$  in  $O$  are  $\mathbf{u}(i) \cdot \mathbf{v}$ . Represent them by a column vector  $\mathbf{v}_O$ . The two sets of components are related by the matrix transformation  $\mathbf{v}_{\text{World}} = R^* \mathbf{v}_O$ . The coefficients  $R$  form a matrix whose *columns* are the components of the  $\mathbf{u}$ 's in World:

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{bmatrix}$$

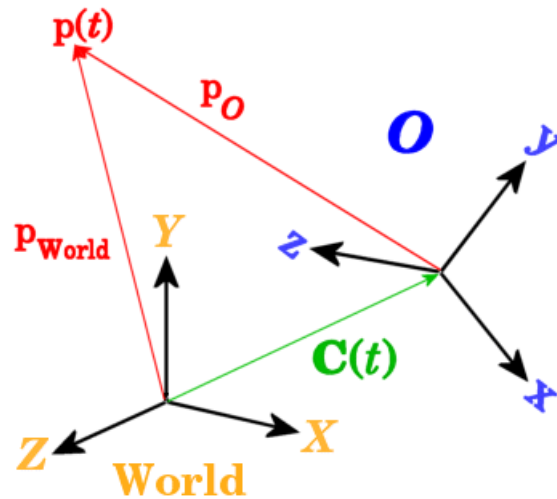
The orthogonality and unit length of the  $\mathbf{u}$ 's guarantee that  $R$  is an orthogonal rotation matrix satisfying  $RR^T = R^T R = I$ , the identity matrix.  $R^T$  is the transpose of  $R$  (switch rows and columns). Thus  $R^{-1} = R^T$ .

- Rotations always follow the *right-hand rule*.

## Observing Motion in Different Coordinate Systems

To the two observer CSs, World and  $O$ , now add a third point  $\mathbf{p}(t)$  in arbitrary motion.  $\mathbf{p}$  could represent a point mass, the center of gravity (CG) of an extended body, or a point fixed in a moving rigid body, for example. The two

observers describe the motion of this point in different ways, related to one another by time-dependent World-to- $O$  coordinate transformations.



The motion of  $p$  is given by its column vector components in some CS. The components of  $p$  as measured in World are a column vector  $p_{World}$  and, measured in  $O$ , are a column vector  $p_O$ . The two descriptions are related by

$$p_{World}(t) = C_{World}(t) + R(t) * p_O(t)$$

Thus the motion as measured by  $p_{World}$ , when transformed and observed by  $O$  as  $p_O$ , has additional time dependence arising from  $C(t)$  and  $R(t)$ .

### Relating Velocities Observed in Different Coordinate Systems

Differentiate once with respect to time the relationship between  $p_{World}$  and  $p_O$ . The result relates the velocity of  $p$  as measured by  $O$  to the velocity as measured in World.

$$dp_{World}/dt = dC_{World}/dt + R * dp_O/dt + dR/dt * p_O$$

The section “The Angular Velocity of a Body from Its Rotation Matrix” on page 3-6 explains how to express the third term in a simpler form.

## Observing a Translating, Rotating Rigid Body

You can obtain the special case essential for rigid body motion by letting  $O$  be the CG CS of an extended rigid body (the origin of  $O$  at the CG itself) and letting  $\mathbf{b}$  be a point somewhere in the same body. Since a moving body in general accelerates both translationally and rotationally,  $O$  is noninertial.

The rotation matrix  $R$  now describes the rotational motion of the body in terms of the rotation of the CG CS axes with respect to the World axes. Furthermore,  $\mathbf{b}$  is now fixed in the body itself and does not move in  $O$ . All of its motion as seen by World is due to  $R$  and  $C$ , and  $d\mathbf{b}_O/dt = 0$ .

## The Angular Velocity of a Body from Its Rotation Matrix

Continue to identify  $O$  with the body CG CS and  $\mathbf{b}$  as a point fixed in the body. The vector components of  $\mathbf{b}$  are observed by World as  $\mathbf{b}_{\text{World}}$  and by the CG CS as  $\mathbf{b}_{\text{Body}}$ . In the body, the point is immobile:  $d\mathbf{b}_{\text{Body}}/dt = 0$ . Its velocity observed by World is composed of the translational and rotational motion of the entire rigid body.

$$d\mathbf{b}_{\text{World}}/dt = d\mathbf{C}_{\text{World}}/dt + dR/dt * \mathbf{b}_{\text{Body}}$$

Because  $RR^T = I$ ,  $(dR/dt)*R^T + R*(dR^T/dt) = 0$ . Insert  $R^T R = I$  to the left of  $\mathbf{b}_{\text{Body}}$  and define an antisymmetric matrix  $\Omega = +(dR/dt)*R^T = -R*(dR^T/dt)$ . Its components are  $\Omega_{ijk} = +\sum_j \varepsilon_{ijk}\omega_j$ .

$$\Omega = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

where  $\omega$  is the body's angular velocity in the World CS.

$$\begin{aligned} d\mathbf{b}_{\text{World}}/dt &= d\mathbf{C}_{\text{World}}/dt + \Omega * R * \mathbf{b}_{\text{Body}} \\ &= d\mathbf{C}_{\text{World}}/dt + \omega_{\text{World}} \mathbf{x} (R * \mathbf{b}_{\text{Body}}) \end{aligned}$$

The motion of  $\mathbf{b}_{\text{Body}}$  decomposes into the motion of the body's CG plus the angular rotation of  $\mathbf{b}_{\text{Body}}$  relative to the CG, all measured in World.

The permutation symbol  $\varepsilon_{ijk}$  is +1 if  $ijk$  is an even permutation (123 or any cyclic permutation thereof) and -1 if  $ijk$  is an odd permutation (321 or any cyclic permutation thereof). It changes sign upon switching any two indices and

vanishes if any two indices are equal. The components of the cross (vector) product  $\mathbf{c} = \mathbf{a} \times \mathbf{b}$  of two vectors  $\mathbf{a}$  and  $\mathbf{b}$  are  $c_i = \sum_{jk} \varepsilon_{ijk} a_j b_k$ .

## How SimMechanics Represents Body Rotations

You state a body's orientation by specifying the tilt of its center of gravity coordinate system (CG CS) axes relative to some other set of axes, either the CG CS axes of another body or the World CS axes. Zero tilt is represented by “no rotation” or the rotational identity.

A general rotation of a body in three dimensions has three independent degrees of freedom. There are many ways to represent these degrees of freedom. SimMechanics implements the following forms in the Body and related Body Actuator, Body Sensor, and RotationMatrix2VR blocks.

### The Axis-Angle Form

The angle-axis form of a rotation is the most basic representation. Specify a rotation axis (unit vector)  $\mathbf{n}$ , then rotate by the right-hand rule about that axis by some angle  $\theta$ .

The axis-angle form is usually written as a 4-vector:  $[n_x n_y n_z \theta]$ . Of the four numbers, three are independent, because  $\mathbf{n}^* \mathbf{n} = n_x^2 + n_y^2 + n_z^2 = 1$ . That is,  $\mathbf{n}$  specifies only a direction, not a length.

To describe continuous rotation in time, you treat  $\mathbf{n}$  and  $\theta$  as functions of time.

### The Rotation Matrix Form

The axis-angle form defines the rotation matrix  $R$  written in the exponential form  $R = \exp(\theta \mathbf{n}^* \mathbf{J})$ , where the  $\mathbf{J}^k$  are real, antisymmetric matrices, and  $\mathbf{n}^* \mathbf{J} = n_x \mathbf{J}^1 + n_y \mathbf{J}^2 + n_z \mathbf{J}^3$ . The rotation matrix  $R$  is orthogonal:  $RR^T = R^T R = I$ .

The  $\mathbf{J}$  matrices are related to the antisymmetric permutation symbol  $\varepsilon_{ijk}$ .

$$(\mathbf{J}^j)_{ik} = \varepsilon_{ijk}$$

The formal exponential for  $R$  is reduced to a closed form by the *Rodrigues identity*:

$$R = \exp(\theta \mathbf{n}^* \mathbf{J}) = I + (\mathbf{n}^* \mathbf{J}) \sin \theta + (\mathbf{n}^* \mathbf{J})^2 (1 - \cos \theta)$$

where  $I$  is the identity matrix and  $\mathbf{n}^* \mathbf{J}$  is given by

$$\mathbf{n}^* \mathbf{J} = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}$$

The inverse of  $R$  is identical to its transpose  $R^T$ . But you can also obtain the inverse by replacing  $\theta$  with  $-\theta$  or by reversing the direction of  $\mathbf{n}$ .

To describe continuous rotation in time, you treat  $\mathbf{n}$  and  $\theta$  as functions of time.

### The Euler Angle Form

An alternative representation for  $R$  is to rotate, in succession, about three independent axes, by three independent *Euler angles*. The full rotation  $R$  relative to *World* composes by multiplying the matrices successively on the *left*:

$$R = R_3 * R_2 * R_1$$

The full rotation  $R$  relative to the *body* composes by multiplying the matrices successively on the *right*:

$$R = R_1 * R_2 * R_3$$

One Euler angle convention is to rotate about one body coordinate axis (which rotates the other two); then rotate about a second body coordinate axis (rotated from its original direction); and lastly, rotate about the first body coordinate axis again (which axis has been rotated into a new direction by the second rotation). The rotation axis sequence *Z-X-Z* is the most common, using  $\phi$ ,  $\theta$ ,  $\psi$  as the first, second, and third angles, respectively:  $R = R_1(\phi) * R_2(\theta) * R_3(\psi)$ .

A two-dimensional rotation about a fixed axis requires one angle. For example, rotating the  $x$ - and  $y$ -axes about the  $z$ -axis by  $\phi$  is represented by

$$R(\phi) = \begin{bmatrix} \cos \phi & -\sin \phi & 0 \\ \sin \phi & \cos \phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To describe continuous rotation in time, you treat the Euler angles as functions of time.

## The Quaternion Form

A *quaternion* represents a three-dimensional rotation as a four-component row vector of unit length:

$$q = [n_x \sin(\theta/2) \quad n_y \sin(\theta/2) \quad n_z \sin(\theta/2) \quad \cos(\theta/2)] = [\mathbf{q}_v \quad q_s],$$

with  $q^*q = \mathbf{q}_v^* \mathbf{q}_v + q_s^2 = 1$ . The vector  $\mathbf{n} = (n_x, n_y, n_z)$  is a three-component unit vector,  $\mathbf{n}^* \mathbf{n} = 1$ , and represents the axis of rotation. The rotation angle about that axis is  $\theta$ . To describe continuous rotation in time, you treat  $\mathbf{n}$  and  $\theta$  as functions of time.

You can construct the equivalent rotation matrix  $R$  from  $q$ .

$$R = (2q_s^2 - 1)I + 2q_s Q_v + 2\mathbf{q}_v \mathbf{q}_v^T$$

$$(Q_v)_{ik} = \sum_j \varepsilon_{ijk}(\mathbf{q}_v)_j$$

The term  $\mathbf{q}_v \mathbf{q}_v^T$  is the *outer product* of  $\mathbf{q}_v$  with itself, the 3-by-3 matrix of  $\mathbf{q}_v$  components multiplied by each other.

## Kinematics and the Machine State

*Kinematics* is the description of a machine's motion without regard to forces, torques, and the mass properties of bodies. Since accelerations are proportional to forces and torques, you only need the positions and their first derivatives (velocities) to integrate a machine's motion, if you know the mass properties of the bodies and the forces and torques applied to them.

The state of motion of a mechanical system is the set of the instantaneous positions and orientations of all its bodies and their first derivatives (linear and angular velocities). In SimMechanics, body positions/orientations are *relative*: one body's state is specified relative to its neighbors. The absolute positions and velocities of the bodies' states are determined via the machine's connection to one or more grounds. These grounds are at rest in World, although they do not have to coincide with the World origin.

The relative position and orientation of a body with respect to a neighbor make up a degree of freedom (DoF). The fundamental DoFs are translational (one body sliding relative to another along a prismatic axis) and rotational (one body rotating relative to another about a revolute axis, or one body pivoting relative to another about a spherical pivot point).

SimMechanics represents DoFs by Joint blocks connected between Body blocks. Bodies without Joints have no DoFs in SimMechanics and acquire DoFs only by having Joints connected to them. SimMechanics represents the *machine state* by the positions (prismatic), angles (revolutes or sphericals), and linear/angular velocities of all Joints.

Refer to “Counting Degrees of Freedom” on page 4-67 and the command reference for `mech_stateVectorMgr` for identification of DoFs in SimMechanics and elaboration of the machine state.



## Summary of Technical Vocabulary

This section summarizes the special technical terms used in this guide and points to more detailed definitions in the “Glossary”:

- “Topology of Mechanical Systems”
- “Coordinate Systems and Reference Frames”
- “Bodies and Mass Properties” on page 3-12
- “Joints, Constraints, and Drivers” on page 3-12
- “Sensors and Actuators” on page 3-12

### Topology of Mechanical Systems

Mechanical systems can be characterized by their *topology*, or the global connectivity of their elements. A SimMechanics model’s elements are the Body blocks, and the connections are the Joints. A complex system in general is a mixture of these basic topological classes: *open systems*, with subclasses of *open chains* and *open trees*, and *closed systems*, which contain loops.

### Coordinate Systems and Reference Frames

SimMechanics uses a family of *coordinate systems* (CSs) and *reference frames* (RFs).

SimMechanics has a built-in inertial RF called *World*. Ground blocks representing ground points are at rest in this RF. You can think of ground points as fixed points on an infinitely large, immobile body. The World RF defines the state of “no motion” or “at rest” for all systems.

World is also a CS, with an origin and axes:

- +x points right
- +y points up (gravity in  $-y$  direction)
- +z points out of the screen, in three dimensions

Users can define additional *Local CSs*:

- *Grounded CSs* attached to Ground blocks at rest in the World RF but displaced from the World CS origin
- *Body CSs* fixed in the system’s rigid bodies and moving rigidly with the bodies

In general bodies accelerate as they move. So Body CSs in general live in noninertial RFs.

### **Bodies and Mass Properties**

A *body* is defined in SimMechanics by its mass properties (*mass* and *inertia tensor*), its position and orientation, and user-specified attached Body CSs. Bodies carry no DoFs.

A body's position is specified by the position of its center of gravity (CG). The CG point is also the origin of the body's CG coordinate system (CG CS). The orientation of the CG CS axes is user-specified. The body also can have other Body CSs attached to it. The origins and axes of these CSs are user-specified.

The orientation of the body itself is set by the components of its inertia tensor, which are always interpreted in the body's CG CS axes. The absolute orientation of the body is then determined by the orientation of that body's CG CS axes with respect to World.

### **Joints, Constraints, and Drivers**

*Joints* carry the *degrees of freedom* (DoFs) of a system in SimMechanics, but none of its mass or inertia. DoFs represent the possible relative motions between pairs of bodies and are specified by translational and rotational axes. Joints can be *primitive* or *composite*, *constrained* or *unconstrained*, and *assembled* or *disassembled*.

*Constraints* are functional relationships between the DoFs that hold regardless of applied forces/torques. *Drivers* are special constraints that specify the motion of DoFs as user-defined functions of time.

### **Sensors and Actuators**

*Sensors* measure the motions of any body or joint, while *actuators* specify the motion of a body or joint in various ways. Important specialized Actuator blocks are the following:

- The Body Actuator, which applies forces to a body
- The Joint Actuator, which applies forces or motions to a joint
- The Joint Initial Condition Actuator, which sets a joint's initial conditions
- The Joint Stiction Actuator, which applies static and kinetic friction to joint motion

# Modeling Mechanical Systems

---

SimMechanics gives you a complete set of block libraries for modeling machine parts and connecting them into a Simulink block diagram.

Modeling Machines (p. 4-2)	How to create a valid mechanical model with SimMechanics and Simulink
Modeling Bodies (p. 4-8)	How to represent the machine environment and the mass and geometric properties of bodies
Modeling Joints (p. 4-17)	How to represent degrees of freedom as joints
Modeling Constraints and Drivers (p. 4-34)	How to represent time-independent and time-dependent constraints on relative body motions
Modeling Actuators (p. 4-40)	How to apply forces, motions, and initial conditions to machines
Modeling Force Elements (p. 4-55)	How to represent internal forces within machines
Modeling Sensors (p. 4-60)	How to measure motions and forces in machines and create feedback loops to model internal forces
Checking Model Validity (p. 4-64)	Special restrictions and requirements for block diagram models with SimMechanics blocks

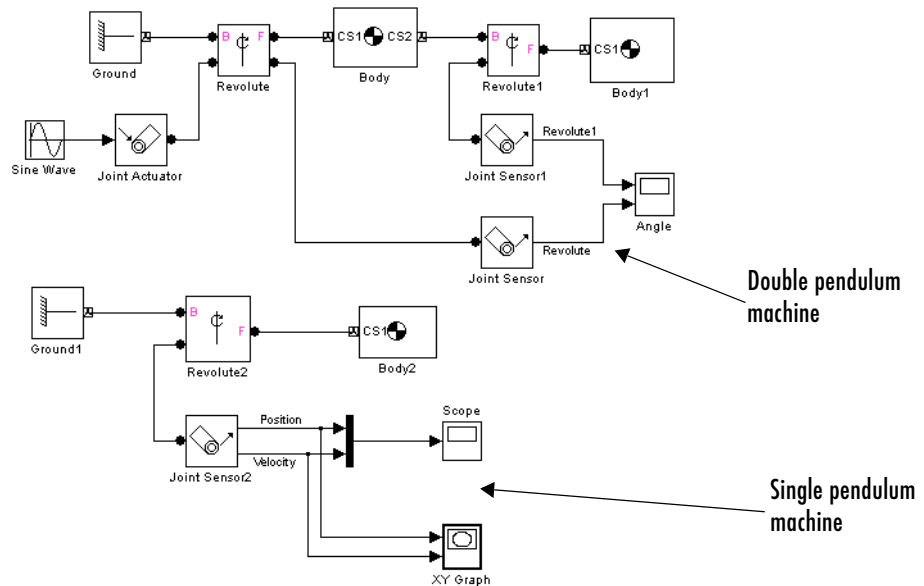
Refer to the “Understanding Mechanical Concepts” chapter to review kinematics. If you need more information on rigid body mechanics, consult the physics and engineering literature, beginning with the “Selected Bibliography.” Classic engineering mechanics texts include Goodman and Warner [3], [4] and Meriam [10]. The books of Goldstein [2] and José and Saletan [6] are more theoretically oriented.

## Modeling Machines

In SimMechanics the term *machine* refers to a dynamic system that includes at least one rigid body. SimMechanics provides a library of Simulink blocks that allow you to create Simulink models of machines.

### About SimMechanics Models

A SimMechanics *model* consists of a *block diagram* comprising one or more *schematics*, each of which is a set of connected blocks representing a machine. For example, the following model contains two schematics.



### Comparison to Other Simulink Models

A SimMechanics model differs significantly from other Simulink models in how it represents a machine. An ordinary Simulink model represents the *mathematics* of a machine's motion, i.e., the algebraic and differential equations that predict the machine's future state from its present state. The mathematical model enables Simulink to simulate the machine. By contrast, a SimMechanics model represents the *physical structure* of a machine, the

geometric and kinematic relationships of its component bodies. SimMechanics converts this structural representation to an internal, equivalent mathematical model. This saves you the time and effort of developing the mathematical model yourself.

## Creating a SimMechanics Model

You create a SimMechanics model in much the same way you create any other Simulink model. First, you open a Simulink model window. Then you drag instances of SimMechanics and other Simulink blocks from the Simulink block libraries into the window and draw lines to interconnect the blocks (see “Connecting SimMechanics Blocks” on page 4-3).

The SimMechanics block library provides the following blocks specifically for modeling machines:

- Body blocks that represent a machine’s components and the machine’s immobile environment, or ground (see “Modeling Bodies” on page 4-8)
- Joint blocks that represent the degrees of freedom of one body relative to another body or to a point on ground (see “Modeling Joints” on page 4-17)
- Constraint and Driver blocks that restrict motions of or impose motions on bodies relative to one another (see “Modeling Constraints and Drivers” on page 4-34)
- Initial condition blocks that specify the initial state of the machine
- Actuator blocks that specify forces or motions applied to joints and bodies
- Sensor blocks that output the forces and motions of joints and blocks

You can use blocks from other Simulink libraries in SimMechanics models. For example, you can connect the output of SimMechanics Sensor blocks to Scope blocks from the Simulink Sinks library to display the forces and motions of your model’s bodies and joints. Similarly, you can connect blocks from the Simulink Sources library to SimMechanics Driver blocks to specify relative motions of your machine’s bodies.

## Connecting SimMechanics Blocks

In general, you connect SimMechanics blocks in the same way you connect other Simulink blocks: by drawing lines between them. Significant differences exist, however, between connecting standard Simulink blocks and connecting SimMechanics blocks. The following sections discuss these differences.

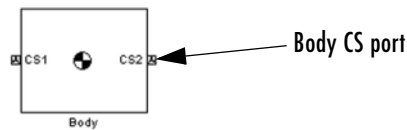
## Connection Lines

The lines that you draw between standard Simulink blocks, called signal lines, represent inputs to and outputs from the mathematical functions represented by the blocks. By contrast, the lines that you draw between SimMechanics blocks, called *connection lines*, represent physical connections and relationships among the bodies represented by the blocks.

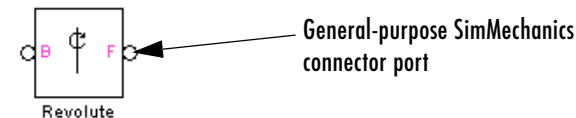
You can draw connection lines only between specialized connection ports available only on SimMechanics blocks (see next section) and you cannot branch existing connection lines. Connection lines appear as solid black when connected and as dashed red lines when either end is unconnected.

## Connector Ports

Standard Simulink blocks have input and output ports. By contrast, most SimMechanics blocks contain only specialized *connector ports* that permit you to draw connection lines among SimMechanics blocks. SimMechanics connector ports are of two types: Body CS ports and general-purpose ports. Body CS ports appear on Body and Ground blocks and define connection points on a body or ground. Each is associated with a local coordinate system whose origin specifies the location of the associated connection point on the body.

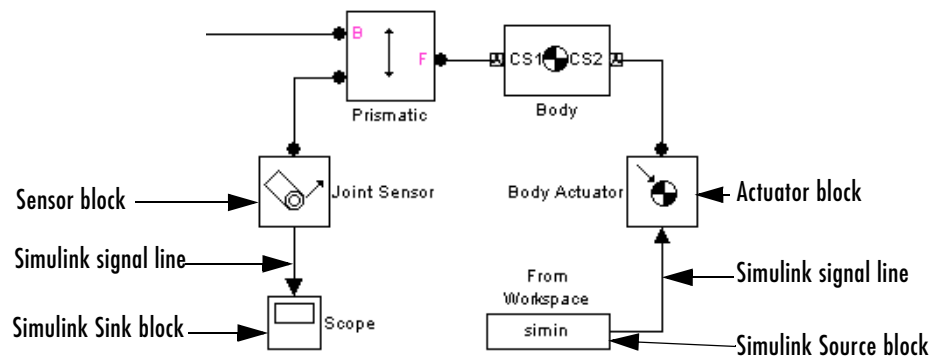


General-purpose connector ports appear on Joint, Constraint, Driver, Sensor, and Actuator blocks. They permit you to connect Joints to Bodies and connect Sensors and Actuators to Joints, Constraints, and Drivers. General-purpose connector ports appear as circles on the block icon. The circle is unfilled if the port is unconnected and filled if the port is connected.



## Interfacing SimMechanics Blocks to Simulink Blocks

SimMechanics Actuator blocks (see “Modeling Actuators” on page 4-40) contain standard Simulink input ports. Thus, you can connect standard Simulink blocks to a SimMechanics model via Actuator blocks. Similarly, SimMechanics Sensor blocks contain output ports (see “Modeling Sensors” on page 4-60). Thus, you can connect a SimMechanics model to Simulink blocks via Sensor blocks.



## Setting SimMechanics Block Properties at the Command Line

You cannot use the Simulink `set_param` and `get_param` commands to set or get SimMechanics block parameters. Instead, you must set block parameters via the block dialog boxes. You can open the dialogs by double-clicking the block, or by right-clicking the block and selecting **Open block**.

## Creating SimMechanics Subsystems

Large, complex block diagram models are often hard to analyze. Enclosing functionally-related groups of blocks in subsystems alleviates this difficulty and facilitates reuse of block groups in different models.

You can create subsystems containing SimMechanics blocks that you can connect to other SimMechanics blocks. You do this in two ways:

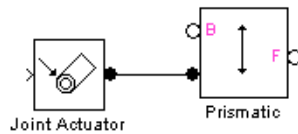
- Automatically
- Manually

The Simulink User's Guide explains more about creating subsystems.

## Creating a Subsystem Automatically

To create a SimMechanics subsystem automatically:

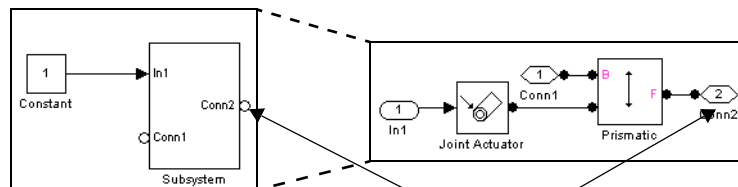
- 1 Create the subsystem block diagram in your model window, leaving unconnected ports for external connections.



- 2 Group-select the subsystem block diagram.

- 3 Select the **Make subsystem** command from the Simulink window.

The command replaces the block diagram with a Subsystem block containing the selected block diagram. The command also creates and connects SimMechanics Connection Port blocks for the ports that you left unconnected in the block diagram. The Connection Port blocks in turn create connector port icons on the subsystem icon, enabling you to connect external SimMechanics blocks to the new subsystem.



Subsystem connector port



## Creating a Subsystem Manually

Sometimes you need to make a subsystem configured differently from an automatically created one. To create a SimMechanics subsystem manually:

- 1 Drag a Subsystem block into your model window.
- 2 Open the Subsystem block.
- 3 Create the subsystem block diagram in the subsystem window.
- 4 Drag a Connection Port block from the SimMechanics Utilities library into the subsystem window for each port that you want to be available externally.
- 5 Connect the external connector ports to the Connection Port blocks.

## Creating Custom SimMechanics Blocks with Masks

You can create your own SimMechanics blocks from subsystems, for example, a spring-loaded Joint block or a sphere Body block. To do this, create a block diagram that implements the functionality of your custom block, enclose the diagram as a subsystem, and add a mask (i.e., user interface) to the subsystem. To facilitate sharing your custom blocks across models or with other users, create a Simulink block library and add these masked subsystem blocks to the library. The Simulink User's Guide explains how to create custom blocks with masks.

### Modeling Bodies

The basic components of any machine are its constituent rigid bodies. In SimMechanics, the term *body* refers to any point or spatially extended object that has mass. SimMechanics bodies, unlike physical bodies, do not have degrees of freedom. The SimMechanics Bodies library contains two blocks for representing bodies in a Simulink model:

- Ground

Models a point on an ideal body of infinite mass and extent that serves as a fixed environment for machines (see “Modeling Grounds” following).

- Body

Models rigid bodies of finite mass and extent, including their attached body coordinate systems (see “Modeling Finite Rigid Bodies” on page 4-9).

### Modeling Grounds

In SimMechanics, ground refers to a body of infinite mass and size that acts both as a reference frame at rest for a machine as a whole and as a fixed base for attaching machine components, e.g., the factory floor on which a robot stands. SimMechanics Ground blocks enable you to represent points on ground in your model. This in turn enables you to specify the degrees of freedom that your system has relative to its environment. You do this by connecting Joint blocks representing the degrees of freedom between the Body blocks representing parts of your machine and the Ground blocks representing ground points.

Each Ground block has a single connector port to which you can connect a Joint block that can in turn be connected to a single Body block. Each Ground block therefore allows you to represent the degrees of freedom between a single part of your model and its environment. If you want to specify the motion of other parts of your machine relative to the environment, you must create additional Ground blocks.

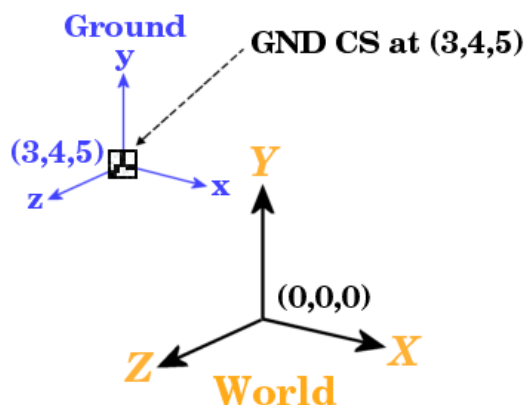
---

**Note** To be valid, a SimMechanics model must contain at least one Ground block connected to a Body block via a Joint block.

---

## The World Coordinate System

SimMechanics uses an internal master coordinate system and reference frame called World. All grounds are at rest in World. The connector port of each Ground block defines a grounded coordinate system called GND. The GND coordinate system's axes are parallel to World. By default the origin of the grounded coordinate system coincides with the origin of the World coordinate system. The **Grounded point** field of a Ground block's dialog box allows you to move the origin of GND to some other point in the World coordinate system, as in the example "Building a Simple Pendulum" on page 2-11.



The GND coordinate system allows you to specify the positions and motions of parts of your machine relative to fixed points in the environment.

## Modeling Finite Rigid Bodies

The SimMechanics Body block enables you to model rigid bodies of finite mass and extent. A body is rigid if its internal parts cannot move relative to one another.

### About Body Blocks

A Body block allows you to specify the following attributes of a rigid body.

**Mass Properties.** These include the body's mass, which determines its response to translational forces, and its inertia tensor, which determines its response to rotational torques.

**Body Coordinate Systems.** By default a Body block defines three local coordinate systems, one associated with a body's center of gravity, labeled CG, and two others, labeled CS1 and CS2, respectively, associated with two other points on the body that you can specify. You can create additional body coordinate systems or delete them as necessary.

A Body block's dialog box allows you to specify a Body CS's origin (see "Setting a Body CS's Position" on page 4-12) and orientation (see "Setting a Body CS's Orientation" on page 4-14). The origin and orientation of a body's CG CS specify the body's initial location and orientation. The origins of the other body coordinate systems specify the initial locations of other points on the body.

SimMechanics allows flexibility in specifying the origins and orientations of a body's coordinate systems. You can specify the origin and orientation of a body CS relative to

- The World CS
- Any other CS on the same body
- The *Adjoining CS*, the CS on the neighboring body or ground directly connected by a Joint, Constraint, or Driver to the selected Body CS you are configuring.

This simplifies creation and maintenance of models. The only limitation is that you must specify the origin and location of at least one of a model's body coordinate systems relative to the World CS.

**Connector Ports.** Any Body CS can display a Body CS Port. A Body CS Port allows you to attach Joints, Actuators, and Sensors to a Body. By default, a Body's CS1 and CS2 coordinate systems each display a Body CS port. You can display a port for any other Body coordinate system as well, including a Body's CG CS.

### Creating a Body Block

To create a Body block:

- 1 Drag a Body block icon from the SimMechanics Bodies library and drop it into your model window.

- 2 Open the Body block's dialog box.
- 3 Enter the mass of the body you are modeling in the **Mass** field.
- 4 Select the units of mass from the adjacent units list.
- 5 Enter a 3-by-3 matrix representing the body's inertia tensor relative to its center of gravity coordinate system (CG CS) origin and axes in the **Inertia** field (see "Determining Inertia Tensors for Common Shapes" on page 4-11).
- 6 Enter the initial positions of the body's CG and coordinate systems in the **Position** pane.
- 7 Enter the initial orientation of the body's CG and coordinate systems in the **Orientation** pane.
- 8 Click **OK** or **Apply**.

### Determining Inertia Tensors for Common Shapes

The following table enables you to determine the inertia tensors for some common shapes. For each shape of mass  $m$ , the table lists the shape's principal moments of inertia,  $I_1$ ,  $I_2$ , and  $I_3$ , along the  $x$ -,  $y$ -, and  $z$ -axes of the shape's CG coordinate system.

Shape	$I_1$	$I_2$	$I_3$
Thin rod of length $L$ aligned along $z$	$\frac{1}{12}mL^2$	$\frac{1}{12}mL^2$	0
Sphere of radius $R$	$\frac{2}{5}mR^2$	$\frac{2}{5}mR^2$	$\frac{2}{5}mR^2$
Cylinder of radius $R$ and height $h$ aligned along $z$	$\frac{1}{4}m(R^2 + \frac{1}{3}h^2)$	$\frac{1}{4}m(R^2 + \frac{1}{3}h^2)$	$\frac{1}{2}mR^2$
Rectangular parallelepiped of sides $a$ , $b$ , and $c$ aligned along $x$ , $y$ , $z$ , respectively	$\frac{1}{12}m(b^2 + c^2)$	$\frac{1}{12}m(a^2 + c^2)$	$\frac{1}{12}m(a^2 + b^2)$

Shape	$I_1$	$I_2$	$I_3$
Cone of base radius $R$ and height $h$ along $z$	$\frac{1}{4}m(\frac{3}{5}R^2 + h^2)$	$\frac{1}{4}m(\frac{3}{5}R^2 + h^2)$	$\frac{3}{10}mR^2$
Ellipsoid of semiaxes $a$ , $b$ , and $c$ aligned along $x$ , $y$ , $z$ , respectively	$\frac{1}{5}m(b^2 + c^2)$	$\frac{1}{5}m(a^2 + c^2)$	$\frac{1}{5}m(a^2 + b^2)$

The corresponding inertia tensor for the shape is the following 3-by-3 matrix:

$$\begin{bmatrix} I_1 & 0 & 0 \\ 0 & I_2 & 0 \\ 0 & 0 & I_3 \end{bmatrix}$$

## Working with Body Coordinate Systems

Every body in SimMechanics has body coordinate systems (CSs) attached to it. The location of a body CS is the origin of that CS. The CS's rectangular  $x$ - $y$ - $z$  coordinate axes are rotated at some orientation. You set up body CS origins and orientations before running your model. But once the bodies start to move, the origins and orientations of a body's CSs remain fixed in the body. The elements of a body's inertia tensor also remain fixed in the body.

The sections "Managing Body Coordinate Systems" on page 4-15 and "Creating Body CS Ports" on page 4-16 explain how to create custom body coordinate systems and Body CS ports or delete existing ports.

### Setting a Body CS's Position

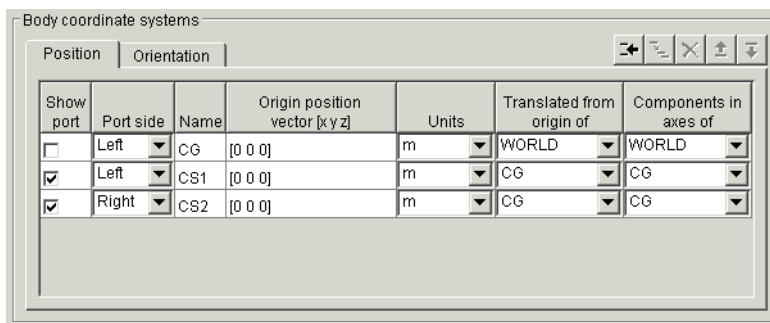
The **Position** pane of a Body block's dialog box allows you to specify the position of any of a body's local coordinate systems.

The **Translated from origin of** and **Components in axes of** lists in the pane together specify which other of your model's coordinate systems you use as reference points and orientations to set up the coordinate systems of the body you are configuring.

To specify the position of a Body CS:

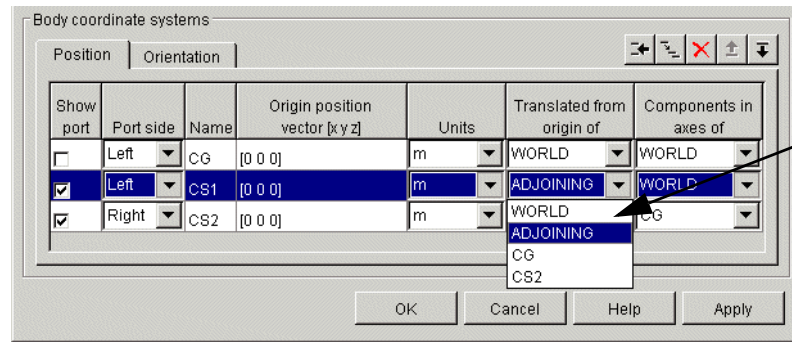
- 1 Open the Body block's dialog box.

The dialog box's **Position** pane lists the body's local coordinate systems in a table.



Each row specifies the position of the coordinate system specified in the **Name** column.

- 2 Select the units in which you want to specify the origin of the Body CS from the CS's **Units** list.
- 3 Specify the reference coordinate systems for the Body CS, i.e., the coordinate system relative to which you want to measure the Body CS origin and the orientation of the Body CS's coordinate axes. The choices are World, the adjoining CS, and other Body CSs on the same Body.



You do this by selecting the origin and orientation of the specification CS from the Body CS's **Translated from origin of** and **Components in axes of** lists, respectively. For example, suppose that you want to specify the position of CS2 relative to another coordinate system, whose origin is at the origin of CS1 but whose axes run parallel to those of the World CS. Then you would select CS1 from the **Translated from origin of** list of CS2 and WORLD from the **Components in axes of** list of CS2.

- 4 Enter a vector specifying the location of the Body CS in the **Origin position vector [x y z]** field of the CS.

The components of the vector must be in the units that you selected and relative to the coordinate system that you selected. For example, suppose that you had selected *m* as the units for specifying CS2's origin and CS1 and WORLD as the origin and orientation of the position specification CS for CS2. Now suppose that you want to specify the location of CS2 as 1 meter to the right of CS1 along the World *x*-axis. Then you would enter [1 0 0] as CS2's position vector.

- 5 Click **Apply** to accept the position setting or **OK** to accept the setting and dismiss the dialog box.

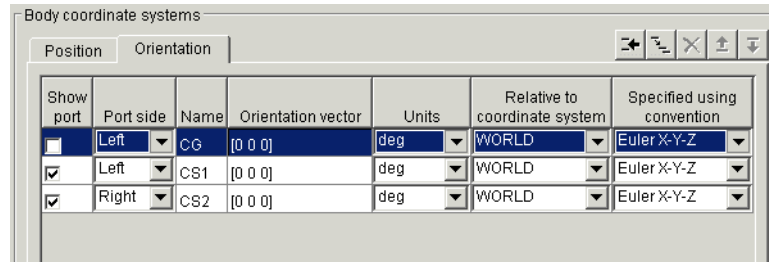
### Setting a Body CS's Orientation

The **Orientation** pane of a Body block's dialog box allows you to specify the orientation of any of a body's local coordinate systems.

To specify the orientation of a Body CS:



- 1 Open the Body block's dialog box.
- 2 Select the dialog box's **Orientation** pane.

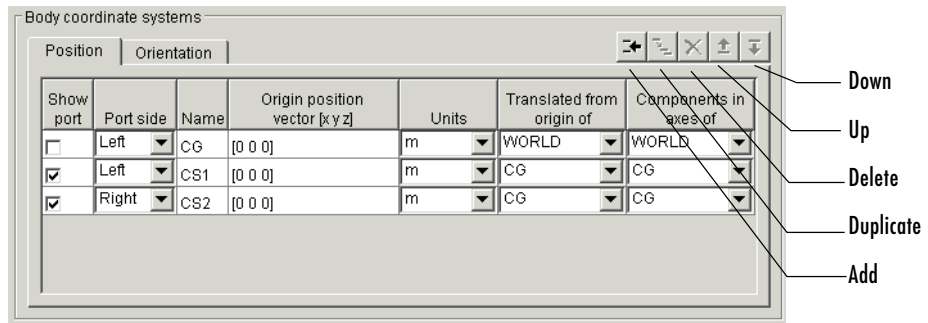


- 3 Select the units (degrees or radians) in which you want to specify the orientation of the CS from the CS's Units pane.
- 4 Select the coordinate system relative to which you want to specify the orientation of the Body CS from the Body CS's **Relative to coordinate system** list. The choices are World, the adjoining CS, and other Body CSs on the same Body.
- 5 Select the convention you want to use to specify the orientation of the Body CS from the CS's **Specified using convention** list.
- 6 Enter a vector that specifies the orientation of the Body CS relative to the selected specification CS, according to the selected specification convention.
- 7 Click **Apply** to accept the position setting or **OK** to accept the setting and dismiss the dialog box.

### Managing Body Coordinate Systems

You will often need to modify the default body coordinate systems of a Body block. You might want to connect a Body to more than two Joints, in which case you need not only more Body CSs, but their Body CS ports as well. Connecting Actuators and Sensors to Bodies requires a Body CS and Body CS port for each connection.

The **Body coordinate systems** panel of a Body block's dialog box contains a row of buttons that allow you to add, delete, duplicate, and reorder a Body's local coordinate systems.



To use these buttons, select a Body CS in the CS table and select

- **Duplicate** to duplicate the CS
- **Delete** to remove the selected CS from the table
- **Up** to move the CS's entry one row up in the CS table
- **Down** to move the CS's entry one row down in the CS table

Select **Add** to add a new CS.

### Creating Body CS Ports

To add or delete a port from a Body block's icon, open the block's dialog box and select or unselect the CS's **Show port** check box in the dialog box's Body CS table. Click **OK** or **Apply** to confirm the change.

## Modeling Joints

In SimMechanics, a joint represents the degrees of freedom (DoF) that one body (the follower) has relative to another body (the base). The base body can be a finite rigid body or a ground. Unlike a physical joint, a SimMechanics joint has no mass, although some joints have spatial extension (see “Modeling with Massless Connectors” on page 4-26).

A SimMechanics joint does not necessarily imply a physical connection between two bodies. For example, a SimMechanics Six-DoF joint allows the follower, e.g., an airplane, unconstrained movement relative to the base, e.g., ground, and does not require that the follower ever come into contact with the base.

SimMechanics joints only add degrees of freedom to a machine, because the Body blocks carry no degrees of freedom. Contrast this with physical joints, which both add DoFs (with axes of motion) and remove DoFs (by connecting bodies). See “Counting Degrees of Freedom” on page 4-67.

SimMechanics provides an extensive Joints library with blocks for modeling various types of joints. This section explains how to use these blocks.

### About Joints

Modeling with Joint blocks requires an understanding of the following key concepts:

- Joint primitives
- Joint types
- Joint axes
- Joint directionality
- Assembly restrictions

### Joint Primitives

Each Joint block conceptually represents one or more joint primitives that together specify the degrees of freedom that a follower body has relative to the

base body. The following table summarizes the joint primitives found singly or multiply in Joint blocks.

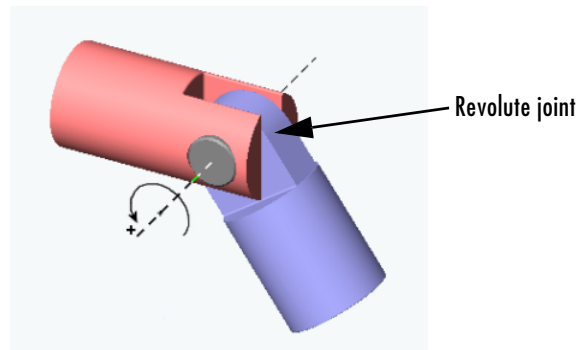
Primitive Type	Symbol	Degrees of Freedom
Prismatic	P	One degree of translational freedom along a prismatic axis
Revolute	R	One degree of rotational freedom about a revolute axis
Spherical	S	Three degrees of rotational freedom about a pivot point
Weld	W	Zero degrees of freedom

## Joint Types

The blocks in the SimMechanics Joints library fall into the following categories:

- Primitive joints

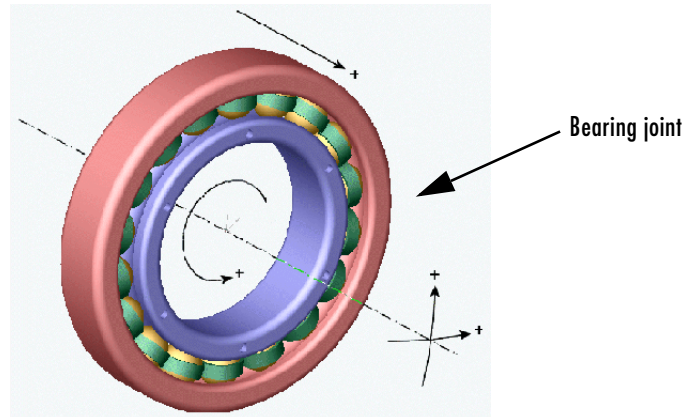
Each of these blocks contains a single joint primitive. For example, the Revolute block contains a revolute joint primitive.



- Composite joints

These blocks contain combinations of joint primitives, enabling you to specify multiple rotational and translational degrees of freedom of one body relative

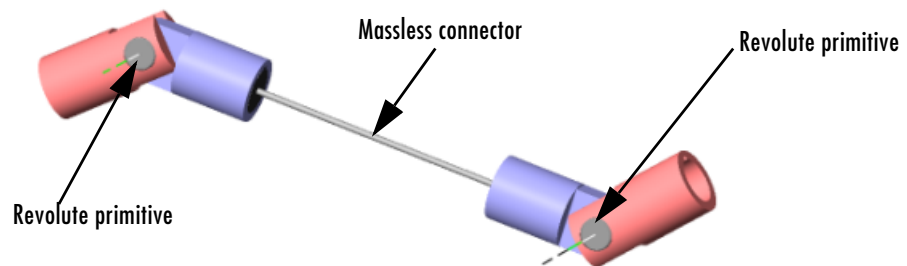
to another. Some model idealized real joints, for example, the Gimbal and Bearing joints.



Others specify abstract combinations of degrees of freedom. For example, the Six-DoF block specifies unlimited motion of the follower relative to the base. The Custom Joint allows you to create joints with any desired combination of rotational and translational degrees of freedom, in any order. The prefabricated composite Joints of the Joints library have the type and order of their primitives fixed. See “Axis Order” following.

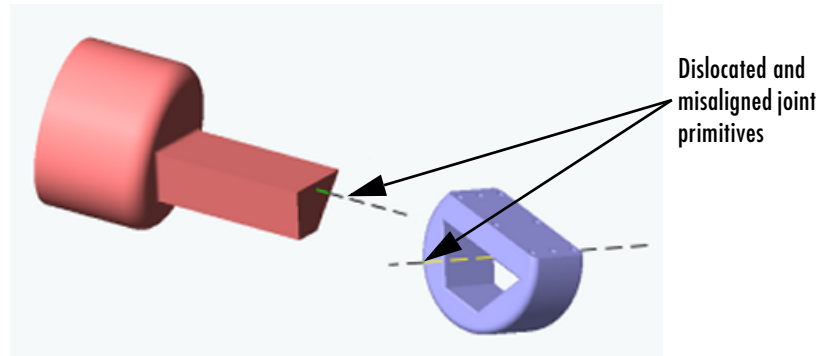
- Massless connectors

These blocks represent extended joints with spatially separated joint primitive axes, for example, a Revolute-Revolute Massless Connector.



- **Disassembled joints**

These blocks represent joints that SimMechanics assembles at simulation time, for example, a Disassembled Prismatic.



See “Assembly Restrictions” on page 4-23 and “Modeling with Disassembled Joints” on page 4-29.

### Joint Axes

Joint blocks define one or more axes of translation or rotation along which or around which a follower block can move in relation to the base block. The axes of a Joint block are the axes defined by its component primitives:

- A prismatic primitive defines an axis of translation.
- A revolute primitive defines an axis of revolution.
- A spherical primitive defines a pivot point for axis-angle rotation.

For example, a Planar Joint block combines two prismatic axes and hence defines two axes of translation.

**Axis Direction.** By default the axes of prismatic and revolute primitives point in the same direction as the  $z$ -axis of the World coordinate system. A Joint block’s dialog box allows you to point its prismatic and revolute axes in any other direction (see “Directing Joint Axes” on page 4-24).

**Axis Order.** SimMechanics executes the motion of composite joints one joint primitive at a time. A joint that defines more than one axis of motion also defines the order in which the follower body moves along each axis or about a

pivot. The order in which the axes and/or pivot appear in the Joint block's dialog box is the order in which the follower body moves.

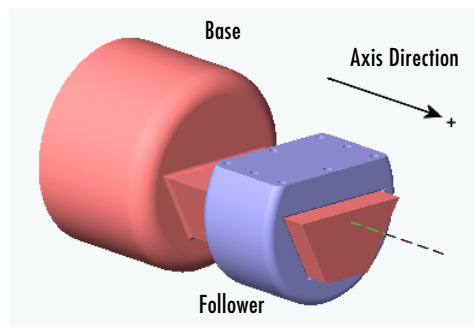
Different primitive execution orders are physically equivalent, unless the joint includes one spherical or three revolute primitives. Pure translations and pure two-dimensional rotations are independent of primitive ordering.

### Joint Directionality

Directionality is a property of a joint that determines the dependence of the joint on the sign of forces or torques applied to it. A joint's directionality also determines the sign of signals output by sensors attached to the joint. SimMechanics assigns a directionality to every joint in your model. You must be able to determine the directionality of a joint in order to actuate it correctly and to interpret the output of sensors attached to it.

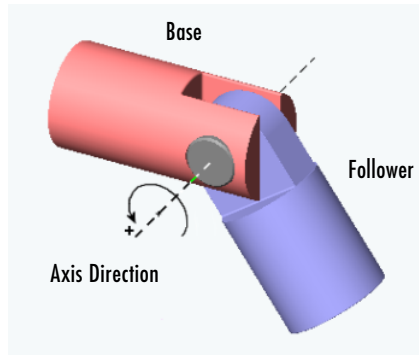
When assigning directionality to a joint, SimMechanics regards the joint's follower as moving relative to the joint's base. SimMechanics then assigns a directionality to the joint, taking into account the type of joint and the direction of the joint's axis, as follows.

**Directionality of a Prismatic Joint.** If the joint is prismatic, a positive force applied to the joint moves the follower in the positive direction along the axis of translation. A sensor attached to the joint outputs a positive signal if the follower moves in a positive direction along the joint's axis of translation relative to the base.



**Directionality of a Revolute Joint.** If the joint is revolute, a positive torque applied to the joint rotates the follower by a positive angle around the joint's axis of rotation, as determined by the right-hand rule. A sensor attached to the

revolute joint outputs a positive signal if the follower rotates by a positive angle around the joint's axis of revolution, as determined by the right-hand rule.



**Directionality of a Spherical Joint.** Spherical joint directionality means the positive sense of rotation of the three rotational DoFs. Pick a rotation axis, rotating using the right-hand rule from the base Body CS axes. Then rotate the follower Body about that axis in the right-handed sense.

**Directionality of Composite Joints.** SimMechanics assigns a directionality separately to each joint primitive, based on the primitive's type and the direction of its axis of translation or rotation. In each case, SimMechanics regards the follower body of the composite joint as moving relative to the base body along or around the joint primitive's axis.

The order of primitives in the composite Joint's dialog determines the spatial construction of the joint. The first listed primitive is attached to the base, the second to the first, and so on, down to the follower, which is attached to the last primitive. Moving the first listed primitive moves the subsequent primitives in the list, as well as the follower, relative to the base. Moving any primitive moves the primitives below it in the list (but not those above it), as well as the follower. Moving the last listed primitive moves only the follower.

**Changing the Directionality of a Joint.** You can change the directionality of a joint by rewiring the Joint block to reverse the roles of the base and follower bodies or by reversing the sign (direction) of the joint axis.



## Assembly Restrictions

Many joints impose one or more restrictions, called *assembly restrictions*, on the positions of the bodies that they join. The conjoined bodies must satisfy these restrictions at the beginning of simulation and thereafter within assembly tolerances that you can specify (see “Setting Assembly Tolerances” on page 5-5). For example, the attachment points of revolute and spherical joints must coincide within assembly tolerances; the attachment points of a Prismatic joint must be collinear with the prismatic axis within assembly tolerances; the attachment points of a Planar joint must be coplanar, etc. Composite joints, e.g., the Six-DoF joint, impose assembly restrictions equal to the most restrictive of its joint primitives. See the block reference for each Joint for information on the assembly restrictions, if any, that it imposes. Positioning bodies so that they satisfy a joint’s assembly restrictions is called *assembling* the joint.

All joints except joints in the SimMechanics Disassembled Joints sublibrary require manual assembly. Manual assembly entails your setting the initial positions of conjoined bodies to valid locations (see “Assembling Joints” on page 4-25). SimMechanics assembles disassembled joints during the model initialization phase of simulation. It assumes that you have already assembled all other joints before the start of simulation. Hence joints that require manual assembly are called *assembled* joints. During model initialization and at each time step, SimMechanics also checks to ensure that your model’s bodies satisfy all assembly restrictions. If any of your model bodies fails to satisfy assembly restrictions, Simulink halts the simulation and displays an error message.

## Creating a Joint

A joint must connect exactly two bodies. To create a joint between two bodies:

- 1 Select the Joint from the SimMechanics Joints library that best represents the degrees of freedom of the follower body relative to the base body.
- 2 Connect the base connector port of the Joint block (labeled B) to the point on the base block that serves as the point of reference for specifying the degrees of freedom of the follower block.
- 3 Connect the follower connector port of the Joint block (labeled F) to the point on the follower block that serves as the point of reference for specifying the degrees of freedom of the base block.

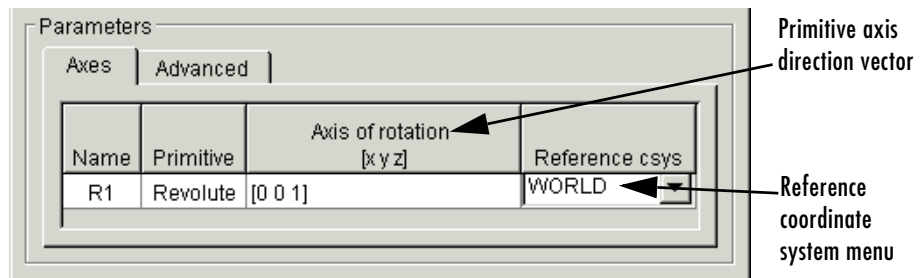
- 4 Specify the directions of the joint's axes (see "Directing Joint Axes" on page 4-24).
- 5 If you plan to attach Sensors or Actuators to the Joint, create an additional port for each Sensor and Actuator (see "Creating Actuator and Sensor Ports on a Joint" on page 4-24).
- 6 If the joint is an assembled joint, assemble the bodies joined by the joint (see "Assembling Joints" on page 4-25).

## Directing Joint Axes

By default the prismatic and revolute axes of a joint point in the same direction as the  $z$ -axis of the World coordinate system. To change the direction of the axis of a joint primitive:

- 1 Open the joint's dialog box and select a reference coordinate system for specifying the axis direction from the coordinate system list associated with the axis primitive.

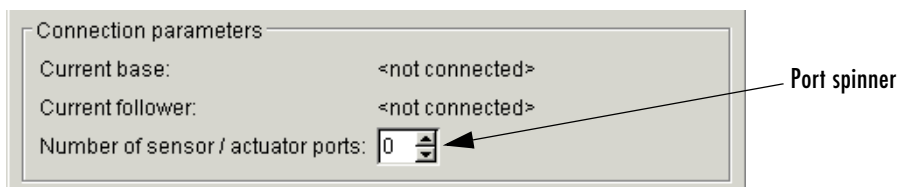
The options are the World coordinate system or the local coordinate systems of the base or follower attachment point. Choose the coordinate system that is most convenient.



- 2 Enter in the primitive's axis direction field a vector that points in the desired direction of the axis in the selected coordinate system.

## Creating Actuator and Sensor Ports on a Joint

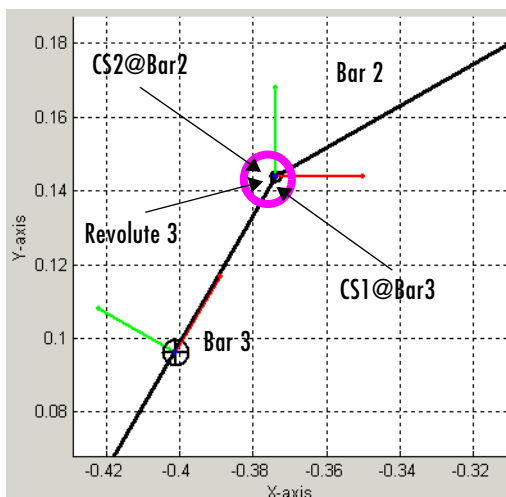
To create additional connector ports on a Joint for Actuators and Sensors, open the Joint's dialog box and set the **Number of sensor/actuator ports** to the number of Actuators and Sensors you plan to attach to the Joint.



Apply the setting by clicking **OK** or **Apply**.

### Assembling Joints

You must manually assemble all assembled joints in your model. Assembling a joint requires setting the initial positions of its base and follower attachment points such that they satisfy the assembly restrictions imposed by the joint (see “Assembly Restrictions” on page 4-23). Consider, for example, the model discussed in “A Four Bar Mechanism” on page 2-36. This model comprises three bars connected by revolute joints to each other and to two ground points. The model collocates the CS origins of the Body CS ports connected to each Joint, thereby satisfying the assembly restrictions imposed by the revolute joints.

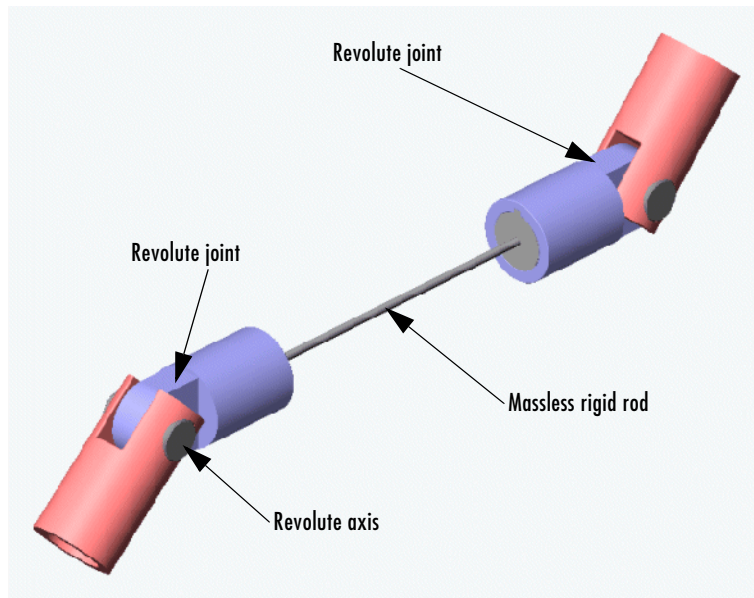


**Assembled Revolute Joint in the Four Bar Mechanism**

### Modeling with Massless Connectors

Massless connectors simplify the modeling of machines that use a relatively light body to connect two relatively massive bodies. For example, you could use a Body block to model such a connector. But the resulting equations of motion might be ill-conditioned, because that connecting body's mass is small, and the simulation can be slow or error prone. A massless connector also avoids global inconsistencies that can arise if you use a Constraint block to model the connector.

A massless connector consists of a pair of joints located a fixed distance apart. Think of a massless connector as a massless rod with a joint primitive affixed at each end.



The initial orientation and length of the massless connector are defined by a vector drawn from the base attachment point to the follower attachment point. During simulation, the orientation of the massless connector can change but not its length. In other words, the massless connector preserves the initial separation of the base and follower bodies during machine motion.

---

**Note** You cannot actuate or sense a massless connector.

---

The SimMechanics Joints/Massless Connectors sublibrary contains three Massless Connectors:

- One with two revolute primitives (Revolute-Revolute)
- One with a revolute primitive and a spherical primitive (Revolute-Spherical)
- One with two spherical primitives (Spherical-Spherical)

You can set the direction of the axes of the revolute primitives.

### Creating a Massless Connector

To create a massless connector between two bodies:

- 1** Drag an instance of a Massless Connector block from the Massless Connectors sublibrary into your model and connect it to the base and follower blocks.

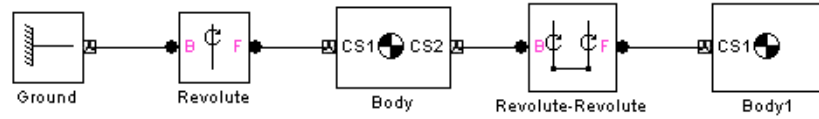
If necessary, point the axes of the connector's revolute joints in the direction required by the dynamics of the machine you are modeling.

- 2** Assemble the connector by setting the initial positions of the base and follower body attachment points to the initial positions required by your machine's structure.

During simulation, the massless connector maintains the initial separation between the bodies though not necessarily the initial relative orientation.

### Massless Connector Example: Triple Pendulum

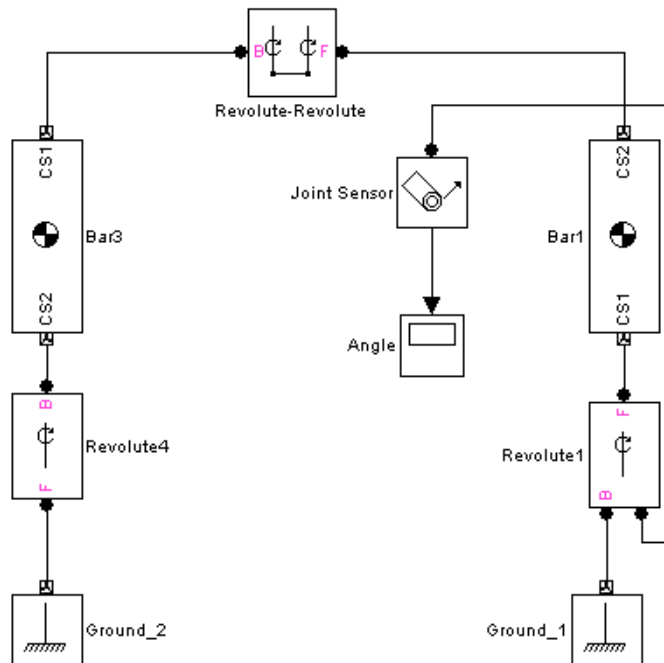
Consider a triple pendulum comprising massive upper and lower bodies and a middle body of negligible mass. The following model uses a Revolute-Revolute massless connector to model such a pendulum.



In this model, the joint axes of the Revolute-Revolute connector have their default orientation along the World  $z$ -axis. As a result, the lower arm (Body1) rotates parallel to the World's  $x$ - $y$  plane.

### Massless Connector Example: Four Bar Mechanism

The following model replaces one of the bars (Bar2) in the mech\_four\_bar model from the Demos library with a Revolute-Revolute massless connector.

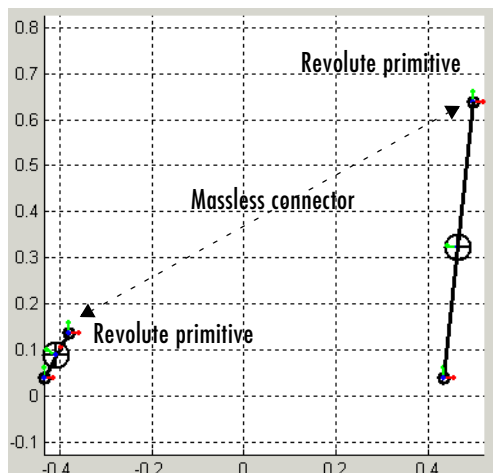


This model changes the Body CS origins of Bar3 to the following values.

Name	Origin position vector	Translated from origin of
CG	[ -0.027 0.048 0 ]	CS1
CS1	[ 0.054 0.096 0 ]	CS2
CS2	[ 0 0 0 ]	ADJOINING (Ground_2)

This creates a separation between Bar3 and Bar1 equal to the length of Bar2 in the original model.

Try simulating both the original and the modified model. Notice that the massless connector version moves differently, because you eliminated the mass of Bar2 from the model. Notice also that the massless bar does not appear in the animation of the massless connector version of the model.



## Modeling with Disassembled Joints

The SimMechanics Joints/Disassembled Joints sublibrary contains a set of joints that SimMechanics automatically assembles at the start of simulation; that is, SimMechanics positions the joints such that they satisfy the assembly restrictions imposed by the type of joint, e.g., prismatic or revolute. Using these joints eliminates the need for you to assemble the joints yourself.

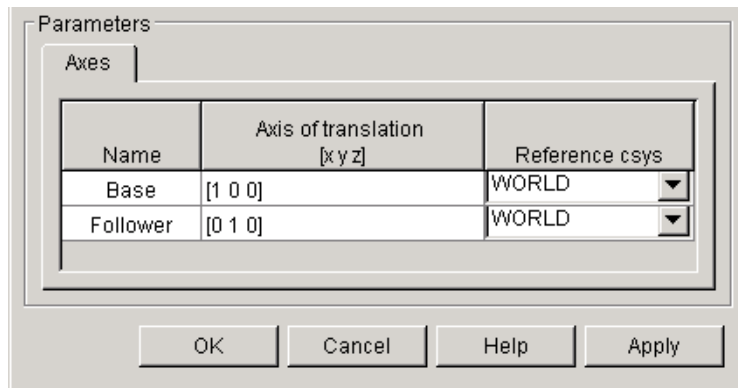
Disassembled joints differ from assembled joints in significant ways. An assembled joint primitive has only one axis of translation or revolution or one spherical pivot point. A disassembled prismatic or revolute primitive has two axes of translation or rotation, one for the base and one for the follower body. A disassembled spherical primitive similarly has two pivot points.

---

**Note** Disassembled joints can appear only in closed loops and each closed loop can contain only one disassembled joint.

---

The dialog box for a disassembled joint allows you to specify the direction of each axis.



During model assembly, SimMechanics determines a common axis of revolution or translation that satisfies model assembly restrictions, and aligns the base and follower axes along the common axis.

### Controlling Automatic Assembly

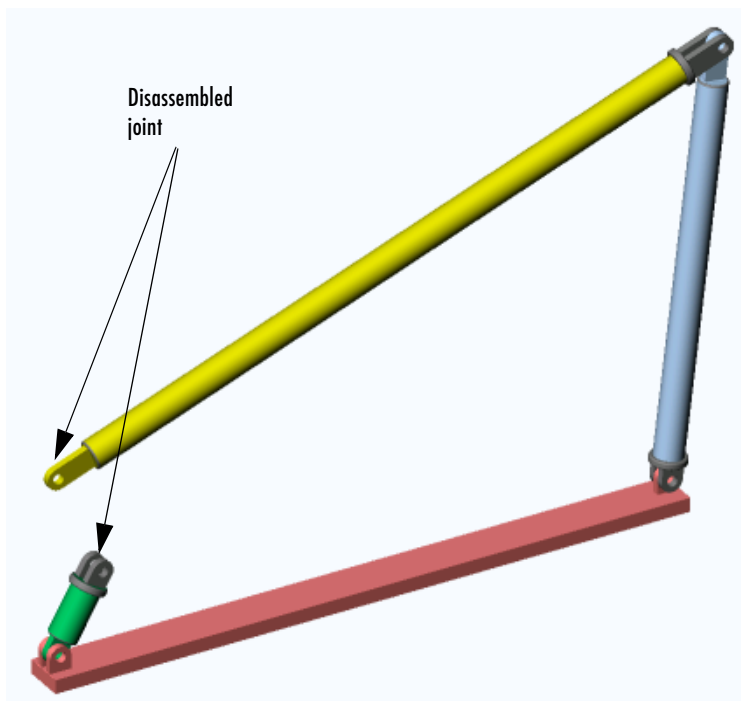
During model assembly, SimMechanics might move bodies connected by assembled joints from their initial positions in order to assemble the disassembled joints. SimMechanics' solution to the assembly problem cannot be predicted beforehand, except in simple cases. To prevent SimMechanics from moving bodies during model assembly, use Joint Initial Condition Actuator (JICA) blocks to specify the initial positions of bodies whose positions you want to remain fixed during the assembly process. This forces



SimMechanics to find assembly solutions that satisfy the initial conditions specified by the JICA blocks.

### Disassembled Joint Example: Four Bar Mechanism

This example creates and runs a model of a disassembled four bar machine.



Refer to the tutorial “A Four Bar Mechanism” on page 2-36, and the mech\_four\_bar demo:

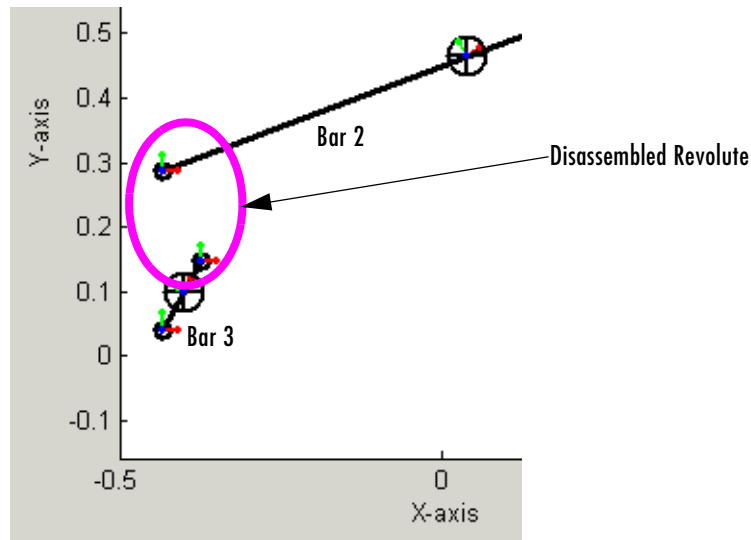
- 1 Disconnect the Joint Sensor1 block from the Revolute3 block.
- 2 Replace Revolute3 with a Disassembled Revolute block from the Joints/Disassembled Joints sublibrary.
- 3 Open the Disassembled Revolute dialog box and, under **Axis of rotation** for both Base and Follower axes, enter  $[0 \ 0 \ 1]$ . Close the dialog.

- 4 Open the Bar2 dialog box and dislocate the joint by displacing Bar2's CS2 origin from Bar 3's CS1 origin.

Do this by entering a nonzero vector under **Origin position vector [x y z]** for CS2, then changing the **Translated from origin of** pulldown entry to ADJOINING. CS1 on Bar3 is the Adjoining CS of CS2 of Bar2. Close the dialog.

- 5 To avoid circular CS referencing, you must check the Bar3 dialog entry for CS1 on Bar3. Be sure that CS1 on Bar3 does *not* reference CS2 on Bar2. Reference it instead to CS2 on Bar3, which adjoins Ground\_2.
- 6 Rerun the model.

Note that the motion is different from the manually assembled case.



### Cutting Closed Loops

Simulink cannot solve models whose block schematics contain closed loops. To simulate a model containing closed loops, SimMechanics internally converts a closed-loop model to an open-loop model, by cutting each of the model's closed loops once, at a Joint, Constraint, or Driver block.

You can specify the Joint to cut if the loop does not contain a disassembled joint. To do this, open the Joint's dialog box and select the **Mark as the preferred cut joint option** check box on the **Advanced** pane in that Joint's dialog **Parameters** area. If you do not specify a preference, Simulink cuts the loop at a Disassembled Joint block if the loop contains one; otherwise, at a Constraint or Driver block if the loop contains a Constraint block; otherwise, at the Joint that has the fewest degrees of freedom.

---

**Note** SimMechanics cuts a loop at a disassembled joint regardless of your preferred cut.

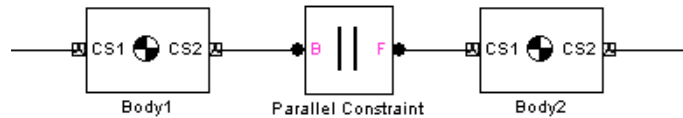
---

To display automatically cut joints, select the **Show automatically cut joints** check box on the **Constraints** pane of the **Mechanical Environment Settings** dialog box (see “Constraints Pane” on page 5-9).

Refer to “Modeling with Disassembled Joints” on page 4-29 for more on disassembled joints. Consult “Checking Schematic Topology” on page 4-64 to see how SimMechanics analyzes closed loops in the model schematic.

## Modeling Constraints and Drivers

The SimMechanics Constraints & Driver library provides a set of blocks to model constraints on the relative motions of two bodies. You model the constraint by connecting the appropriate Constraint or Driver block between the two bodies. As with joints, the blocks each have a base and follower connector port, with the body connected to the follower port viewed as moving relative to the body connected to the base port. For example, the following model constrains Body2 to move along a track that is parallel to the track of Body1.



The blocks enable you to model time-independent constraints or time-dependent drivers.

Constraint and unactuated Driver blocks model *scleronomic* (time-independent) constraints.

Actuated Driver blocks (see “Actuating a Driver” on page 4-49) model *rheonomic* (time-dependent) constraints.

- Actuated Driver blocks (see “Actuating a Driver” on page 4-49) model *rheonomic* (time-dependent) constraints.

*Holonomic* constraint functions depend only on body positions, not velocities:

$$f(\mathbf{x}_B, \mathbf{x}_F; t) = 0$$

Constraints of the form

$$g(\mathbf{x}_B, \dot{\mathbf{x}}_B, \mathbf{x}_F, \dot{\mathbf{x}}_F; t) = 0$$

can sometimes be integrated into a form dependent only on positions; but if not, they are *nonholonomic*.

See the reference pages for information on the specific constraint that a Constraint or Driver block imposes.

Constraints and drivers can only remove degrees of freedom from a machine. Constraints and unactuated Drivers prevent the machine from moving in certain ways. Actuated Drivers externally impose a relative motion between pairs of bodies, and those pairs of bodies are no longer free to respond to externally applied forces or torques. See “Counting Degrees of Freedom” on page 4-67.

## Directionality of Constraints and Drivers

Like joints, constraints and drivers have directionality. The sequence of base to follower body determines the directionality of the constraint or driver. The directionality determines how the sign of Driver Actuator signals affects the motion of the follower relative to the base and the sign of signals output by constraint and driver sensors.

## Solving Constraints

When simulating a model, SimMechanics uses a constraint solver to find the motion, given the model’s Constraint and Driver blocks. You can specify both the constraint solver type and the constraint tolerances that SimMechanics uses to find the constraint solution. See “Choosing a Constraint Solver” on page 5-5 for more information.

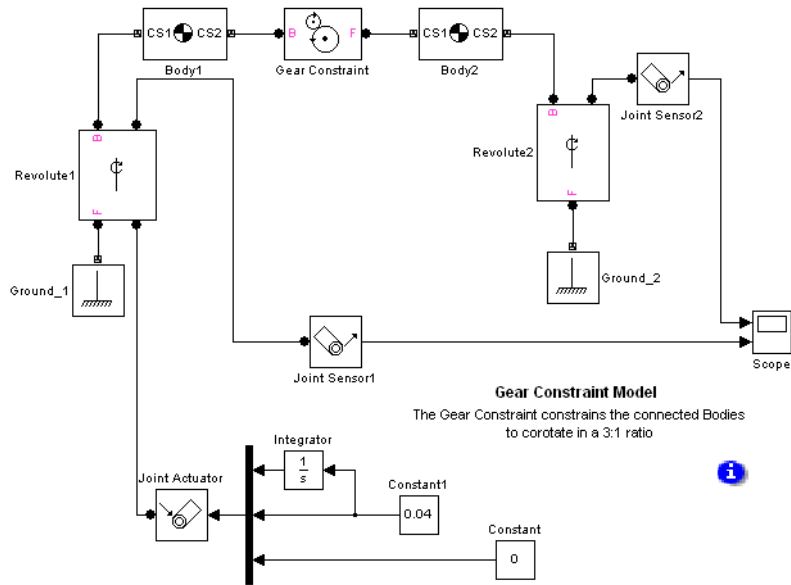
## Restrictions on Using Constraint and Driver Blocks

The following restrictions apply to the use of Constraint and Driver blocks in a model:

- Constraint and Driver blocks can appear only in closed loops.
- A closed loop cannot contain more than one Constraint or Driver block.
- A Constraint or Driver must connect exactly two Bodies.

## Constraint Example: Gear Constraint

The `mech_gears` model illustrates the Gear Constraint. Open the Body and Gear Constraint blocks.



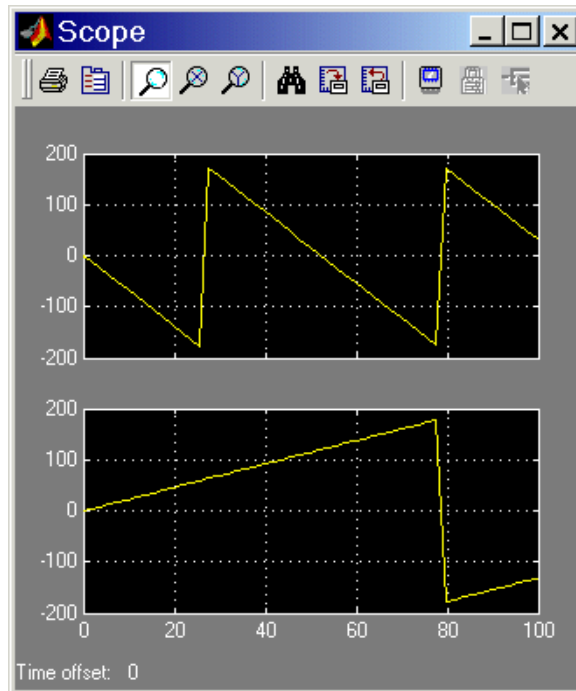
Body1 and Body2 have their CG positions 2 meters apart. CS1 and CS2 on Body1 are collocated with the Body1 CG, and similarly, CS1 and CS2 on Body2 are collocated with the Body2 CG.

The Gear Constraint between them has two pitch circles. One is centered on the CS2 at the base Body, which is Body1, and has radius 1.5 meters. The other is centered on CS1 at the follower Body, which is Body2, and has radius 0.5 meters. The distance between CS2 on Body1 and CS1 on Body2 is 2 meters. The sum of the pitch circle radii equals this distance, as it must.

### Visualizing the Gear Motion

The model is set up to open the visualization feature automatically upon simulation start, with Handle Graphics and convex hulls, as explained in “Viewing Machines with Handle Graphics” on page 6-10. Start the simulation and watch the CG CS axis triads spin around. The CG triad at Body2 rotates three times faster than the CG triad at Body1, because the pitch circle centered on Body2 is three times smaller.

You can see the same behavior in the Scope. The upper plot shows the motion of Revolute2, and the lower plot the motion of Revolute1. Note that angular motion is mapped to the interval  $(-180^\circ, +180^\circ]$  degrees.



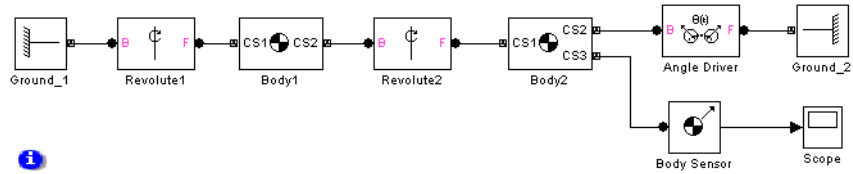
The Gear Constraint is inside a closed loop formed by

Ground\_1–Revolute1–Body1–Gear Constraint–Body2–Revolute2–  
Ground\_2

Although Ground\_1 and Ground\_2 are distinct blocks, they represent different points on the same immobile ground at rest in World. So the blocks form a loop.

### Driver Example: Using the Angle Driver

The following two models illustrate the Angle Driver, both with and without a Driver Actuator. The first is mech\_angle\_unact. Open the Body2 block.



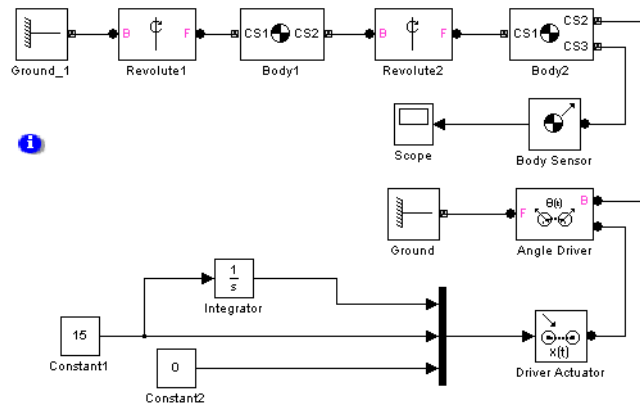
The bodies form a double pendulum of two rods. The Body Sensor is connected to Body2 at CS3 = CS2 and measures all three components of Body2's angular velocity vector with respect to the ground.

The Angle Driver is connected between Body2 and Ground\_2. Because the Angle Driver is not actuated in this model, it acts during the simulation as a time-independent constraint to hold the angle between Body2 and Ground\_2 constant at its initial value.

### Visualizing the Angle Driver Motion

The model is set up to open the visualization feature automatically upon simulation start, with Handle Graphics and convex hulls, as explained in "Viewing Machines with Handle Graphics" on page 6-10. Start the simulation. The upper body swings like a pendulum, but the lower body maintains its horizontal orientation with respect to the horizontal ground. The Scope measures Body2's angular velocity with respect to ground, and this remains at zero.

The second model is mech\_angle\_act. Open the Driver Actuator block.





The Driver Actuator drives the Angle Driver block. The Actuator accepts a constant angular velocity signal from the Simulink blocks. The Actuator also requires the angle itself and the angular acceleration, together with the angular velocity, in a vector signal format.

The Body Sensor again measures three components of Body2's angular velocity with respect to the ground. Constant1 drives the angle at 15°/second. While the simulation is running, this angle changes at the constant rate. But at the same time, the assembly and the constant length of the two pendulum rods must be maintained by Simulink, while both rods are subject to gravity. These requirements slow the simulation down, as they require increasing computing power to maintain all of them simultaneously.

As in the Gear Constraint model, the two Ground blocks in these models represent points on the same immobile ground at rest in World, so the Angle Driver is part of a closed loop.

# Modeling Actuators

The SimMechanics Sensors & Actuators library provides a set of Actuator blocks that enable you to apply time-dependent forces and motions to bodies, joints, and drivers. You can also vary a body's mass and inertia tensor.

---

**Note** SimMechanics allows you to connect an Actuator to a Ground. But it displays an error if you attempt to simulate or update a model containing such a connection. This is because ground is immobile and hence cannot be actuated.

---

You can use Actuator blocks to perform the following tasks:

- Apply a time-varying force or torque to a body or joint
- Specify the position, velocity, and acceleration of a joint or driver as a function of time
- Specify the initial position and velocity of a joint primitive

In general, actuators can apply any combination of forces and motions to a machine provided that

- The applied forces and motions are consistent with each other and with the machine's geometry, constraints, and assembly restrictions.
- It is possible to find a unique solution for the motion of each actuated degree of freedom (DoF).

## Actuating a Body

You can use body actuators to apply forces and/or torques, but not motions, to bodies. (You can apply motions to a body indirectly, using Joint Actuators. See "Applying Motions to Bodies" on page 4-42.)

To actuate a body:

- 1 If there is not already an unused connector port available for the Actuator, create a Body CS port on the Body for the Actuator. See the Body block reference if you need to learn how.

- 2 Drag a Body Actuator block from the Sensors & Actuators library into your model and connect its output port to a Body CS port on the Body.
- 3 Open the Actuator's dialog box.
- 4 Choose to apply a force or torque to the body:
  - Select the **Apply force** check box if you want to apply a force to the body, and select the units of force from the adjacent list.
  - Select the **Apply torque** check box if you want to apply a torque to the body, and select the units of torque from the adjacent list.
- 5 Select the coordinate system used to specify the applied torque from the **Using reference coordinate system** list.

The list allows you to choose either the World CS or the Body CS of the port to which you attached the Actuator.

- 6 Create vector signals that specify the value of the applied torque and force at each time step.

You can use any Simulink source block (for example, an Input port or a Sine Wave block) or combination of Simulink blocks to create the Body Actuator signal. You can also use the output of a Sensor block connected to the Body as input to the Actuator, thereby creating a feedback loop. Such loops are useful for modeling springs and dampers (see “Checking Model Validity” on page 4-64.)

- 7 Connect the force and/or torque signal to the input port of the Actuator.

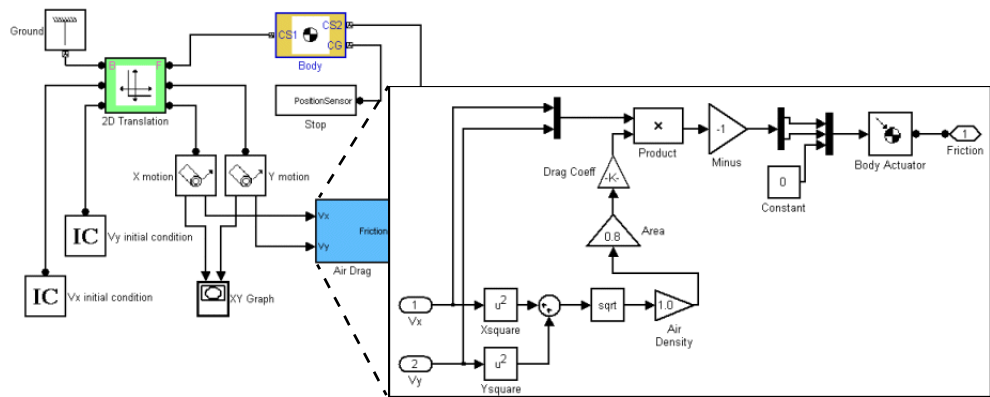
If you are applying both a force and a torque to the body, connect the force and torque signals to the inputs of a two-input Mux block. Then connect the output of the Mux block to the input of the Actuator.

### **Body Actuator Example: Pure Kinetic Friction**

The `mech_ballistic_kin_fric` model in the Demos library provides an example of how to implement pure kinetic friction. This type of friction is a continuous force that depends on a body's motion relative to a medium (such as air), as well as on physical characteristics of the body. Kinetic friction, unlike “stiction,” involves no “sticking” or locking of motion, and the friction is not

discontinuous. While you could use the Joint Stiction Actuator, this is not necessary. This model applies air friction or drag to a projectile with a Body Actuator.

Open the Air Drag subsystem. If you double-click the block, a mask dialog box opens asking for the drag coefficient  $C_d$ . If you right-click the block and select **Look under mask**, the subsystem itself appears:



The Air Drag subsystem computes the air friction according to a standard air friction model. (See the Aerospace Blockset documentation for more information.) The drag always opposes the projectile's motion and is proportional to the product of the air density, the projectile's cross-sectional area, and the square of its speed.

Run the model with the default drag coefficient (zero). The XY Graph window opens to plot the parabolic path of the projectile. Now open the Air Drag dialog again and experiment with different drag coefficients  $C_d$ . Start with small values such as  $C_d = 0.05$ . For a rigid sphere,  $C_d$  is two. The effect of the drag is dramatic in that case.

## Applying Motions to Bodies

The Body Actuator block cannot actuate a Body with motion signals. But you can construct such body motion actuators with a combination of other blocks. See "Joint Actuator Example: Body Driver" on page 4-46.

## Varying a Body's Mass and Inertia Tensor

The Variable Mass & Inertia Actuator block gives you a way to vary a body's mass and/or inertia tensor as external functions of time. You specify these functions with incoming Simulink signals.

The Variable Mass & Inertia Actuator block does not apply any thrust forces or torques to the Body so actuated. Mass loss or gain in a particular direction results in thrust forces and torques. You must apply these forces and torques to the Body separately with Body Actuator blocks.

The Variable Mass & Inertia Actuator block changes the actuated Body's mass and rotational inertia by attaching an invisible body to the actuated body at a particular Body coordinate system (CS). This invisible body has a mass and an inertia tensor that vary in time as specified by the Actuator's external Simulink signal. SimMechanics treats the actuated body and the invisible body as a single composite body with a new mass, new center of gravity (CG), and new inertia tensor compounded from the two constituent bodies. You can add multiple Variable Mass & Inertia Actuator blocks to one Body.

To vary the mass and/or inertia tensor of a Body with this Actuator:

- 1** From the Sensors & Actuators library, drag a Variable Mass & Inertia Actuator block into your model.
- 2** Attach the Actuator's connector port to the Body CS on the Body where you want the invisible variable mass to be. If a suitable Body CS port does not exist on the Body, open its dialog and create one.
- 3** Create an external Simulink signal to model the time-varying mass and/or inertia tensor for this invisible body. Connect it to the Variable Mass & Inertia Actuator block's Simulink input port.

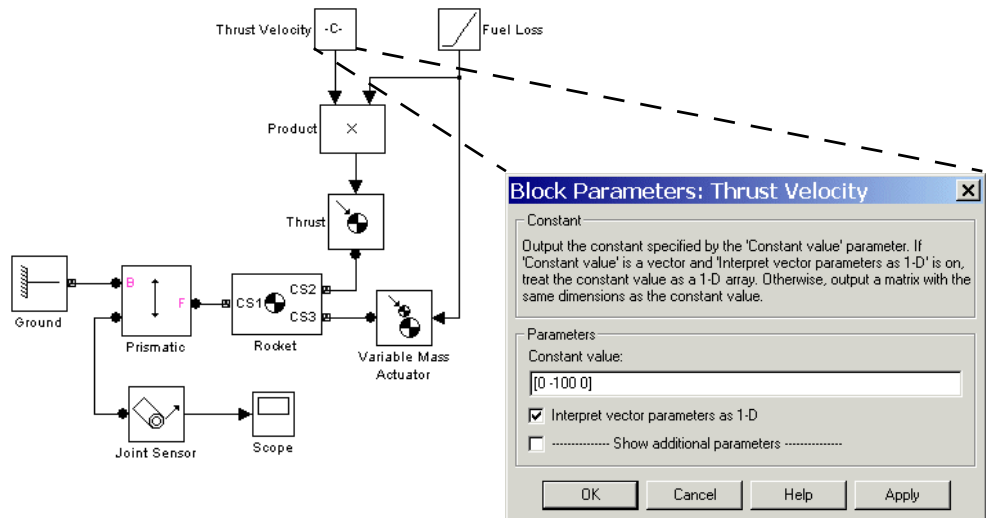
This Simulink signal can have one, nine, or ten components, depending on whether you are varying the mass only, the inertia tensor only, or both.

### Example: Point Rocket

The following model simulates a simple rocket. It treats the rocket as a point mass moving upward (+y direction) with an exhaust pointing downward (-y direction). The rocket loses mass at a constant rate.

The Rocket block is the point mass. The Thrust Velocity block represents the downward exhaust and, multiplied by the mass loss represented by the Fuel Loss block, actuates the Rocket body with a thrust force pointing upward. The Thrust block (a body actuator) applies this force at the *local* Body CS, which, for a point rocket, is identical to the Rocket's CG CS.

The same mass loss from the Fuel Loss block that produces the thrust force also must vary the rocket's mass directly. The Variable Mass Actuator block accomplishes this by feeding the same mass loss signal to the Rocket block.



## Actuating a Joint

You individually actuate each of the prismatic and revolute primitives of an assembled joint. You can apply

- Forces or translational motions (but not both) to prismatic primitives
- Torques or rotational motions (but not both) to revolute primitives

---

**Note** You cannot actuate spherical or weld primitives, disassembled joints, or massless connectors.

---

To actuate a prismatic or revolute joint primitive of an assembled joint:

- 1 Create an Actuator port on the Joint block for the primitive (see “Creating Actuator and Sensor Ports on a Joint” on page 4-24).
- 2 Drag a Joint Actuator or Joint Stiction Actuator from the Sensors & Actuators library into your model and connect its output port to the Actuator port on the Joint.

The remaining steps in this procedure apply to the creation of a standard Joint Actuator. For information on creating a stiction actuator, which applies classical Coulombic friction to a prismatic or revolute joint, see the Joint Stiction Actuator block reference page.

- 3 Open the Joint Actuator’s dialog box.
- 4 Select the primitive you want to actuate from the **Connected to primitive** list on the dialog box.
- 5 Select the type of actuation you want to apply, either **Generalized forces** or **Motion**.
- 6 If you are actuating a prismatic primitive:
  - If you selected **Generalized Forces** as the actuation type, select the units of force from the **Apply force** list.
  - If you selected **Motion** as the actuation type, select the units for each motion to be actuated (position, velocity, acceleration).
- 7 If you are actuating a revolute primitive:
  - If you selected **Generalized Forces** as the actuation type, select the units of torque from the **Apply torque** list.
  - If you selected **Motion** as the actuation type, select the units for each motion to be actuated (angle, angular velocity, angular acceleration).

- 8 Click **OK** to apply your choices and dismiss the dialog box.

Each joint primitive that you motion-actuate is lost as a true degree of freedom in your machine. That is because the DoF can no longer respond freely to externally applied forces or torques. See “Counting Degrees of Freedom” on page 4-67.

- 9 Create a signal that specifies the applied force, torque, or motions at each time step.

You can use any Simulink source block or any combination of blocks to create the actuator signal. You can also connect the output of a Sensor block attached to the Joint to the Actuator input, thereby creating a feedback loop. You can use such loops to model springs and dampers attached to the joint.

A force or torque signal must be a scalar signal. A motion signal must be a 1-D array signal comprising three components: position, velocity, and acceleration. The directionality of the joint determines the response of the follower to the sign of the actuator signal (see “Joint Directionality” on page 4-21).

- 10 Connect the Actuator signal to the Actuator port on the Joint.

---

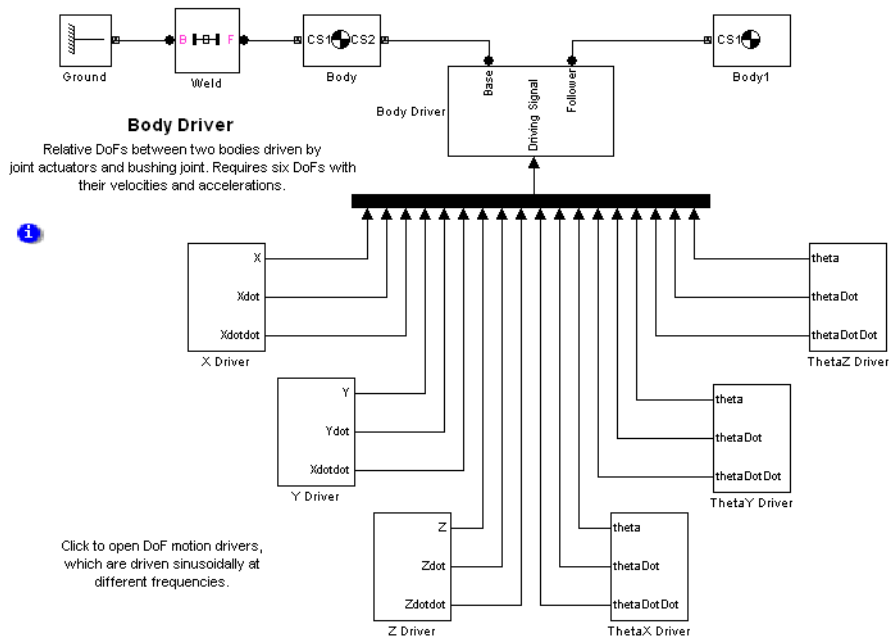
**Note** SimMechanics allows you to connect multiple Actuators to the same primitive. But it halts and displays an error message if you attempt to update or simulate a model containing such a connection. *Exception:* You can apply a Joint Initial Condition Actuator and force or torque actuation (including stiction) to the same primitive. See “Specifying Initial Positions and Velocities” on page 4-50.

---

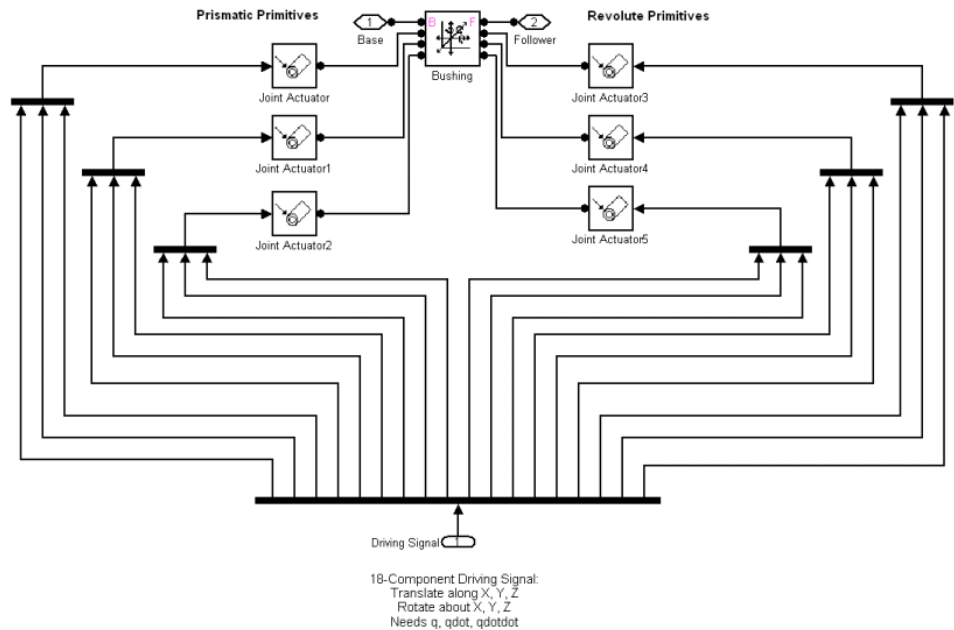
### Joint Actuator Example: Body Driver

The `mech_body_driver` model illustrates the use of Joint Actuators to create a custom driver.





The Body Driver subsystem accepts an 18-component signal that feeds the coordinates, velocities, and accelerations for all six relative DoFs between Body and Body1. The subsystem uses a Bushing block that contains three translational and three rotational primitives to represent the relative DoFs:

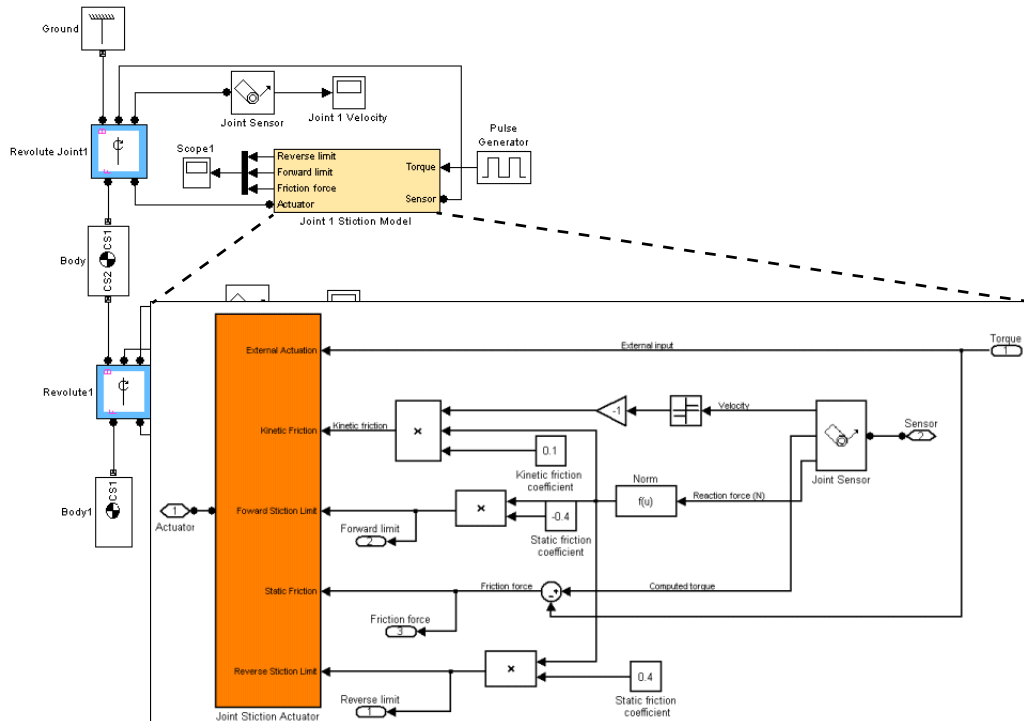


You can modify the body driver to move only one of the bodies, thereby creating a motion actuator. To move Body1 relative to World, for example, remove the blocks Body and Weld and connect the subsystem Body Driver directly to Ground.

### Joint Stiction Actuator Example: Mixed Static and Kinetic Friction

The `mech_dpen_sticky` model in the Demos library illustrates a driven double pendulum, with “sticky” friction or stiction applied to both revolute joints with the Joint Stiction Actuator block.

Open the unmasked Joint1 or Joint2 Stiction Model blocks (marked in yellow) to view the subsystems:



Each Stiction subsystem contains a Joint Stiction Actuator block (marked in orange) that requires static and kinetic friction coefficients via their respective blocks. For either revolute, an angular velocity threshold, specified through the block dialog, determines if a joint locks. Once locked, the joint cannot move until a combination of forces reaches a threshold specified by the Forward Stiction Limit or Reverse Stiction Limit.

Run the model with different kinetic and static friction coefficients and different velocity thresholds. View the results in the Scope blocks and through a visualization window. You can find more details on how stiction works in SimMechanics by consulting the Joint Stiction Actuator block reference page.

## Actuating a Driver

Actuating a Driver allows you to specify the time dependence of the rheonomic constraint applied by the Driver.

To actuate a Driver:

- 1 Create an additional connector port on the Driver for the Actuator.

Create the additional port in the same way you create an additional Driver port on a Joint (see “Creating Actuator and Sensor Ports on a Joint” on page 4-24).

- 2 Drag an instance of a Driver Actuator from the Sensors & Actuators library into your model.
- 3 Connect the Actuator’s output port to the Actuator port on the Driver.
- 4 Create a signal that specifies the time dependence of the Driver constraint.
- 5 Connect the actuation signal to the input port of the Driver Actuator.

### Specifying Initial Positions and Velocities

The Joint Initial Condition Actuator (JICA) block allows you to specify the initial positions and velocities of unactuated joints and hence the bodies attached to them. You can use JICA blocks to

- Specify nonzero initial joint velocities  
The default initial velocity of a joint primitive is zero. You must use a JICA block to specify a joint’s initial velocity if the initial velocity is not zero.
- Override the initial position settings of a body pair  
The CG CS origin settings in the dialog boxes of Body blocks specify the bodies’ initial positions. Using JICA blocks, you can override these initial body positions by resetting their relative positions in the Joints connecting them.

### Using JICA Blocks

Specifying initial conditions on a joint primitive is a special kind of actuation: one that occurs only once at the beginning of simulation. That is why the JICA block resides in the Sensors & Actuators library.

---

**Note** A JICA block, unlike other Actuators, does not have an input port. The JICA's dialog box completely specifies the Actuator input.

---

With a JICA block, you can specify the initial positions and velocities of any combination of prismatic and revolute primitives within a given Joint. (You cannot specify ICs for spherical and weld primitives.)

To specify the initial velocity and/or position of a joint primitive:

- 1** Drag a JICA block from the Sensors & Actuators library and drop it into your model window.
- 2** Create an additional connector port on the Joint block containing the primitive whose initial condition you want to specify.
- 3** Connect the connector port on the JICA block to the new connector port on the Joint block.

---

**Note** Do not connect the JICA block to the Joint ports marked “B” or “F” (base or follower). These ports are intended for connecting to Bodies.

---

- 4** Open the JICA block's dialog box. From the primitive list for the Joint, choose the primitives you want to actuate by selecting their check boxes.
- 5** Enter the initial positions of the actuated primitives, relative to the Body CSs attached to the Joint, in the **Position** field.

From the pull-down menu on the right, select **Units** for the initial positions.

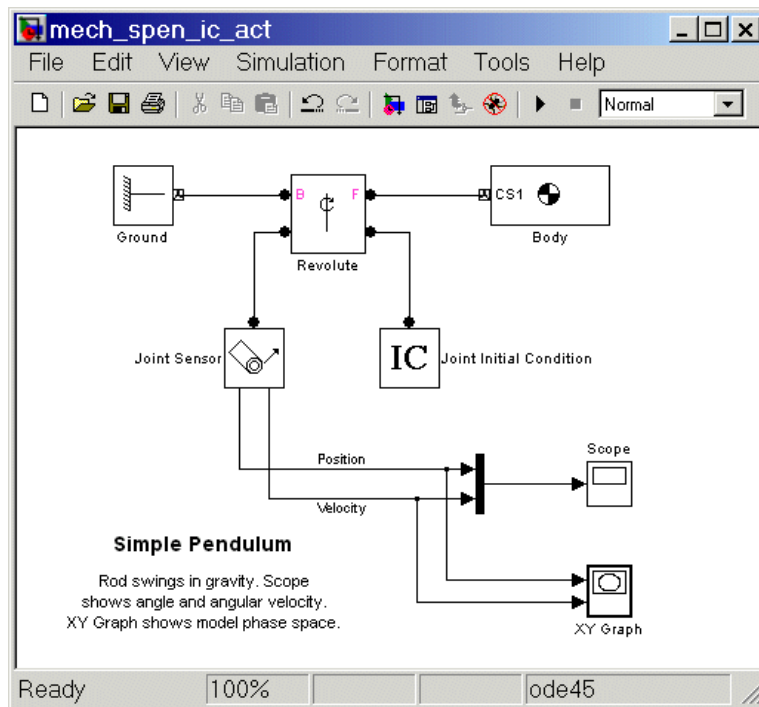
- 6** Enter the initial velocities of the actuated primitives, relative to the Body CSs attached to the Joint, in the **Velocity** field.

From the pull-down menu on the right, select **Units** for the initial velocities.

- 7** Click **Apply** or **OK**.

## JICA Example: A Simple Pendulum

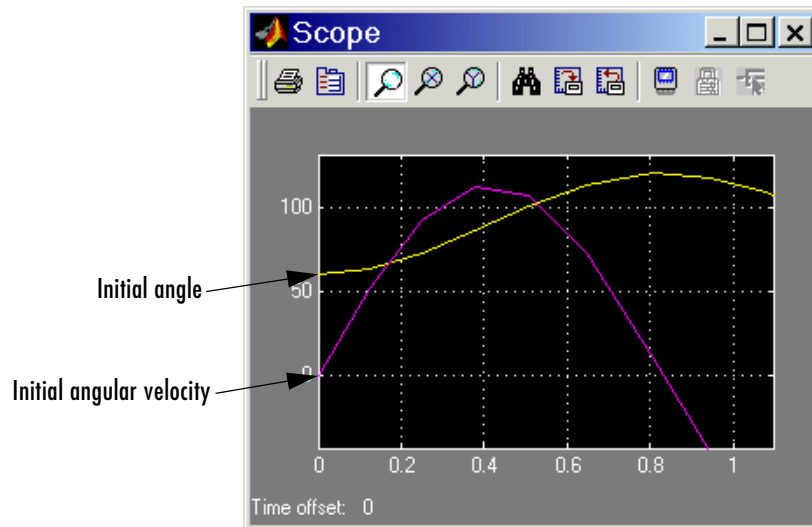
Open mech\_spen from the Demos library, then open the Sensors & Actuators library. Follow the steps from the preceding section, “Using JICA Blocks” on page 4-50, to connect one Joint Initial Condition Actuator block to the Revolute block and configure it. This Joint contains only one primitive, R1, which is the primitive listed in the JICA dialog box.



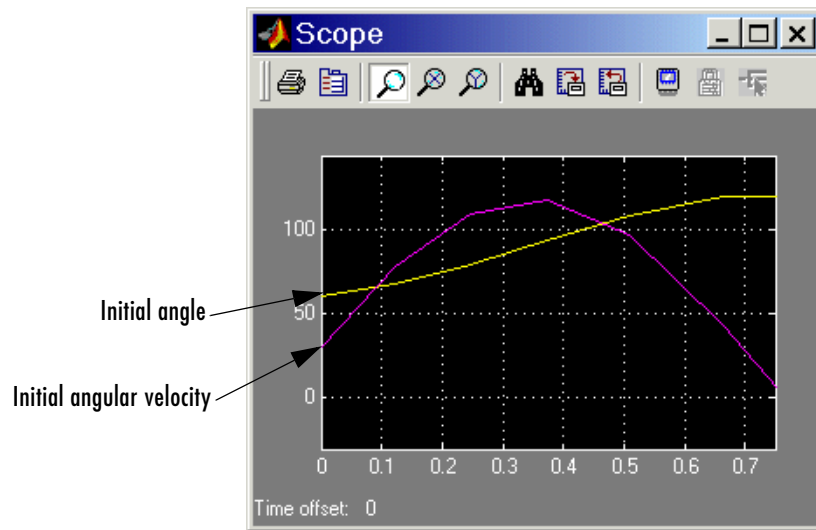
Set the initial conditions in two ways and compare the resulting simulations in the scope:

- 1 First set the initial **Position** (angle) to 60 deg, which is 60° down from the left horizontal (30° clockwise from vertically down), and set the initial **Velocity** to 0 deg/s.

- 2 Run the simulation for one second. Note in the scope that the initial angle (yellow curve) is displaced upward to  $60^\circ$ , while the initial velocity (purple curve) still starts at zero.



- 3 Now reset the initial **Velocity** to 30 deg/s, leaving the initial **Position** (angle) at 60 deg.
- 4 Rerun the simulation for one second. Note in Scope that the initial angle is still displaced upward to  $60^\circ$ , but the initial velocity is also displaced upward to 30°/sec.



The joint directionality is assigned in `mech_spen` so that the positive rotation axis is the  $+z$ -axis. Looking from the front, positive rotation swings down and right, counterclockwise.



## Modeling Force Elements

Internal forces are forces the machine applies to itself as a result of its own motion. Unlike actuation forces, you do not apply these forces from outside the machine with Simulink signals. The body motions instead generate the forces and torques directly.

The Force Elements library provides ready-made blocks to represent certain kinds of internal forces and torques acting between bodies.

- “Inserting a Linear Force Between Bodies”
- “Inserting a Linear Force or Torque Through a Joint” on page 4-57

You can also create your own customized sensor-actuator feedback loops to model springs, dampers, and more complex internal forces.

- “Customizing Force Elements with Sensor-Actuator Feedback” on page 4-58

### Inserting a Linear Force Between Bodies

A generalized linear force between two bodies is a linear function of the two bodies’ relative displacement vector  $\mathbf{r}$  and relative velocity  $\mathbf{v}$ , with constant coefficients. The Body Spring & Damper block models a force acting between two bodies along the axis  $\mathbf{r}$  connecting them:

$$\mathbf{F} = -k(\mathbf{r} - r_0) - b\mathbf{v}_{||}$$

The block is connected on either side to Bodies at a Body coordinate system (CS). The displacement  $\mathbf{r}$  is a vector from one Body CS on one Body to the other Body CS on the other Body. Newton’s third law requires that the forces that the bodies exert on one another be equal and opposite.

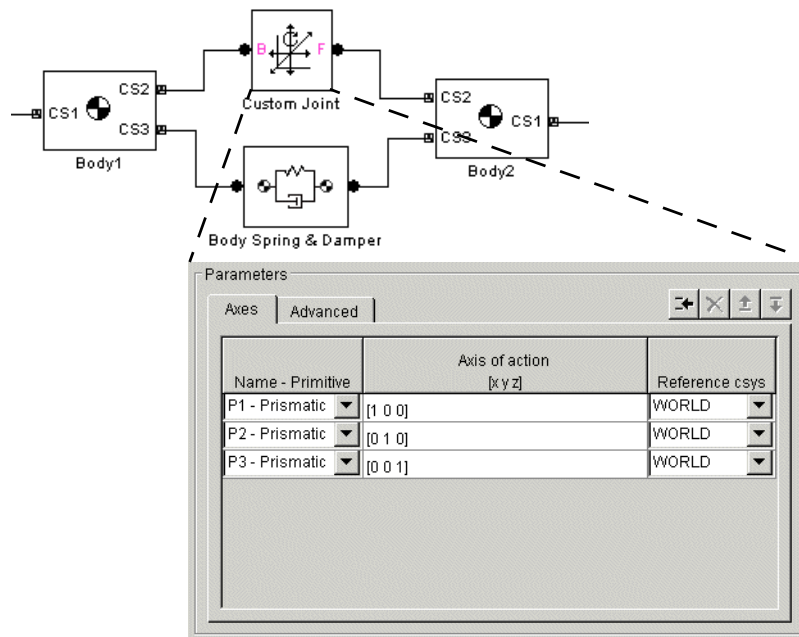
The common physical system this force model represents is a spring-damper combination, where the damper is a dashpot acting only along the spring axis. The damping is solely a function of the component  $v_{||}$  of the velocity vector projected along the displacement  $\mathbf{r}$ . (Thus the damping in this block cannot represent the damping due to a viscous medium, because there is no damping force perpendicular to the spring axis. See “Inserting a Linear Force or Torque Through a Joint” on page 4-57.)

You enter the constant parameters  $r_0$ ,  $k$ , and  $b$  in the Body Spring & Damper dialog.  $r_0$  is the spring’s natural length, the length it has when no forces are

acting on it. The spring constant  $k$  and damping constant  $b$  should be nonnegative.

To complete a linear force model between bodies, you need to model the translational degrees of freedom (DoFs) between them, as the Force Element block itself does not represent these DoFs. You can use any Joint block containing at least one prismatic primitive to represent translational motion. The two Bodies, the Joint, and the Body Spring & Damper must form a closed loop.

The following block diagram represents two Bodies with a damped spring between them. The Custom Joint represents the bodies' relative translational DoFs with three prismatic primitives. In this case, CS2 and CS3 on Body1 are the same, and CS2 and CS3 on Body2 are the same. Thus, the Joint is connected to the same Body CSs that define the ends of the spring-damper axis.



## Inserting a Linear Force or Torque Through a Joint

Another way of inserting a linear force element between two bodies is to connect it to a joint that already connects the bodies. You have to apply the force element, like an actuator, to each primitive in the joint individually. This approach has several advantages over the Body Spring & Damper:

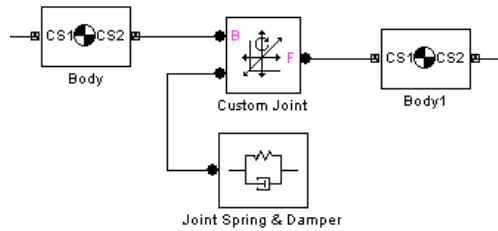
- You can create a different force law, with a different spring length, spring constant, and damping constant, for each of the joint's primitives.
- The spring and damper forces acting on each primitive act independently in their respective directions, instead of depending on just the interbody distance with a single spring length, spring constant, and damping constant. This allows you to create spring and damping forces that act independently in two or three dimensions, unlike the Body Spring & Damper force, which acts only along a single axis. Damping forces acting on multiple primitives act as a two- and three-dimensional viscous medium, not as a dashpot.
- The joint representing the DoFs between the bodies is already present.

You use the Joint Spring & Damper block to implement such spring-damper forces/torques together with a Joint. With it, you can apply a linear spring and damper force to each prismatic primitive and a linear torsion and damper torque to each revolute primitive in a Joint block. (You cannot apply these torques to a spherical primitive.)

Pick a Joint already connected between two Bodies. You connect the Joint Spring & Damper block to a Joint block at a sensor/actuator port on the Joint. (The section “Actuating a Joint” on page 4-44 explains how to create such a port.) The Joint Spring & Damper dialog then lists each primitive in the Joint.

For each prismatic primitive you want to actuate with a spring-damper force, you specify a natural spring length (offset), spring constant, and damping constant. For each revolute primitive you want to actuate with a torsion-damper torque, you specify a natural torsion angle (offset, or angle in which the primitive points absent any torques), torsion constant, and damping constant. You make these specifications in the Joint Spring & Damper dialog.

Here are two bodies connected by a Custom Joint in turn connected to a Joint Spring & Damper block.



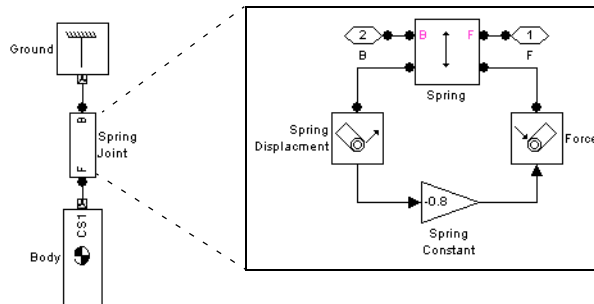
Unlike the example in the preceding section, “Inserting a Linear Force Between Bodies” on page 4-55, the Custom Joint can have up to three prismatic and three revolute, each with a separate linear force or torque acting through it. Each force or torque acts equally and oppositely on each body, following Newton’s third law.

## Customizing Force Elements with Sensor-Actuator Feedback

You can create your own force elements acting through Joints or on Bodies by using Sensor-Actuator feedback loops. With this technique, you can not only model linear forces, but any force that depends on body or joint positions and velocities.

This simple example illustrates the method with a linear spring force law. Hooke’s law states that the force exerted by an extended spring is proportional to its displacement from its unextended position:  $F = -kx$ .

The following SimMechanics model represents a spring that obeys Hooke’s law.



The model uses the Gain block labeled Spring Constant to multiply the displacement of the prismatic joint labeled Spring along the World's  $y$ -axis by the spring constant  $-0.8$ . The output of the Gain block is the force exerted by the spring. The model feeds the force back into the prismatic joint via the Actuator labeled Force. The model encapsulates the spring block diagram in a subsystem to clarify the model and to allow a spring to be inserted elsewhere.

# Modeling Sensors

The SimMechanics Sensors & Actuators library provides a set of Sensor blocks that enable you to measure

- Body motions (see “Sensing Body Motions” on page 4-60)
- Joint motions and forces or torques on joints (see “Sensing Joint Motions and Forces” on page 4-62)
- Constraint reaction forces and torques (see “Sensing Constraint Reaction Forces” on page 4-62)

---

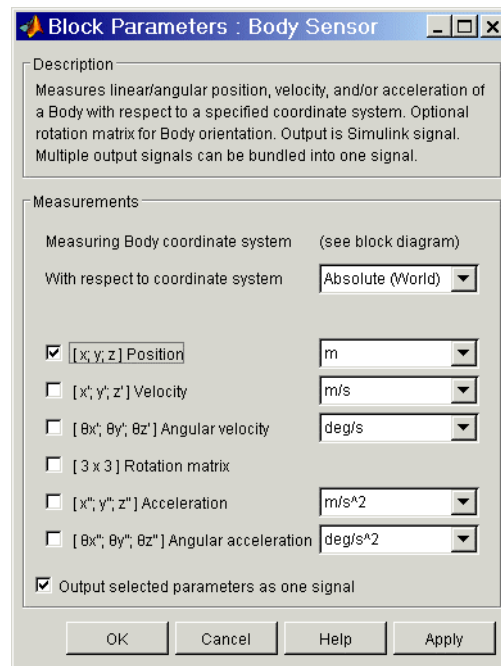
**Note** You can feed Sensor output back into Actuator blocks to model springs, dampers, and other mechanical devices that depend on force feedback. See “Actuating a Body” on page 4-40, “Actuating a Joint” on page 4-44, and “Checking Model Validity” on page 4-64.

---

## Sensing Body Motions

To sense the position, velocity, or acceleration of a body represented by a Body block:

- 1 If the Body block does not have a spare local coordinate system with a Body CS port, create one (see “Managing Body Coordinate Systems” on page 4-15).
- 2 Drag a Body Sensor block from the Sensors & Actuators library into your model.
- 3 Connect its connector port to a spare Body CS port on the Body.
- 4 Open the Sensor’s dialog box.



- 5 Select the coordinate system relative to which the sensor measures its output from the **With respect to coordinate system** list.
- 6 Select the check boxes next to the motions that you want to sense (see the Body Sensor block reference page).
- 7 If you have chosen to sense more than one type of motion and want the Sensor to multiplex the motions into a single output signal, select the **Output selected parameters as one signal** check box.
- 8 Click **OK** or **Apply**.
- 9 Connect the output of the Body Sensor block to a Simulink Scope or other signal sink or to a motion feedback loop, depending on your needs.

### Sensing Joint Motions and Forces

The SimMechanics Joint Sensor block enables you to measure the motions of joints. It can also measure the relative forces and torques between the bodies connected to the joint. These include the *computed force or torque* (the force or torque needed to reproduce the joint's motion) and the *reaction force and torque* on a joint primitive. (You cannot measure the computed force or torque on a spherical or weld primitive.) You must connect a separate Joint Sensor block to a Joint block for each joint primitive that you want to sense.

To sense the motions, forces, and torques of a joint primitive contained by a Joint block:

- 1 If the Joint block does not have a spare Sensor port, create one (see “Creating Actuator and Sensor Ports on a Joint” on page 4-24).
- 2 Drag a Joint Sensor block from the Sensors & Actuators library into your model.
- 3 Connect its connector port to the spare Sensor port on the joint.
- 4 Use the Sensor's dialog box to configure the Sensor to measure the motions, forces, and torques that you want to measure (see the Joint Sensor block reference page).
- 5 Connect the output of the Joint Sensor block to a Simulink Scope or other signal sink or to a motion feedback loop, depending on your needs.

### Sensing Constraint Reaction Forces

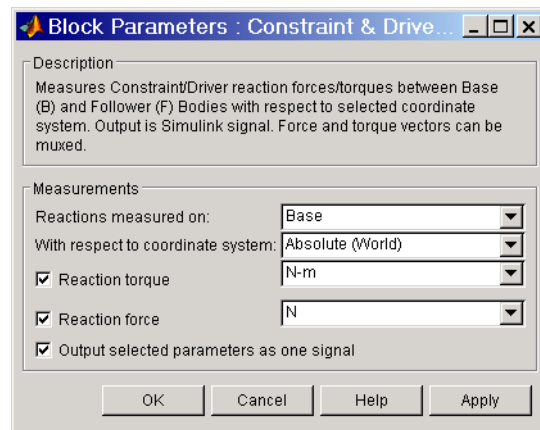
The SimMechanics Constraint & Driver Sensor block enables you to measure the reaction forces and torques induced on the constraints modeled by SimMechanics Constraint and Driver blocks.

To sense the reaction force and/or torque induced by a Constraint or Driver block:

- 1 If the Constraint or Driver block does not have a spare Sensor port, create one.
- 2 Drag a Constraint & Driver Sensor block from the Sensors & Actuators library into your model.



- 3 Connect its connector port to a spare Sensor port on the Constraint or Driver block.
- 4 Open the Sensor block's dialog box.



- 5 Select the body (follower or base) on which to measure the reaction force from the **Reactions measured on** list.
- 6 Select the coordinate system relative to which the Sensor measures its output from the **With respect to coordinate system** list.
- 7 Select the **Reaction torque** check box if you want the Sensor to output the reaction torque on the base (or follower) body.
- 8 Select the **Reaction force** check box if you want the Sensor to output the reaction force on the base (or follower) body.
- 9 If you have chosen to output both reaction force and torque and want the Sensor to multiplex them into a single output signal, select the **Output selected parameters as one signal** check box.
- 10 Click **OK** or **Apply**.
- 11 Connect the output of the Body Sensor block to a Simulink Scope or other signal sink or to a motion feedback loop, depending on your needs.

# Checking Model Validity

Simulink can simulate a SimMechanics model only if it is valid. A model is valid if it satisfies the following rules:

- Every schematic in the model is topologically valid (see “Checking Schematic Topology” on page 4-64).
- The model contains at least one degree of freedom (see “Counting Degrees of Freedom” on page 4-67).

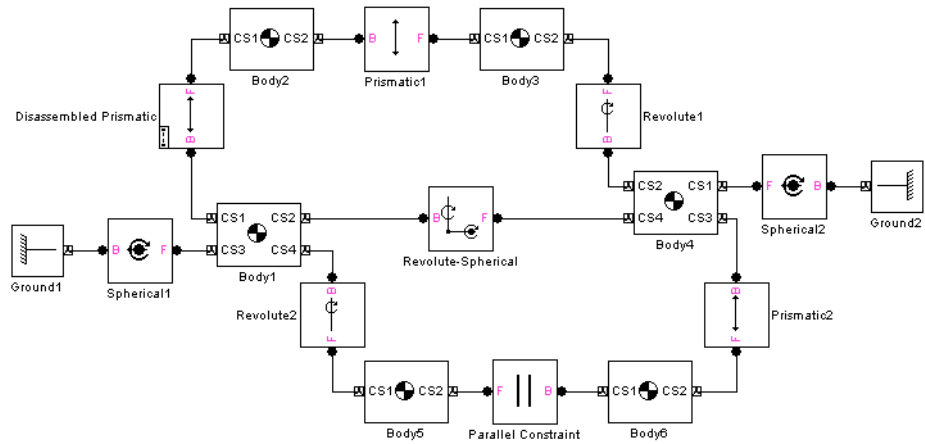
## Checking Schematic Topology

To avoid simulation failures, you must ensure that the topology of your block diagram is valid. A block diagram is topologically valid if each schematic that it contains is valid. A schematic is valid if its *spanning tree* is valid. Thus to determine if your model is valid, first determine the spanning tree of each schematic that it contains and then the validity of the resulting spanning trees.

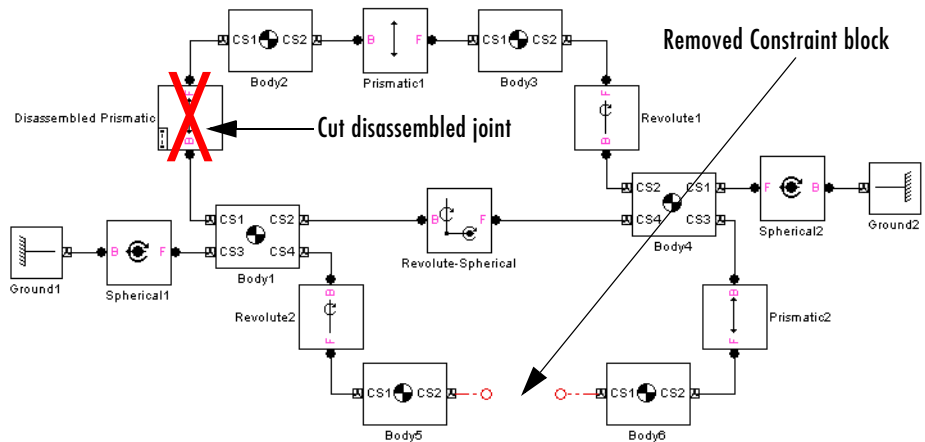
### Determining a Schematic’s Spanning Tree

To determine the spanning tree of a schematic, remove all blocks from the schematic except Body and Joint blocks and open every closed loop in the resulting reduced schematic. To open a closed loop, conceptually follow the loop-cutting rules in “Cutting Closed Loops” on page 4-32.

For example, here is a schematic with two closed loops.



Cutting the top loop at the Disassembled Prismatic and removing the Parallel Constraint block (thus simultaneously cutting the bottom loop) yields the schematic's spanning tree.



### Determining the Validity of a Spanning Tree

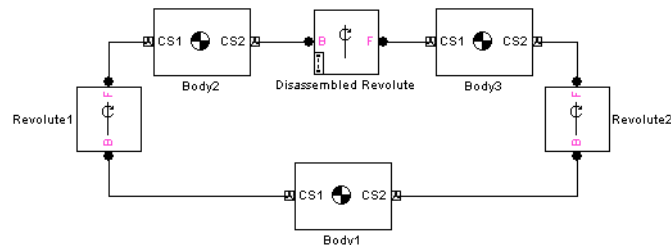
To be valid, a spanning tree must meet the following requirements:

- The spanning tree must have at least one Ground block to serve as a reference to World.
- Every Joint block must be connected to exactly two Body blocks.
- Every non-Ground Body block must have a unique path to a Ground block. (This need not be true of the model schematic itself.) This ensures that, while each body moves via joints relative to other bodies, SimMechanics can resolve all bodies' motions into absolute motions with respect to World.
- Every non-Ground Body block at an end of a sequence of Bodies must have nonzero inertia (mass or inertial moment) associated with all joint primitives that can move. Each translational DoF must carry a nonzero mass, and each rotational DoF a nonzero inertial moment. This prevents infinite accelerations when forces and torques are applied.

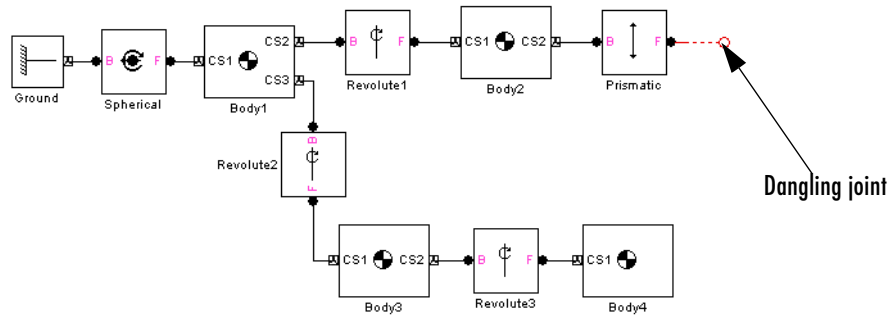
### Examples of Invalid Schematic Topologies

Here are some examples of invalid topologies:

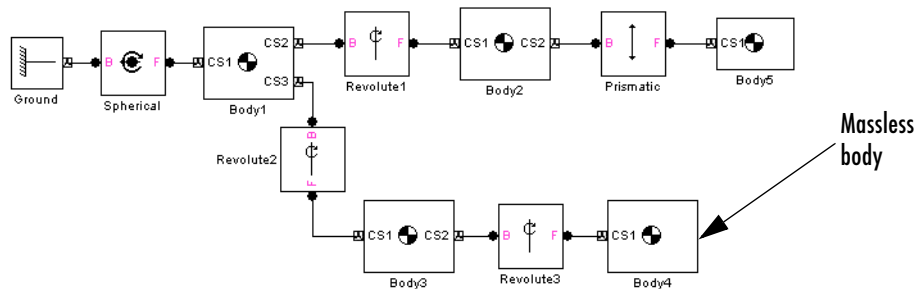
- This one-loop schematic lacks a Ground block.



- This open machine schematic has a dangling Joint block.



- Another open machine schematic features a zero-mass body at one end of a chain of bodies.



The last two invalid examples are dynamically (but not topologically) equivalent, because a zero-mass body is dynamically no body at all.

## Counting Degrees of Freedom

Identifying and counting the independent degrees of freedom (DoFs) of a machine are important for trimming and linearizing SimMechanics models (see “Trimming SimMechanics Models” on page 7-12 and “Linearizing SimMechanics Models” on page 7-23) and for correcting simulation errors (see “Troubleshooting Simulation Errors” on page 5-16).

Your SimMechanics model must have at least one DoF to be valid. A free physical body has six DoFs: three translational and three rotational. But in a

machine, connections between bodies by joints, constraints, and drivers, and motion actuation by joint and body actuators reduce the machine's independent DoFs to a smaller number. You also reduce a body's DoFs if you confine the machine's motion to one or two spatial dimensions.

In SimMechanics, a Body block has no DoFs. Connecting Joints to a Body adds DoFs to the machine. The joint primitives represent the Body's DoFs relative to other connected Bodies or Grounds. Connecting Constraint and Driver blocks to Bodies or motion-actuating joint primitives in Joints removes DoFs from the machine.

### Finding Independent Degrees of Freedom

Here is the formula for determining the *number of independent DoFs* your model has:

$$\begin{aligned} \# \text{ of independent DoFs} = & \# \text{ of body DoFs} + \# \text{ of primitive DoFs} - \\ & \# \text{ of motion restrictions} \end{aligned}$$

The following three steps define each term on the right-hand side:

- 1 Calculate the *number of body DoFs* from the number of Body and Joint blocks in your model:

$$\# \text{ of body DoFs} = 6 * (\text{number of Bodies} - \text{number of Joints})$$

If you have confined the machine to move in only two dimensions, replace the 6 by 3. If you have confined the machine to move in only one dimension, replace the 6 by 1.

- 2 Calculate the *number of primitive DoFs* by adding up the primitive DoFs from the Joint dialog boxes:
  - Count one for each prismatic (P) or revolute (R) primitive.
  - Count three for each spherical (S) primitive.
  - Count zero for each weld (W) primitive.

Do not count a primitive DoF that is motion-actuated by a Joint Actuator.

- 3 Calculate the *number of motion restrictions* by adding up the motion restrictions of each Joint Actuator, Constraint, and Driver block. Different

blocks from the Constraints & Drivers library impose different numbers of motion restrictions.

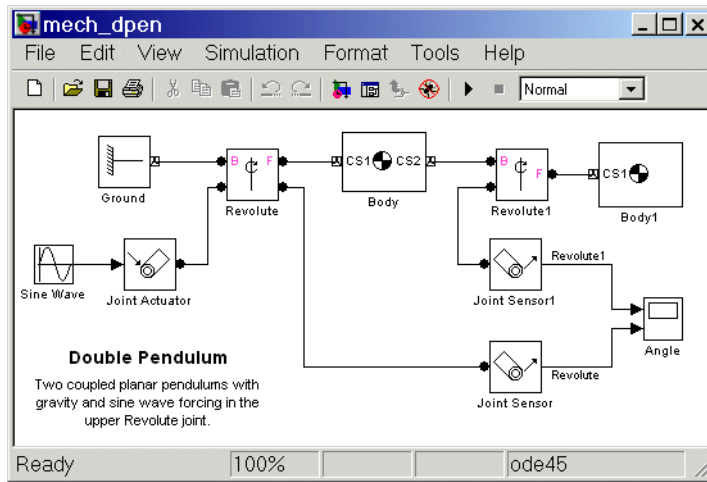
<b>Constraint Block</b>	<b>Restrictions</b>	<b>Driver Block</b>	<b>Restrictions</b>
Gear	One	Angle	One
Parallel	Two	Distance	One
Point-Curve	Two	Linear	One
		Velocity	One

Be sure not to count redundant motion restrictions. These are restrictions that forbid the motion of joint primitives that could not move anyway even if the constraint were removed, because of how the joints are configured.

*Example:* A body is connected to a ground by a single prismatic. You place a constraint on the body that prevents it from moving perpendicularly to the prismatic axis. The body could not move in that direction even if you removed the constraint. So the constraint is redundant, and you would not count it as a motion restriction.

### **DoF Example: Double Pendulum**

The `mech_dpen` model from the Demos library represents planar double pendulum motion actuated by a Joint Actuator.



The double pendulum has two rigid bodies, such as two rods, confined to move in two dimensions. Ignoring the Joint Actuator temporarily, there are two bodies, two joints, and two revolute primitives, and thus  $3 * (2 - 2) + 2 = 2$  independent DoFs. There are many ways to represent these two DoFs, but the two revolute primitives are the simplest way.

Including the Joint Actuator in the DoF count removes the revolute primitive in the Revolute block as an independent DoF. So this model actually only has one independent DoF, the revolute primitive in the Revolute1 block.

### DoF Example: Four Bar Mechanism

The four bar mechanism of “A Four Bar Mechanism” on page 2-36 has four revolutes. You can establish that only  $3 * (3 - 4) + 4 = 1$  of these DoFs is actually independent and arrive at the same result obtained in the Four Bar Mechanism example.



# Running Mechanical Models

---

SimMechanics gives you multiple ways to simulate and analyze machine motion in the Simulink environment. Running a mechanical simulation is similar to running a simulation of any other type of Simulink model. It entails setting various simulation options, starting the simulation, and dealing with simulation errors. See the Using Simulink guide for a general discussion of these topics. This chapter focuses on aspects of simulation specific to SimMechanics models.

Choosing Simulation Options  
(p. 5-2)

Special settings in SimMechanics and Simulink for running mechanical models

Mechanical Environment  
Settings Dialog Box (p. 5-7)

Controlling the SimMechanics simulation parameters

How SimMechanics Works  
(p. 5-14)

Overview of how SimMechanics analyzes and simulates a mechanical model

Troubleshooting Simulation  
Errors (p. 5-16)

Fixing SimMechanics simulation errors

Generating Code (p. 5-22)

Using Real-Time Workshop to translate your SimMechanics models into C code

Limitations (p. 5-25)

How to use compatible Simulink simulation tools with SimMechanics models

## Choosing Simulation Options

Simulink provides an extensive set of simulation options that apply to any type of model. SimMechanics provides additional options that apply specifically to simulating mechanical models. The following sections discuss those standard Simulink options for which mechanical models entail special consideration and the additional mechanical system-specific options of SimMechanics.

You choose the special mechanical settings in the **Mechanical Environment Settings** dialog (see “Mechanical Environment Settings Dialog Box” on page 5-7).

### Choosing an ODE Solver

SimMechanics uses an ODE solver to solve a system’s equations of motion, typically in tandem with a constraint solver (see “Choosing a Constraint Solver” on page 5-5). Simulink provides an extensive set of ODE solvers that represent the most advanced numerical techniques available for solving differential equations in general and equations of motion in particular. The **Solver** pane of a model’s **Simulation Parameters** dialog box allows you to select any of these solvers for use by Simulink in solving the model’s dynamics (see “The Solver Pane” in the Simulink User’s Guide more information).

### Solving Stiff Systems

The Dormand-Prince solver (ode45) that Simulink uses by default works well for many mechanical systems, but might require too much time to solve systems that are stiff, that is, have bodies that move at widely varying speeds or that have many discontinuities in their motion. An example of a stiff system is a pair of coupled oscillators in which one body is much lighter than the other and hence oscillates much more rapidly. Any of the following solvers might require significantly less time than the default solver to solve a stiff system:

- ode15s: Variable-order solver based on a backward differentiation rule (variant of Gear’s method).
- ode23t: Trapezoidal rule solver. Use this solver if your system is slightly stiff, to avoid numerical damping.
- ode23tb: Implicit Runge-Kutta method solver combining trapezoidal rule and a backward differentiation rule of order 2. More efficient than ode15s if the solution has many discontinuities.

- `ode23s`: Modified Rosenbrock method solver of order 2. This solver is also more efficient than `ode15s`, if the solution has many discontinuities.

Try each of these solvers in turn and then use the solver that gives the best results.

### Setting ODE Solver Tolerances

By default Simulink automatically determines the absolute tolerance used by ODE solvers. The resulting tolerance might not be small enough for a mechanical system, particularly a nonlinear or chaotic system. Try running a simulation with the relative tolerance set to  $1e-3$  (the default) and the absolute tolerance set to  $1e-4$ . Then increase the tolerances if the simulation takes too long or decrease them if the solution is not sufficiently accurate.

If your model contains one or more Joint Stiction Actuator blocks, you must also take into account the velocity thresholds of these blocks when setting the absolute tolerance of the ODE solver. If the absolute tolerance of the solver is greater than a joint's velocity threshold, the simulation might never detect the unlocking of a joint. To prevent this from happening, set the absolute tolerance to be no more than 10% of the size of the smallest stiction velocity threshold in your model.

### Handling Singularities

Singularities in a system's equations of motions can dramatically slow down a standard Simulink ODE solver or even prevent it from finding a solution to a system's equations of motions. Because singularities are frequent in mechanical equations of motion, SimMechanics provides an optional feature, called Robust Singularity Handling, that works together with the currently selected ODE solver to solve singular equations of motions efficiently. This feature allows Simulink to simulate models that otherwise cannot be simulated or cannot be solved in a reasonable amount of time because of the singularities.

To enable this feature, select **Use robust singularity handling** on the **Constraints** pane of the **Mechanical Environment Settings** dialog (see "Use robust singularity handling" on page 5-10). Because this option requires extra computation whether or not singularities exist, you should select this option only as a last resort, i.e., select this option only if you cannot find a standard Simulink solver that solves your model in a reasonable amount of time without it.

### Choosing an Analysis Mode

You can use SimMechanics to compute the motion that results from applying forces to a mechanical system or the forces required to produce a specified motion in a mechanical system. To compute either type of result, you must build an appropriate model and choose an appropriate mode of analysis.

#### Forward Dynamics Mode

Computes the positions and velocities of a system's bodies at each time step, given the initial positions and velocities of its bodies and any forces applied to the system. Use this mode to simulate a model that represents the initial positions and velocities of the system's bodies and the forces on those bodies.

You ran the examples from previous chapters in the Forward Dynamics mode

- “Running a Demo Model” on page 1-5
- “Building a Simple Pendulum” on page 2-11 and “A Four Bar Mechanism” on page 2-36

as well as the many examples of the “Modeling Mechanical Systems” chapter.

#### Inverse Dynamics Mode

Computes the forces required to produce a specified velocity for each body of an open-loop system. Use this mode to simulate an open-loop system whose model specifies the velocity of every degree of freedom of every body at every time step. See “Open-Topology Example: Double Pendulum” on page 7-7 for an example of using this mode to find the forces on an open-loop system.

#### Kinematics Mode

Computes the forces required to produce a specified velocity for each body of a closed-loop system. Use this mode to simulate a closed-loop system whose model specifies the velocity of every independent degree of freedom at every time step. See “Closed-Loop Example: Four-Bar System” on page 7-3 for an example of using this mode to find the forces on a closed-loop system.

#### Trimming Mode

This is a variant of Forward Dynamics mode that allows you to run the Simulink `trim` command on your model. The `trim` command in turn allows you to find steady-state solutions for your model (see “Equilibrium Point

Determination” in the “Analyzing Simulation Results” section of the Simulink documentation or enter `help trim` at the MATLAB command line).

Trimming mode inserts a subsystem and an output port at the top level of your model. These blocks output signals corresponding to the constraints on the system represented by your model. Configure the `trim` command to find equilibrium points where the constraint signals are 0. This ensures that the equilibrium points found by the `trim` command satisfy the constraints on the modeled system. See “Constrained Example: Four-Bar System” on page 7-18 for an example of using this mode to find the equilibrium points of a constrained system.

## Setting Assembly Tolerances

The linear and angular assembly tolerance specify the precision with which:

- A model must specify the initial locations and angles of a system’s joints.
- SimMechanics must solve the initial positions and angles of a model’s unassembled joints.

Simulink checks the locations and angles of a model’s assembled joints during the model initialization phase of a simulation. If any of the joint locations or angles fail to meet the corresponding assembly tolerances, Simulink halts the simulation and displays an error message. If this happens, you should check your model to ensure that it specifies the locations and angles of its assembled joints to the precision specified on the **Parameters** pane of the **Mechanical Environment Settings** dialog. If not, either change the locations and angles that fail to meet the assembly tolerances or increase the tolerances themselves.

## Choosing a Constraint Solver

If your model contains implicit or explicit constraints on a system’s motion, SimMechanics uses one of the three possible constraint solvers to find a solution to its equations of motion that meet the constraints:

- Stabilizing solver
- Tolerancing solver
- Machine precision solver

The following sections describe these solvers.

### Stabilizing Constraint Solver

Adds a self-correcting term to the state equations to be solved that stabilizes the numerical solution, i.e., causes it to gravitate toward, rather than drift away from, the actual solution. SimMechanics uses this solver by default. It is typically faster than the other solvers, but can stop at a solution that exceeds the model's assembly tolerances. If assembly tolerance errors occur during the simulation, use one of the following solvers instead.

### Tolerancing Constraint Solver

Solves the constraints on the system's states to the tolerance that you specify. Specifically the solver stops refining the numerical solution of the state equation when the difference between two successive solutions satisfies the equation

$$|error| < |rtol * x + atol|$$

where *error* is the difference between successive solutions, *rtol* is the relative constraint tolerance (see "Constraint relative tolerance" on page 5-10), *x* is the state to be solved, and *atol* is the absolute constraint tolerance (see "Constraint absolute tolerance" on page 5-10).

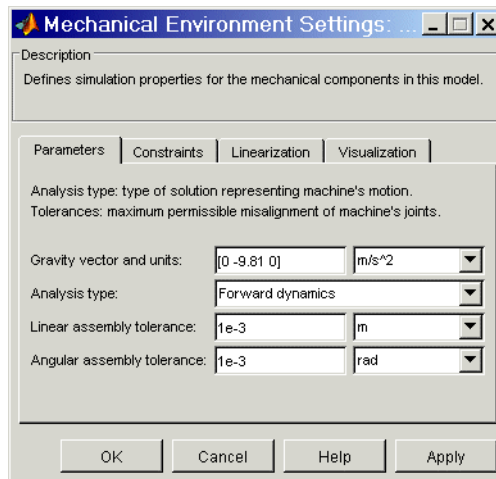
Use this solver if you plan to run the simulation in Kinematics mode and you want to trade off simulation accuracy for simulation time. Relaxing the constraint solver tolerances generally reduces the time required to simulate the model. Decreasing the tolerances increases the accuracy of the simulation but also increases the time required to simulate the model.

### Machine-Precision Constraint Solver

Solves the constraints to the numerical precision of the computer on which the simulation is running. Select this solver if you want to obtain the most accurate simulation permitted by the computer, regardless of simulation time.

## Mechanical Environment Settings Dialog Box

The **Mechanical Environment Settings** dialog box allows you to choose simulation options that are specific to models using SimMechanics blocks. To display this dialog, select **Mechanical environment** from the model's **Simulation** menu. The **Mechanical Environment Settings** dialog appears. The dialog groups SimMechanics global settings onto four tabbed panes: **Parameters**, **Constraints**, **Linearization**, and **Visualization**. The following sections explain the settings on each pane.



### Parameters Pane

This pane includes the following simulation settings.

#### Gravity vector and units

The value of this parameter is a MATLAB vector that specifies the magnitude and direction of gravitational acceleration in the model's world coordinate system. The adjacent list box specifies the units that the vector represents. The default vector is  $[0 \ -9.81 \ 0]$ . The default units are  $\text{m/s}^2$  (meters per square second). Use the menu to the right if you want to reset the units.

### Analysis mode

Specifies the type of analysis to be performed during the simulation. Choose one from the menu. See “Choosing an Analysis Mode” on page 5-4 for more information.

Analysis Mode	Description
Forward dynamics	Computes the positions and velocities of the system’s bodies, given forces, torques, and initial conditions. This is the default mode.
Inverse dynamics	Computes the forces and torques required to produce the specified motions of an open system (see “Open-Topology Example: Double Pendulum” on page 7-7).
Kinematics	Computes the forces and torques required to produce the specified motions of a closed-loop system (see “Constrained Example: Four-Bar System” on page 7-18).
Trimming	Variant of Forward Dynamics mode to be used with the Simulink <code>trim</code> command to determine steady-state or other points in the state space of a system (“Trimming SimMechanics Models” on page 7-12).

### Linear assembly tolerance

Maximum position error allowed between bodies connected by prismatic joints. The default is  $1e-3$  m. Use the menu on the right to set the units. See “Setting Assembly Tolerances” on page 5-5 for more information.

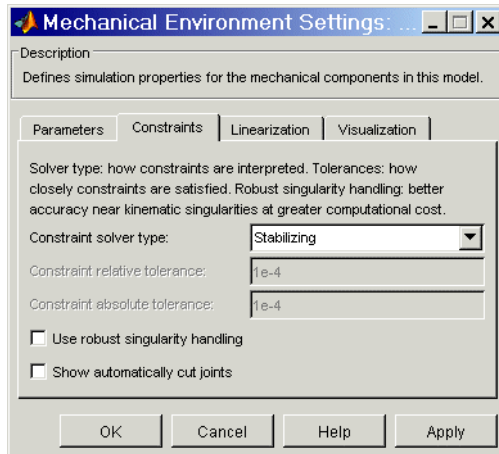
### Angular assembly tolerance

Maximum angular error allowed between bodies connected by revolute joints. Default is  $1e-3$  rad. Use the menu on the right to set the units. See “Setting Assembly Tolerances” on page 5-5 for more information.



## Constraints Pane

This pane contains settings for models that contain constraint or driver blocks.



### Constraint solver type

Type of solver used to solve constraints on the mechanical system's states specified by the model's constraint and driver blocks. Choose one from the menu. See "Choosing a Constraint Solver" on page 5-5 for more information.

Solver Type	Description
Stabilizing	Adds a self-correcting term to the state equations to be solved that stabilizes the numerical solution, i.e., causes it to gravitate toward, rather than drift away from, the actual solution. This is the default.
Tolerancing	Solves the constraints on the system's states to a specified degree of accuracy.
Machine precision	Solves the constraints to the numerical precision of the computer on which the simulation is running.

### Constraint relative tolerance

The relative tolerance used by the Tolerancing constraint solver to determine when to stop refining a solution. Default is  $1e-4$ . See “Tolerancing Constraint Solver” on page 5-6 for more information.

### Constraint absolute tolerance

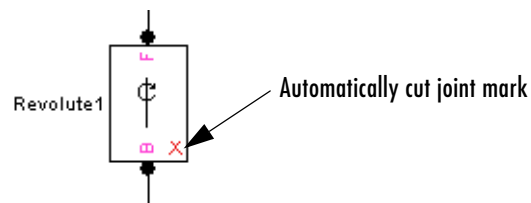
The absolute tolerance used by the Tolerancing constraint solver to determine when to stop refining the solution of a model state. Default is  $1e-4$ . See “Tolerancing Constraint Solver” on page 5-6 for more information.

### Use robust singularity handling

Select this check box if you want Simulink to take extra steps to handle singularities in a system’s equations of motion. This option increases the length of time required to solve a system’s equations of motion regardless of whether they have singularities. Hence, you should select this option only as a last resort, i.e., only if Simulink ODE solvers cannot otherwise solve the system’s equations of motion or require an excessively long time to do so. See “Handling Singularities” on page 5-3 for more information.

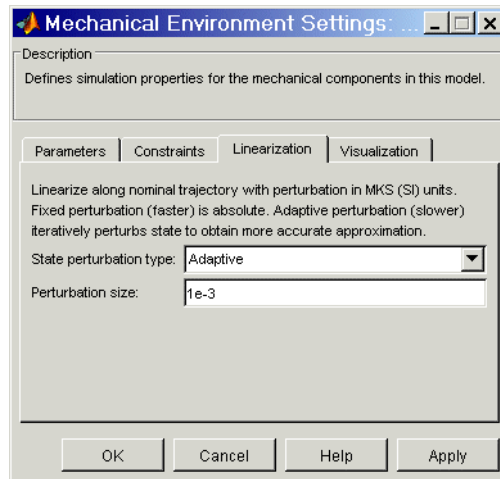
### Show automatically cut joints

Selecting this check box causes Simulink to mark the icons of Joint or Constraint/Driver blocks that it cuts during simulation of the model. See “Cutting Closed Loops” on page 4-32 for an additional discussion.



### Linearization Pane

This pane allows you to configure the state perturbation used by the Simulink model linearization command, `linmod`, when linearizing a SimMechanics model (see “Linearizing SimMechanics Models” on page 7-23).



### State perturbation type

Specifies the type of state perturbation used by `linmod` to linearize a mechanical model. The default is **Fixed**.

- **Adaptive** recomputes the size of the perturbation used at each step in the linearization process to ensure accurate computation of the linearization coefficients. It starts with the entry in the **Perturbation size** field as an initial guess.
- **Fixed** uses the perturbation size specified in the **Perturbation size** field for every step.

**Fixed** perturbation is usually faster but can also be less accurate than **adaptive** perturbation. You should choose the **fixed** option only if the **adaptive** option takes too long and you are willing to devote time to finding, either analytically or by trial and error, a fixed perturbation size that produces an accurate linearization of your model. If you choose the **fixed** option, you should check its results against the linearization produced by the **adaptive** option.

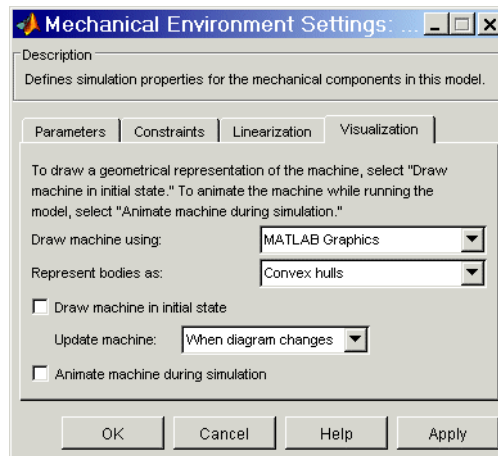
### Perturbation size

Specifies the relative size of the perturbation used by the **Fixed** perturbation option. Specifies the relative size of the initial guess perturbation used by the

Adaptive perturbation option. The perturbation size is relative to the size of the state being perturbed. The default is  $1e-5$ .

### Visualization Pane

This pane enables you to select various visualization options for your model. For additional information on visualization, see Chapter 6, “Visualizing and Animating Machines.”



#### Draw machine using

Specifies the visualization tool that Simulink uses to render your model when you are building and/or simulating the model. If the Virtual Reality Toolbox is installed on your system, the adjacent menu allows you to choose between MATLAB Graphics or the Virtual Reality Toolbox. If the Virtual Reality Toolbox is not installed on your system, Simulink uses MATLAB (Handle) Graphics to render your model.

#### Represent bodies as

Specifies the type of shape used to render the model's bodies. You can choose Equivalent ellipsoids or Convex hulls. See “Rendering Body Shapes in SimMechanics” on page 6-5.

**Draw machine in initial state**

Selecting this check box causes Simulink to render the model in its initial state when you select the **Apply** or **OK** button on the **Mechanical Environment Settings** dialog and at the times specified by the **Update machine** option.

**Update machine**

Specifies when Simulink updates the initial state rendering of your model. The menu options are

- When diagram changes (whenever you change your block diagram)
- Only at Update diagram (whenever you select **Update diagram** from the model's **Edit** menu)

**Animate machine during simulation**

Selecting this check box causes Simulink to animate the model's visualization during simulation, i.e., recreate the rendering of the model at each time step to show the current positions of the model's bodies.

## How SimMechanics Works

You might find a brief overview of how SimMechanics works helpful for understanding and fixing errors, a topic discussed in “Troubleshooting Simulation Errors” on page 5-16.

There are four major steps of machine simulation. The first two occur before SimMechanics actually starts machine motion.

- “Model Validation”
- “Machine Initialization”
- “Force Analysis and Motion Integration” on page 5-15
- “Stiction Mode Iteration” on page 5-15

### Model Validation

SimMechanics first checks your data entries from the dialogs and the local connections among neighboring blocks. It then validates the Body coordinate systems; the joint, constraint, and driver geometries; and the model topology.

### Machine Initialization

The assembly tolerances of Joints that you manually assembled are checked next.

SimMechanics then cuts each closed loop once. An invisible internal constraint replaces each cut Joint, Constraint, or Driver block. SimMechanics checks all constraints and drivers for mutual consistency and eliminates redundant constraints.

Any Joint Initial Condition Actuators now impose initial positions and velocities, changing body geometries from their dialog box configurations as necessary. SimMechanics then finds an assembly solution for disassembled joints and initializes them in position and velocity. Assembly tolerances are checked again.

A “sticky” joint primitive, actuated by a Joint Stiction Actuator, can be in one of three stiction modes: locked, waiting, or unlocked. SimMechanics finds a mutually consistent set of stiction modes for all sticky joints.

## **Force Analysis and Motion Integration**

In the Forward Dynamics or Trimming analysis modes, SimMechanics now begins the solution of machine motion by applying and integrating external forces and torques, stepping in simulation time. It maintains assembly, constraint, and solver tolerances and checks constraint and driver consistency. It also detects whether, within one Joint block, distinct joint primitive axes align and destroy an independent DoF (joint axis singularity).

In the Inverse Dynamics and Kinematics modes, SimMechanics now applies motion constraints, drivers, and actuators to find the machine motion and derive forces and torques. It also checks tolerances and consistency and detects singular alignment of joint primitives.

## **Stiction Mode Iteration**

If stiction is present, SimMechanics checks at each time step whether the sticky joints transition from one stiction mode to another, then checks for mutual consistency of locked and unlocked sticky joint primitives across the whole machine.

## Troubleshooting Simulation Errors

SimMechanics simulations can stop before completion with one or more error messages. You might find the previous section, “How SimMechanics Works” on page 5-14, useful for tracing errors. Some common errors also appear in “Modeling Machines” on page 4-2 and “Checking Model Validity” on page 4-64. This section discusses generic error types.

Most errors and error-fixing strategies fall into broad categories. These groupings are reflected in the keywords occurring in the error messages that SimMechanics displays. The following sections summarize these groupings.

- “Data Validation Errors”
- “Ground and Body Geometry Errors”
- “Joint Geometry Errors” on page 5-17
- “Block Connection and Topology Errors” on page 5-17
- “Motion Singularity and Inconsistency Errors” on page 5-18
- “Analysis Mode Errors” on page 5-21

### Data Validation Errors

Every numerical entry you make in SimMechanics must be a real numerical expression or MATLAB equivalent. Spatial vectors are 3-vectors, such as [3 4 5]. Spatial tensors are 3-by-3 matrices, such as rotation matrices and the inertia tensor.

---

**Note** You can specify a two-dimensional curve in the Point-Curve Constraint block with 2-vectors.

---

### Ground and Body Geometry Errors

Every model must have a least one Ground block. Every Body block must have at least one Body CS, defined at the body’s center of gravity (CG). You must directly or indirectly define the Body coordinate systems (CSs) of a machine relative to a Ground or to World. You cannot enter cyclic Body CS definitions. For example, defining CS3 relative to CS2, defining CS2 relative to CS1, then defining CS1 relative to CS3, results in a definition both cyclic and missing any



reference to a Ground or World. You could break the cycle by referencing CS1 to a Ground or to World.

To be rendered in visualization, a Body must be connected to at least one Joint that is connected to the rest of the machine. You cannot visualize with equivalent ellipsoids a body whose principal inertial moments do not satisfy the *triangle inequalities*. (See “Rendering Body Shapes in SimMechanics” on page 6-5.)

## Joint Geometry Errors

The geometric configuration of joints, constraints, and drivers can conflict with assembly requirements and restrictions on certain blocks.

### Assembly Violated

Assembled joints must satisfy assembly tolerances on their connected Body CSs at all times. Disassembled joints assembled at machine initialization must also satisfy assembly tolerances during the simulation.

### Zero Massless Connector Distance

The initial distance between two Body CS origins connected by a massless connector must be nonzero. The massless connector holds the distance between two Body CS origins constant during motion.

### Composite Joints: Restrictions Among Primitives

Certain composite Joint blocks place restrictions on their primitive joint axes. For example, Bearing must have its prismatic axis P1 aligned to its third revolute axis R3.

## Block Connection and Topology Errors

General rules on how to connect SimMechanics blocks are discussed in the “Modeling Mechanical Systems” chapter. In particular, consult that chapter’s sections:

- “Modeling Machines” on page 4-2
- “Checking Model Validity” on page 4-64

Some restrictions are properties of individual blocks, as explained in their reference pages. See the “SimMechanics Block Reference.”

### **Motion Singularity and Inconsistency Errors**

Motion simulation errors often occur because of *singularities* or dividing by very small numbers. SimMechanics can integrate certain singularities, at a cost (see “Choosing an ODE Solver” on page 5-2).

Inconsistencies in motion arise from misapplication of constraints, drivers, and actuators and from conflicting stiction requirements.

### **Zero Masses and Moments of Inertia**

A body moving on a prismatic axis must have nonzero mass if you actuate it with forces. A body rotating about a revolute axis or pivoting about a spherical must have nonzero inertial moments about the axis or pivot if you actuate it with torques. If you want a massless rigid body, consider using a Massless Connector from the Joints/Massless Connectors sublibrary.

---

**Note** You can use point bodies (nonzero mass but zero moments) in SimMechanics, if the connected revolute axes and spherical pivots are dislocated from the body. Although the moments are zero about a point body’s CG, the displacement of the body from the axis or pivot shifts the moments from zero to nonzero values.

---

### **Alignment of Distinct Primitives**

Within a single Joint block, two distinct prismatic axes or two distinct revolute axes should never align during the simulation. If either occurs, a translational or rotational DoF is lost, and SimMechanics cannot determine the subsequent motion. An example of primitive axis alignment singularity is “gimbal lock.” Two of the three revolute primitive axes in the Gimbal block become parallel, reducing the number of independent DoFs in the Joint from three to two.

### **No Degrees of Freedom**

Your machine cannot move if it has no degrees of freedom. Each Constraint, Driver, and motion-actuating Actuator block you add to a model reduces the number of independent DoFs. (See “Counting Degrees of Freedom” on page 4-67.) Cure such errors by removing one or more of these blocks from your model, until you have at least one independent DoF.

## Redundant Constraints

Some constraints can restrict what another constraint is already restricting. Fix these errors by identifying and removing the redundancies.

## Violated Constraints

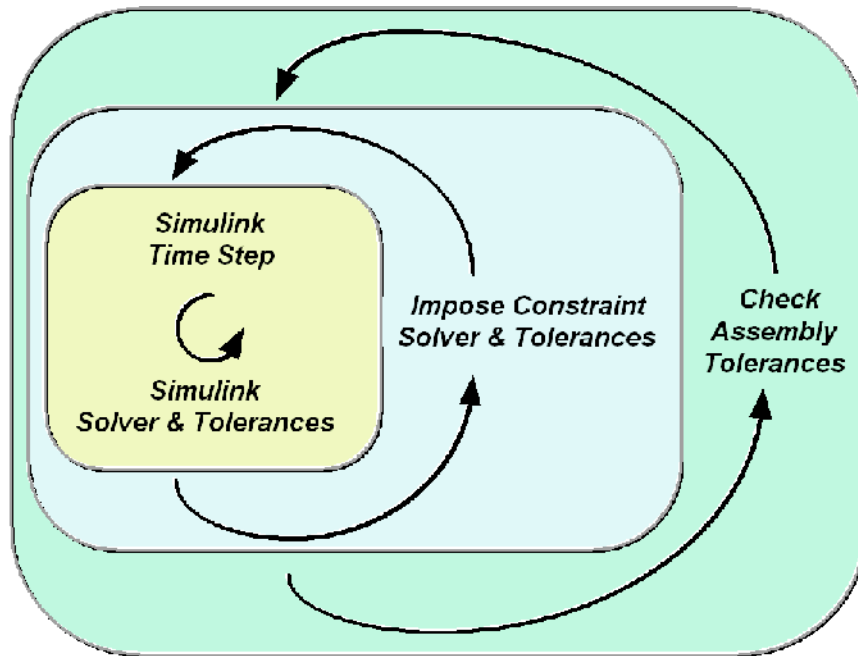
Some machine motions or simulations might not be able to maintain assembly tolerances at a particular simulation step while simultaneously satisfying the constraints. One or more joints become disassembled, followed by an error.

You can correct this situation in several ways. First, identify the joint, constraint, or driver causing the error and examine its physical configuration when the error occurs to isolate the conflict. Then try any combination of these steps:

- Decrease the ODE solver tolerances.
- Switch to a more robust ODE solver.
- Decrease the constraint solver tolerances.
- Switch to the machine precision constraint solver.
- Increase the assembly tolerances.

See “Choosing an ODE Solver” on page 5-2, “Choosing a Constraint Solver” on page 5-5, and “Constraints Pane” on page 5-9.

SimMechanics tries to harmonize your choices of ODE solver and solver tolerances, constraint solver and tolerances, and assembly tolerances in this dynamic hierarchy:



### Conflicting Actuators

You cannot put more than one actuator on a joint primitive.

---

**Note** You can simultaneously place an initial condition actuator and a force/torque actuator on a joint primitive.

The Joint Stiction Actuator block does accept an input signal for nonfrictional forces/torques, which the block adds to the stiction.

---

### Sticky Joints in Conflict

If your machine has two or more stiction-actuated (“sticky”) joints, a conflict among them can put SimMechanics into an infinite loop and prevent determination of the machine motion. Or one locked joint can prevent the other joints, sticky or not, from moving. The machine stops moving.

For example, one sticky joint becomes unlocked and requires the other to lock, which then requires the first to lock.

Remove these conflicts by removing one or more stiction actuators or by changing the Joint Stiction Actuator locking thresholds.

## **Analysis Mode Errors**

Certain restrictions apply to the analysis modes presented in “Choosing an Analysis Mode” on page 5-4. Consult individual analysis modes for more:

- “How SimMechanics Works” on page 5-14
- “Trimming SimMechanics Models” on page 7-12
- “Linearizing SimMechanics Models” on page 7-23

## Generating Code

You can use SimMechanics together with Real-Time Workshop to generate C code from your mechanical models and enhance simulation speed and portability. Certain features of Simulink itself make use of generated or external code. Some SimMechanics features are restricted when you translate a model into code. See “Limitations” on page 5-25.

---

**Note** Code generated from SimMechanics models is intended for rapid prototyping and hardware-in-the-loop applications. Production and embedded use of the generated code is not recommended.

---

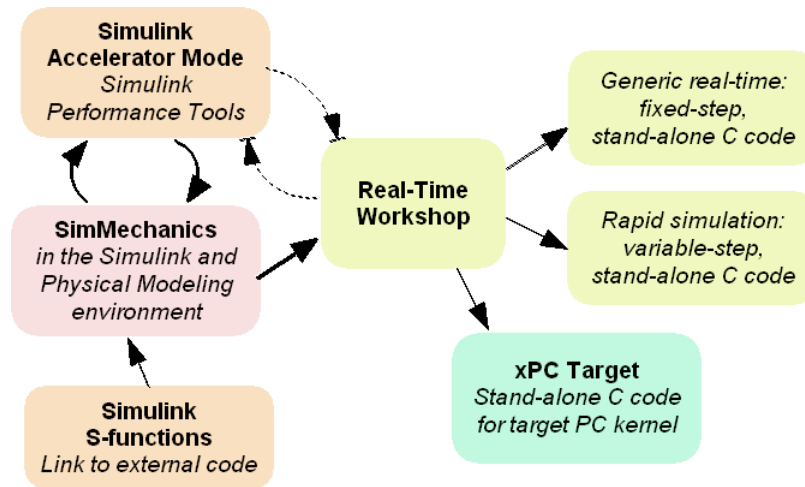
## Using Code-Related Products and Features

Simulink, SimMechanics, Real-Time Workshop, and xPC Target, using five code-related technologies, enable you to link existing code to your models and generate code versions of your models.

<b>Code-Related Task</b>	<b>Component or Feature</b>
Link existing code written in C or other supported languages to Simulink models	Simulink S-functions to generate customized blocks
Speeding up Simulink simulations	Simulink Accelerator mode*
Generate stand-alone fixed-step C code from Simulink models	Real-Time Workshop Generic Real-Time Target (GRT)
Generate stand-alone variable-step C code from Simulink models	Real-Time Workshop Rapid Simulation Target (RSIM)
Convert Simulink model to C code and compile and run it on a target PC	Real-Time Workshop and xPC Target

\*Accelerator mode speeds up simulations within Simulink by using selected Real-Time Workshop components. You do not need to have Real-Time Workshop installed to take advantage of this feature.

The following diagram summarizes the product and feature flow of generating code from models and linking existing code to models.



### Related Simulink Code Generation Documentation

Consult the documentation for Real-Time Workshop, xPC Target, and Simulink's Accelerator mode and S-function feature for general information on installing and using these products. See “Limitations” on page 5-25 for important restrictions when using SimMechanics with these features.

## How SimMechanics Code Generation Differs from Simulink

In general, using the code generated from SimMechanics models is similar to using code generated from normal Simulink models. There are some minor differences.

### SimMechanics and Normal Simulink Code Generated Separately

Real-Time Workshop generates code from the SimMechanics blocks separately from the normal Simulink blocks in your model. The generated SimMechanics code does not pass through `model.rtw` or the Target Language Compiler<sup>®</sup>. All the code generated from a single model ends up in the same directory, however.

### Locating and Changing Run-Time Parameters

As with code generated from any Simulink model, you can change the nontunable run-time parameters in the global parameter data structure named `rtP` in the model's generated C source files. You can then recompile the source without having to regenerate the code from your SimMechanics model. You can also use these parameters with target-specific run-time parameter utilities such as `rsimgetrtP`. Run-time parameters for each machine are pooled into a single vector in the `rtP` structure.

In addition to the usual C and header files, SimMechanics generates special files, `rt_mechanism_data.h` and `rt_mechanism_data.c`, containing separate data structures and functions for the run-time parameters. A special data structure for each machine explicitly associates each parameter with a specific SimMechanics block. One special function unpacks a machine's pooled run-time parameters from `rtP` into the special header file. The other special function repacks the parameters.

### SimMechanics Code Generated From Reusable Subsystems

Reusable subsystems in Simulink reuse code that is generated once from the subsystem. Code generated from SimMechanics models placed in reusable subsystems is rebuilt each time the subsystem is called, rather than being generated once and reused.



## Limitations

Some Simulink features and tools do not work with models containing SimMechanics blocks. Others work with SimMechanics models but only on the normal Simulink blocks in those models.

### Changing Block Properties

You cannot change the block properties of SimMechanics blocks at the command line.

### Unsupported Simulink Tools

The following tools do not work with SimMechanics models.

- External mode
- Profiler

### Simulink Tools Working Only With Normal Simulink Blocks

Some Simulink tools and features do not work with SimMechanics blocks.

- Execution order tags do not appear on SimMechanics blocks.
- SimMechanics blocks do not invoke user-defined callbacks.
- You cannot tune SimMechanics block parameters.
- You cannot set breakpoints on SimMechanics blocks.
- Reusable subsystems cannot contain SimMechanics blocks.
- You cannot use the Fixed-Point Settings interface with SimMechanics blocks.
- The Report Generator does not generate information on SimMechanics blocks.
- The S-Function Target tool cannot generate an S-function from a subsystem containing SimMechanics blocks.

### Restrictions with Generated Code

Stiction implemented with Joint Stiction Actuator blocks requires algebraic loops iterated at one time step to detect discrete events. In generated code versions of models with stiction, the iteration to find locking and unlocking of joints occurs over multiple time steps.



# Visualizing and Animating Machines

---

You can visualize your machine's bodies in SimMechanics with two different tools. One tool is based on Handle Graphics, a built-in feature of MATLAB. The other uses an internal SimMechanics interface to the optional Virtual Reality Toolbox. You can also construct your own virtual worlds and interface them to SimMechanics.

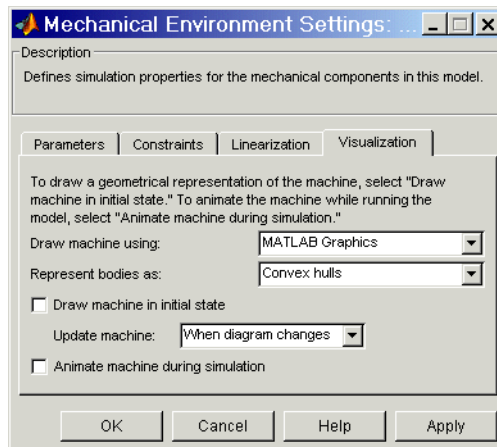
Choosing Visualization Options in SimMechanics (p. 6-2)	Different ways that SimMechanics renders and animates component rigid bodies of machines
Rendering Body Shapes in SimMechanics (p. 6-5)	Understanding and choosing body shapes for machine components visualizations
Viewing Machines with Handle Graphics (p. 6-10)	Viewing machine bodies in SimMechanics with MATLAB Handle Graphics
Viewing Machines in Virtual Reality (p. 6-24)	Viewing machine bodies in SimMechanics with Virtual Reality Toolbox
Creating Custom Machine Visualizations (p. 6-30)	Building an interface from SimMechanics to an externally rendered virtual reality machine

## Choosing Visualization Options in SimMechanics

You need to make three choices to set up SimMechanics visualization:

- Visualize the machine's initial state and/or animate its motion
- Pick which kind of body shape for the machine components you want, convex hulls or equivalent ellipsoids
- Render the scene with MATLAB Handle Graphics or Virtual Reality Toolbox

You implement these choices via the **Mechanical Environment Settings** dialog box. From the Simulink menu bar, select **Simulation**, then **Mechanical environment**. The **Mechanical Environment Settings** dialog opens. Click the **Visualization** tab. The **Visualization** pane appears below the tab. Keep this dialog open so you can see the visualization options. See “Mechanical Environment Settings Dialog Box” on page 5-7 for more information.



The sections of this chapter guide you to making the appropriate visualization choices within SimMechanics. The present section explains the why and how of visualizing the machine's initial state and animating its motion:

- “Uses of the SimMechanics Visualization Tools” on page 6-3
- “Setting Up Static and Dynamic Visualization” on page 6-4

You can find more on the body shapes in the next section:

- “Rendering Body Shapes in SimMechanics” on page 6-5

These two sections spell out the procedures for setting up the two SimMechanics visualization tools:

- “Viewing Machines with Handle Graphics” on page 6-10
- “Viewing Machines in Virtual Reality” on page 6-24

You can bypass the built-in visualization tools of SimMechanics by creating a virtual reality world of your own design to represent your machine’s bodies. With Virtual Reality Toolbox, you build a custom interface from your model to the virtual world and animate its virtual bodies:

- “Creating Custom Machine Visualizations” on page 6-30

## Uses of the SimMechanics Visualization Tools

The visualization tools of SimMechanics serve two distinct purposes, static and dynamic visualization. In both cases, you can change your observer viewpoint and navigate through the scene. You can change the body properties of the visualization only by changing the corresponding Body blocks in your model. Changing a body’s mass, inertia tensor, and coordinate systems can change its visual rendering.

### Static Rendering

Static rendering of machines in their initial state, during construction. Either choice is valid:

- Open the visualization before or while you build your model. You render each body as you add it to your model. Having a visualization tool open during model building lets you keep track of your machine parts and how they are connected. You can see unphysical or mistaken constructions before you finish the model.
- Open the visualization after you finish the model. All the bodies in the model appear together.

### Dynamic Animation

Animation of machines while the SimMechanics model is running. Use this feature to watch the model’s dynamics in three dimensions and visualize

motions and relationships more easily than is possible with Scope blocks alone. Chapter 5, “Running Mechanical Models” presents the steps for running SimMechanics models.

### Setting Up Static and Dynamic Visualization

Begin configuring the **Visualization** pane of the **Mechanical Environment Settings** dialog box:

- 1 You can separately choose static and/or animated visualization:
  - Render the static machine in its initial state by selecting the **Draw machine in initial state** check box.
  - Animate the machine during simulation by selecting the **Animate machine during simulation** check box.
- 2 Choose the updating mode in the **Update machine** pull-down menu. The visualization tools reflect changes to the Simulink model by either immediate or delayed update:
  - The **When diagram changes** option updates the visualized machine immediately upon changes to the Simulink model. For complex models, frequent visualization updates can be slow.
  - The **Only at Update diagram** option ignores changes you make to the model until you choose **Update diagram** in the model window **Edit** menu. This option allows you to update the visualization in one step after making a large number of changes.
- 3 Implement your visualization choices at any time by clicking **Apply**. Clicking **OK** applies your choices and closes the dialog box. A visualization window opens.
- 4 Refresh the entire visual scene at any time by selecting **Update diagram** from the model window **Edit** menu.

## Rendering Body Shapes in SimMechanics

The visualization tools render the bodies in either of two shapes:

- Equivalent ellipsoid for each body, based on its mass properties and center of gravity (CG) position, explained in “Equivalent Ellipsoids” on page 6-5
- Convex hull for each body, based on its Body coordinate systems (CSs), explained in “Convex Hulls” on page 6-8

### Choosing the Body Shape

You choose the body rendering in the **Visualization** pane, with the **Represent bodies as** menu. Choose Equivalent ellipsoids or Convex hulls. The default is Convex hulls.

### Equivalent Ellipsoids

The inertia tensor  $\mathbf{I}$  of a rigid body is real and symmetric, so it has three real eigenvalues ( $I_1, I_2, I_3$ ) and three orthogonal eigenvectors. These eigenvectors are the principal axes of the body. In the coordinate system defined by those axes, the inertia tensor is diagonal. The *trace* of the inertia tensor,  $\text{Tr}(\mathbf{I}) = I_1 + I_2 + I_3$ , is the same in any coordinate system with its origin at the body’s center of gravity (CG).

Every rigid body has a unique equivalent ellipsoid, a homogeneous solid ellipsoid of the same inertia tensor. The ellipsoid surface is given by

$$\left(\frac{x}{a_x}\right)^2 + \left(\frac{y}{a_y}\right)^2 + \left(\frac{z}{a_z}\right)^2 = 1$$

The three parameters ( $a_x, a_y, a_z$ ) are the *generalized radii* of the ellipsoid. For axis  $i = 1, 2, 3$

$$a_i = \sqrt{5((\text{Tr}(\mathbf{I}) - 2I_i)/(2m))}$$

### Triangle Inequalities

The *principal moments* ( $I_1, I_2, I_3$ ) must satisfy the *triangle inequalities*:

$$I_2 + I_3 \geq I_1$$

$$I_3 + I_1 \geq I_2$$

$$I_1 + I_2 \geq I_3$$

Violation of the triangle inequality for  $I_i$  leads to an unphysical imaginary generalized radius  $a_i$ .

---

**Caution** Visualizing the equivalent ellipsoids of bodies whose principal moments do not satisfy the triangle inequalities leads to a SimMechanics error.

---

### Ellipsoids with Special Symmetry

In general, all three  $I_i$ ,  $i = 1, 2, 3$ , are unequal. Such a body is an *asymmetric top*. If two of the three  $I_i$  are equal (double degeneracy), the body is a *symmetric top*. The third axis is the axis of symmetry. If all three  $I_i$  are equal (triple degeneracy), the body is a *spherical top* and dynamically equivalent to a homogeneous sphere.

### Reduced-dimension Ellipsoids

In special cases, the equivalent ellipsoid reduces to a planar, linear, or point figure.

Let  $(i, j, k)$  label the three axes  $(1, 2, 3) = (x, y, z)$  in any order.

- For a true *ellipsoid*, with nonzero volume, all the  $a_i$  are nonzero. The triangle inequalities are strict inequalities in this case:

$$I_j + I_k > I_i$$

$$I_k + I_i > I_j$$

$$I_i + I_j > I_k$$



- For an *ellipse*, with zero volume but nonzero area, one  $a_i = 0$  and the other two  $a_j, a_k$  are nonzero. One of the triangle inequalities becomes an equality:

$$\begin{aligned} I_j + I_k &= I_i \\ I_k + I_i &> I_j \\ I_i + I_j &> I_k \end{aligned}$$

- For a *line*, with zero volume and area but nonzero length, two  $a_i, a_j = 0$  and the third  $a_k$  is nonzero. Two of the triangle inequalities become equalities:

$$\begin{aligned} I_j + I_k &= I_i \\ I_k + I_i &= I_j \\ I_i + I_j &> I_k \end{aligned}$$

Equivalently,  $I_i = I_j$  are nonzero and  $I_k = 0$ .

- For a *point*, with no spatial extent, all three  $a_i$  vanish. All three triangle inequalities become equalities:

$$\begin{aligned} I_j + I_k &= I_i \\ I_k + I_i &= I_j \\ I_i + I_j &= I_k \end{aligned}$$

Equivalently, all three  $I_i$  vanish.

### Example: Simple Pendulum Rod

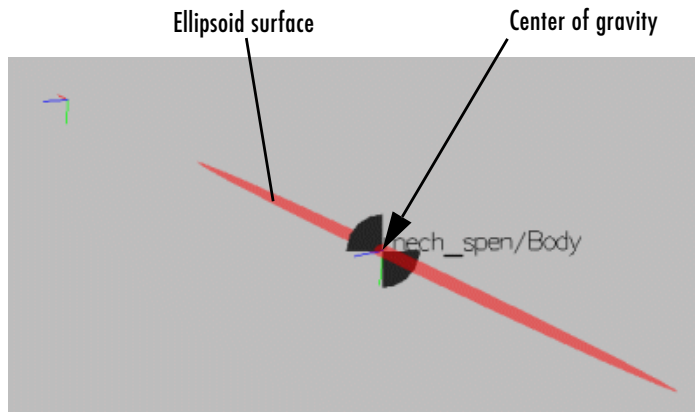
Consider the simple pendulum rod in “Visualizing a Simple Pendulum” on page 2-30. You can open the model by entering `mech_spen` at the command line.

The rod length  $L = 1$  m, and its radius  $r = 1$  cm. The inertia tensor is

$$\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} \frac{mr^2}{2} & 0 & 0 \\ 0 & \frac{mL^2}{12} & 0 \\ 0 & 0 & \frac{mL^2}{12} \end{bmatrix}$$

Because the rod has an axis of symmetry, the  $x$ -axis in this case, two of its three principal moments are equal:  $I_{yy} = I_{zz}$ , and two of its three generalized radii are equal:  $a_y = a_z$ . The rod is a symmetric top and, since  $r$  is much smaller than  $L$ , its equivalent ellipsoid is almost a line of zero volume and area.

The generalized radii of the equivalent ellipsoid are  $a_x = \sqrt{5/3}(L/2) = 0.646$  m and  $a_y = a_z = \sqrt{5}(r/2) = 1.12$  cm. This is the rod rendered in virtual reality:



## Convex Hulls

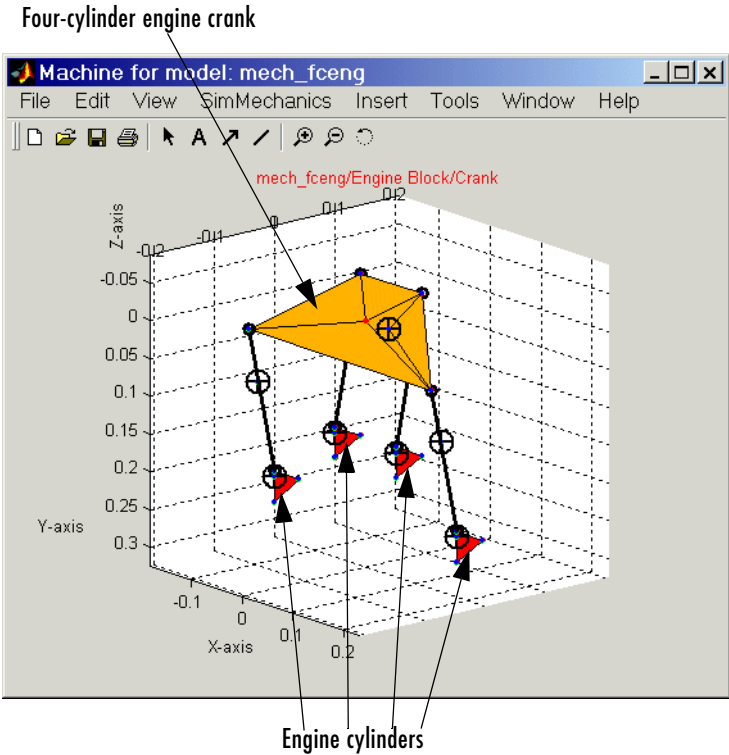
Every Body has at least one Body coordinate system (CS) at the CG. A Body also has one or more extra Body CSs for the attached Joints, as well as possible Actuators and Sensors. Each Body CS has an origin point, and the collection of all these points, in general, defines a volume in space. The minimum outward-bending surface enclosing such a volume is the convex hull of the Body CSs, and this is the alternative way SimMechanics has to render a body.

To enclose a nonzero volume, the set must have at least four non-coplanar Body CSs. Three non-collinear Body CSs are rendered instead by a triangle, and two non-coincident origins by a line. One is displayed just as a point. (The minimum one Body CS would be just the CG CS.) Four or more coplanar origins are rendered by a triangle, three or more collinear origins are rendered by a line, and two or more coincident origins are rendered by a point.

**Example: Four Cylinder Engine Crank**

Refer to the four-cylinder engine model of the Demos library by entering `mech_fceng` at the command line.

Double-click the Engine Block subsystem and note the Crank block representing the engine crank. This Body block has six Body CSs. Visualize the engine as convex hulls with Handle Graphics. The large block in your visualization is this engine crank, and it encloses a nonzero volume.



**Four Cylinder Engine Example: Engine Crank Convex Hull (Yellow)**

## Viewing Machines with Handle Graphics

The Handle Graphics feature for viewing your machine is built into SimMechanics. It uses a special set of symbols to draw bodies and Body coordinate systems (CSs). The Handle Graphics window also features special SimMechanics controls and tools.

The following sections explain the use of Handle Graphics visualization in SimMechanics:

- “Using the Standard Handle Graphics Controls” on page 6-12
- “Interpreting the Special SimMechanics Symbols” on page 6-12
- “Using Special SimMechanics Functions” on page 6-13

---

**Note** The Handle Graphics visualization feature is based on the Handle Graphics tools of MATLAB. Refer to the MATLAB graphics documentation for a full discussion of Handle Graphics.

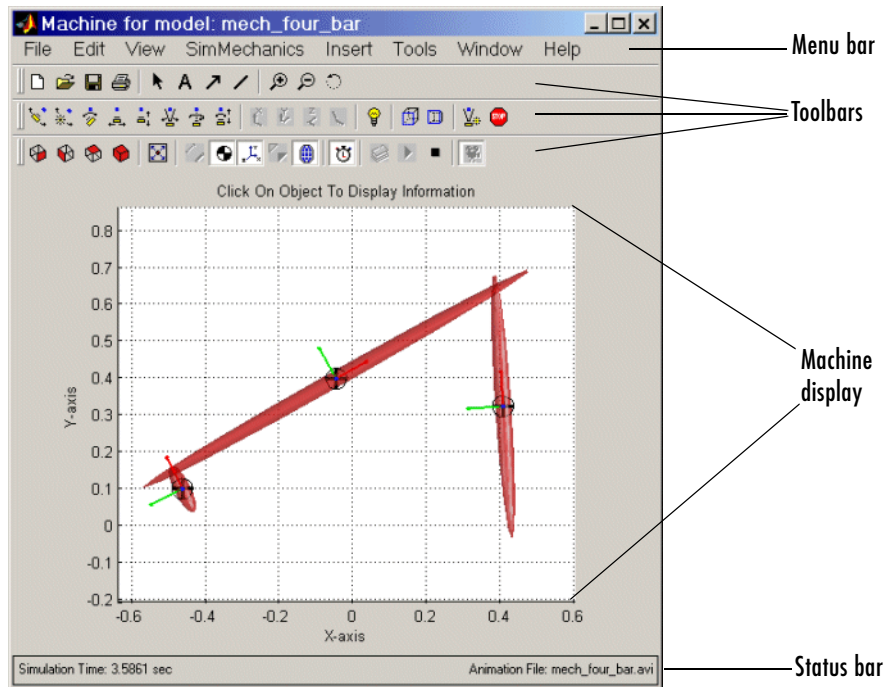
This section mainly focuses on features special to SimMechanics Handle Graphics visualization. Certain standard Handle Graphics features are disabled or missing in the SimMechanics window.

---

### Choosing Handle Graphics Visualization

You choose this visualization tool in the **Visualization** pane of the **Mechanical Environment Settings** dialog:

- 1** In the **Draw machine using** menu, choose MATLAB Graphics.
- 2** Select the **Draw machine in initial state** and/or **Animate machine during simulation** check boxes.
- 3** Click **Apply** or **OK**. A Handle Graphics window opens.



## Handle Graphics Window Displaying a SimMechanics Machine

### Changing the Machine Display Refresh Rate

By default, the machine display is updated at every major time step during simulation. You can change this refresh rate by reconfiguring the Simulink output options. Taking more sample points makes the animation smoother. Using fewer sample points leads to a more disjointed animation, but a faster simulation and a smaller recorded animation file (see “Recording and Playing Animations” on page 6-20).

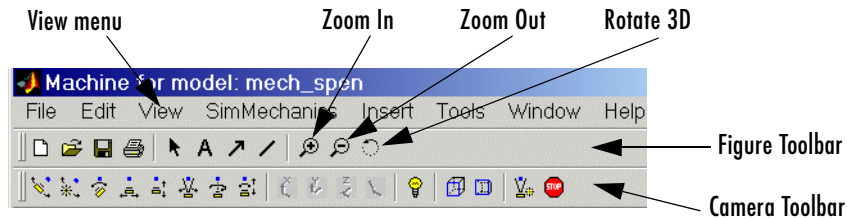
Open **Simulation parameters** in the **Simulation** menu of your model window. On the first pane, **Solver**, under **Output options**, change the pull-down menu entry to **Produce specified output only**. You must now use the **Output times** field on the right to specify explicitly how often Simulink should capture the simulation output. In this field, enter a vector of sample times. The sample time range must be the same as or lie within the **Start time** and **Stop time** range in **Simulation time** at the top of the **Solver** pane.

If you want a uniform output sampling, use the `linspace` command to specify the time range and number of sample points: `linspace(start-time, end-time, number-of-points)`. *Example:* To sample 200 points from zero to 10 seconds, enter `linspace(0,10,200)` in the **Output times** field.

### Using the Standard Handle Graphics Controls

The main controls of the **Figure Toolbar** are standard on all MATLAB Handle Graphics windows. Refer to the MATLAB Graphics documentation to learn how to configure Handle Graphics windows and objects.

For SimMechanics model visualizations, the **Zoom in**, **Zoom out**, and **Rotate 3D** buttons are especially useful. Select the button and click in the figure area to activate the function. Click, hold, and roll the figure to rotate the machine in three dimensions.



### Handle Graphics Window Viewing Controls

The useful **Camera Toolbar**, also discussed in the MATLAB Graphics documentation, is enabled by default in SimMechanics. You can enable or disable it, as well as the **Figure Toolbar**, from the **View** menu.

### Interpreting the Special SimMechanics Symbols

When SimMechanics calls the internal Handle Graphics window, it uses special conventions to render the bodies of your machine. You can control some of these conventions through the special **SimMechanics** menu (see “Changing How the Machine Is Displayed” on page 6-16).

The machine bodies are represented by one of the two SimMechanics body shapes (see “Rendering Body Shapes in SimMechanics” on page 6-5):

- The window displays all the machine's bodies. If a body has one of these associated surfaces, the surface is shaded red:
  - Equivalent ellipsoid
  - Convex hull of one or more surface patches (a triangle of coplanar points or an enclosing surface of four or more points)

A line convex hull is a black stick figure.

- Surfaces used in marking out enclosed convex hull volumes or coplanar triangles are tiled with patches.

The rendering uses two special symbols:

- The center of gravity (CG) point of each body is marked by a circle-plus symbol  $\oplus$ .
- Each Body CS is marked by coordinate axis triads. The color coding is X-Y-Z axes = RGB = red-green-blue.



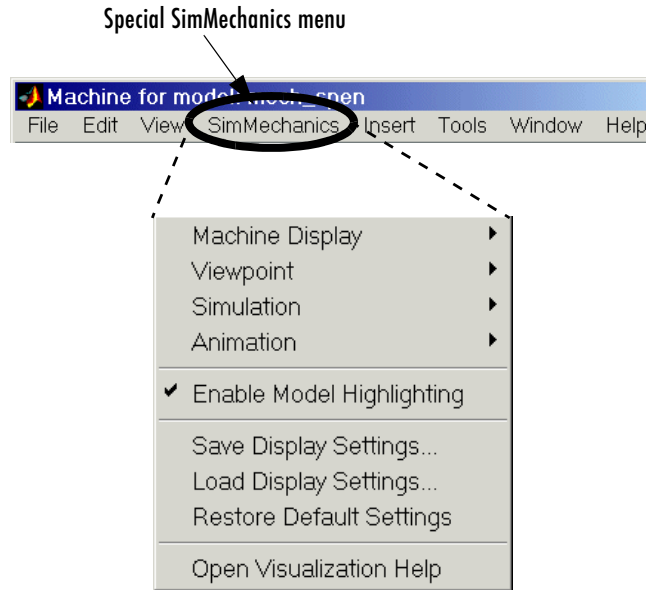
## Using Special SimMechanics Functions

You can perform a large number of tasks with the special **SimMechanics** menu, submenus, and toolbar:

- “Highlighting Bodies and Body Blocks” on page 6-15
- “Changing How the Machine Is Displayed” on page 6-16
- “Changing Perspective and Window Size” on page 6-19
- “Controlling the Simulation from the Window” on page 6-20
- “Recording and Playing Animations” on page 6-20
- “Saving and Recalling Display Settings” on page 6-22

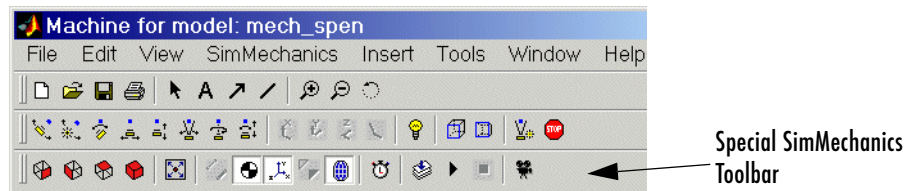
Once you open a Handle Graphics window, you have two ways to control the machine display and implement these tasks:

- Use the special **SimMechanics** menu in the middle of the menu bar. The menu contains special items and submenus.



### SimMechanics Menu for Handle Graphics Visualization

- Use the buttons in the special SimMechanics toolbar. Every feature on this toolbar occurs in the special menu, although the reverse is not true. See the “Summary of the SimMechanics Toolbar” on page 6-22.



### SimMechanics Toolbar for Handle Graphics Visualization

Certain options in the **SimMechanics** menu are enable/disable-selected. By selecting the item, you either enable it and a check mark appears, or you disable it and the check mark disappears.



Other menu items, when you select them, instead trigger an immediate action. These are the top-level items in the special **SimMechanics** menu.

### **SimMechanics Menu Items of the Handle Graphics Window**

<b>Menu Item or Group</b>	<b>Function</b>	<b>Action or Default</b>
<b>Machine Display</b>	Control machine rendering	Submenu
<b>Viewpoint</b>	Control perspective and window and axes size	Submenu
<b>Simulation</b>	Control model simulation	Submenu
<b>Animation</b>	Record simulation animation	Submenu
<b>Enable Model Highlighting</b>	Body-Body block highlighting	Enabled
<b>Display Settings</b>	Save, load, and restore SimMechanics display settings	Immediate action
<b>Open Visualization Help</b>	Open Help browser to this section of SimMechanics documentation	Immediate action

## **Highlighting Bodies and Body Blocks**

Clicking a body in the Handle Graphics window causes the following:

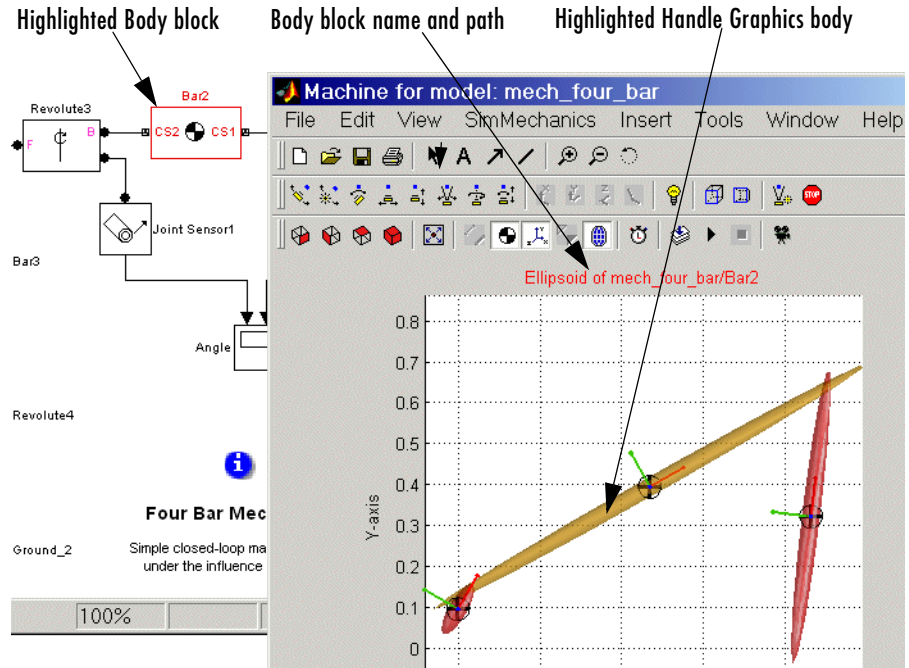
- The rendered body surface changes color from red to yellow.
- The Handle Graphics window displays the associated Body block name and its path on the line just above the machine display.
- The model window comes back to focus with the associated Body block highlighted in red.

To unhighlight a body, click anywhere in the white area of the machine display.

You can enable or disable the model Body block highlighting from the **SimMechanics** menu item **Enable Model Highlighting**. The default is enabled. If you disable model highlighting, the associated Body block is not

highlighted when you click the rendered body. But whether model highlighting is enabled or disabled, clicking a body in the Handle Graphics window always highlights the body in yellow.

The mech\_four\_bar model from the Demos library is shown with Bar2 (the middle bar) highlighted.



**Handle Graphics Window with Body and Body Block Highlighting**

## Changing How the Machine Is Displayed

The window displays different symbols to represent the machine. (See “Interpreting the Special SimMechanics Symbols” on page 6-12.) You can enable or disable these symbols from the **Machine Display** submenu. Some options are grayed depending on which one of the two body renderings you choose, equivalent ellipsoids or convex hulls (see “Rendering Body Shapes in SimMechanics” on page 6-5).

## Changing the Machine Display with Equivalent Ellipsoids

In the equivalent ellipsoid rendering, the **Machine Display** submenu has three active items:

<b>Centers of Gravity (CGs)</b>	Center of gravity of each body
<b>Coordinate Systems (CSs)</b>	Body coordinate systems
<b>Ellipsoids</b>	Machine bodies

All three are enabled by default. You can disable or reenable any one or more of these by selecting the items. When an item is disabled, the corresponding symbols immediately disappear from the machine display in the window. When an item is reenabled, the corresponding symbols immediately reappear.

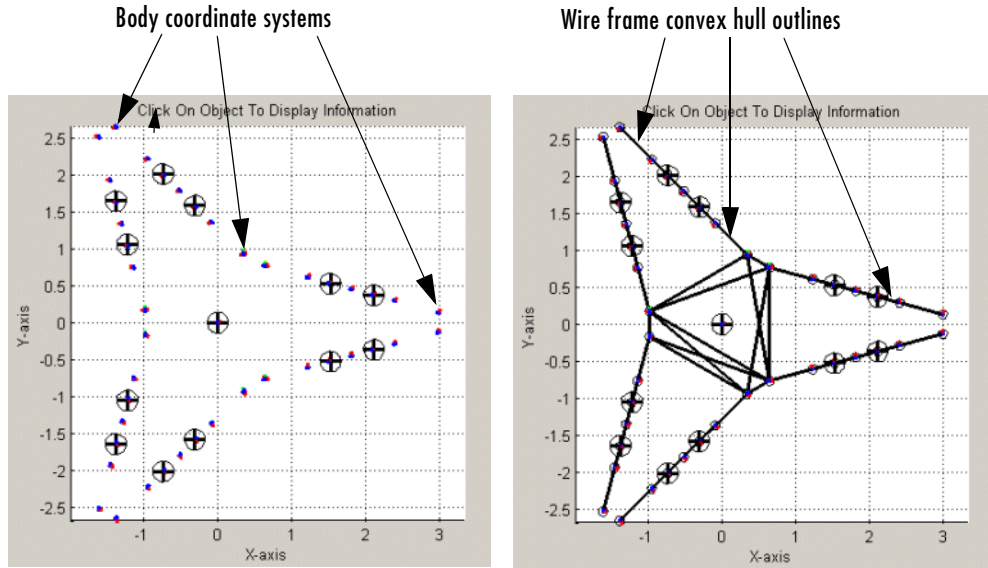
## Changing the Machine Display with Convex Hulls

In the convex hull rendering, the **Machine Display** submenu has four active items:

<b>Wire Frames</b>	Outline convex hull of each body
<b>Centers of Gravity (CGs)</b>	Center of gravity of each body
<b>Coordinate Systems (CSs)</b>	Body coordinate systems
<b>Patch Surfaces</b>	Fill body surfaces in red

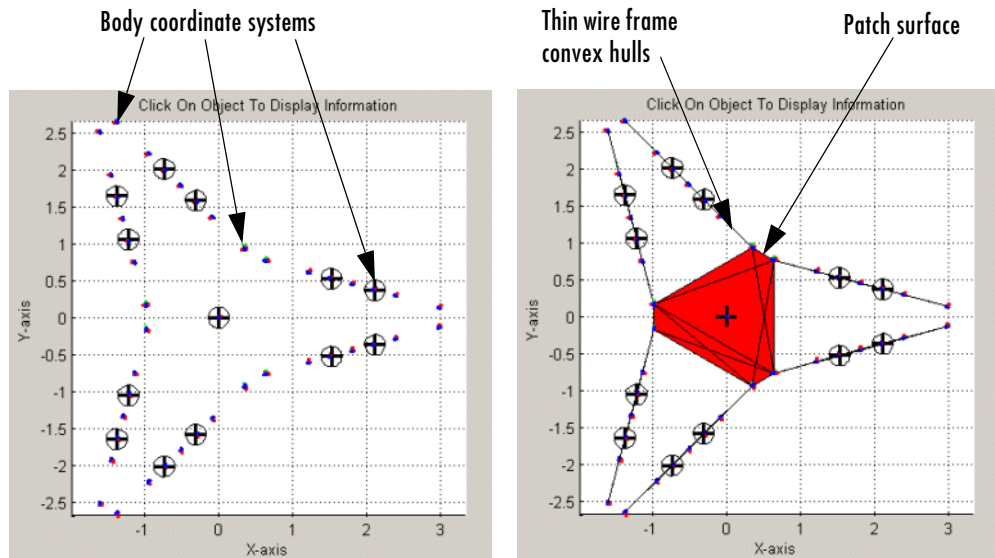
The **Centers of Gravity (CGs)** and **Coordinate Systems (CSs)** items are enabled by default. These items display the same symbols as they do for the equivalent ellipsoids.

The **Wire Frame** and **Patch Surface** items are also enabled by default. The wire frame option outlines with thick black lines the convex hull of each body. The Stewart platform of `mech_stewart_trajectory` demo illustrates the wire frame rendering:



**Stewart Platform Without (L) and With (R) Wire Frame Convex Hull Outlines**

The patch surface option renders the convex hulls as thin black lines and fills body surfaces in red. A linear 1-D body is rendered as a black stick figure. The mech\_stewart\_trajectory model renders the top plate of the Stewart platform as a 2-D planar figure with red fill and the legs as stick figures:



**Stewart Platform Without (L) and With (R) Patch Surface**

## Changing Perspective and Window Size

You can change your perspective view of the machine and the size of the window with the **Viewpoint** submenu. Selecting a viewpoint immediately changes your view in the window.

The available automatic perspectives are **X-Y Plane**, **Y-Z Plane**, **X-Z Plane**, and the trimetric **3-D** ( `view([1 -1 1])` ). The coordinate axes in SimMechanics are

- +x (rightward)
- +y (upward)
- +z (out of the screen)

A planar view projects the machine onto the selected plane. The trimetric view displays the machine from the viewpoint along the axis  $(1, -1, 1)$ . You can find any perspective you want manually by using the **Rotate 3D** feature of the **Figure Toolbar** or by rotating the camera from the **Camera Toolbar**. See “Using the Standard Handle Graphics Controls” on page 6-12.

**Fit Machine to View** immediately resizes the display window and the machine to fit together. It does not change the display perspective.

**Enable Automatic Axis Resize** forces SimMechanics to resize the axes and window to fit the machine motion during an animation. If this option is disabled, the machine's bodies can move out of view of the display. The default is enabled.

### Controlling the Simulation from the Window

Certain Simulink model functions are mapped into the **SimMechanics** menu so you can use them from the Handle Graphics window. You find these functions in the **Simulation** submenu.

You can start a model (if it is not running) and stop a model (if it is running) from this submenu by selecting **Start** or **Stop**. The keyboard shortcut **Ctrl+T** initiates the same actions, if the Handle Graphics or the Simulink model window is in focus.

You can update your Simulink model diagram by selecting **Update Simulink Diagram** or entering **Ctrl+D** at the keyboard with the Handle Graphics or Simulink model window in focus.

You can also enable or disable the display of the simulation running time by selecting **Display Simulation Time** in this submenu. The default is disabled. The simulation running time is shown in the left corner of the status bar at the bottom of the Handle Graphics window as **Simulation Time: ... sec.**

### Recording and Playing Animations

The Handle Graphics visualization tool allows you to record animations of your machine simulations. The animations are stored in Audio Video Interleave (AVI) format. You control animation recording through the **Animation** submenu.

To activate animation recording, enable **Store in AVI File**. The default is disabled. By default, recorded AVI files are placed in your current MATLAB directory. (To see your current directory, check the **Current Directory** browser or **Current Directory** field on the MATLAB desktop, or enter `pwd` at the command line.) If the name of your Simulink model file is `model-name.mdl`, the name of the recorded AVI file is `model-name.avi`. When you enable animation recording, this name appears in the right corner of the status bar at the bottom of the Handle Graphics window as **Animation File: model-name.avi**.

SimMechanics first records a MATLAB movie by capturing the machine display at every major simulation or output time step. Then, in the termination phase of your simulation, it converts this movie to AVI format and stores it in the AVI file. A small **AVI Conversion** window opens to indicate that conversion and storage are complete and to display the path of the AVI file. Click **OK** to close this prompt.

---

**Caution** SimMechanics overwrites any existing file with the same name as the AVI file in the same directory. The AVI file write fails if a file of the same name that is locked by another application exists in the same directory.

---

### Changing the Storage Directory of the Animation File

If you want to change the directory in which your AVI files are stored, you must implement the change *before* beginning the simulation. Change the AVI storage directory by selecting **Choose AVI File Location**. The **AVI File Location** browser appears. Change to the directory you want the AVI file stored in, then click **Save**.

### Compressing the Animation File

You can reduce the size of your AVI file by compressing it. Select **Compress AVI File** to enable this feature. The default is enabled.

---

**Note** AVI compression is available only on the Windows platform. SimMechanics uses the Indeo 5 compression algorithm, and your AVI player must be configured to decompress it.

---

### Playing Back the Animation File

You need an AVI-compatible video player to view the recorded file. You can use the internal MATLAB movie viewer or an external video player.

If you open an AVI file from MATLAB, an **Import Wizard** prompts you to load the AVI stream. Click the **Play Movie** button. The **MATLAB Movie Viewer** opens and runs the animation.

## Saving and Recalling Display Settings

As you work with the special **SimMechanics** Handle Graphics menu, you might want to save the setting changes that you make. The main menu has three functions that allow you to save, load, and reset display settings.

**Save Display Settings** saves the current configuration of SimMechanics Handle Graphics settings. The settings are stored in a MAT file. By default, this MAT file is placed in your current MATLAB directory. (To see your current directory, check the **Current Directory** browser or **Current Directory** field on the MATLAB desktop, or enter `pwd` at the command line.) If the name of your model is `model-name.mdl`, the default settings file is `model-name.mat`. When you save the settings, the **Save Display Settings** browser allows you to change the directory and name of the file. You can save multiple display settings MAT files under different names. Using an existing MAT file name overwrites the existing MAT file.

The name of the *last* saved MAT file is stored in the Simulink MDL file itself. (You must save your MDL file in order to save this name.) If you close Handle Graphics or close the entire MDL file, SimMechanics automatically loads this last saved MAT file of display settings when you restart Handle Graphics visualization. If you want to load a different settings MAT file, select **Load Display Settings** and choose a file in the **Load Display Settings** browser.

If at any time you want to revert to the default display settings documented in this section, select **Restore Default Settings**.

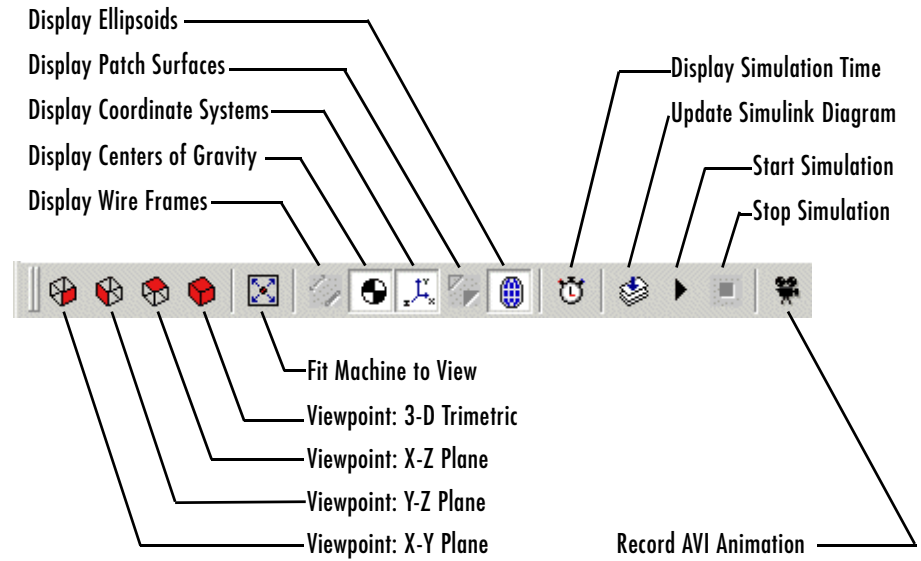
If you do not choose to store your settings before you close Handle Graphics visualization, your settings are not saved. Handle Graphics visualization, upon restarting, reverts to the default settings if the MDL file lacks a name of a display settings MAT file to load.

## Summary of the SimMechanics Toolbar

You can activate many of the special **SimMechanics** menu functions by pressing buttons on the special SimMechanics toolbar. The setting changes initiated by the toolbar are the same as the corresponding menu actions: either you enable or disable a feature, or you initiate an immediate action.

Hovering your mouse cursor over a toolbar button displays the button's tooltip indicating its function. If a menu function is grayed, depending on your choice of equivalent ellipsoid or convex hull rendering, the corresponding button is also grayed.





### Special SimMechanics Toolbar Functions (Equivalent Ellipsoid Case)

# Viewing Machines in Virtual Reality

The virtual reality feature for viewing your machine requires the separate Virtual Reality Toolbox. It too symbolizes bodies and Body coordinate systems (CSs) in a special way. Use of the virtual reality visualization is presented in this section, in “Interpreting Special Virtual Reality Symbols” on page 6-25 and “Changing the Observer’s Viewpoint” on page 6-26.

---

**Note** This optional feature requires the Virtual Reality Toolbox to be installed on your MATLAB path. You can use the default toolbox viewer or the blaxxun Contact viewer (version 4.4) plug-in. Both ship with the toolbox. This user’s guide assumes that you are using the default Virtual Reality Toolbox viewer. The blaxxun viewer must be installed separately and works only on Windows platforms.

Refer to the Virtual Reality Toolbox documentation for full information on installing the toolbox and using the toolbox viewer. If the Virtual Reality Toolbox is not installed, the Virtual Reality Toolbox option in the **Draw machine using** menu is missing.

---

## Choosing the Virtual Reality Visualization Tool

SimMechanics has an invisible internal interface to the Virtual Reality Toolbox, activated in the **Visualization** pane of the **Mechanical Environment Settings** dialog as follows:

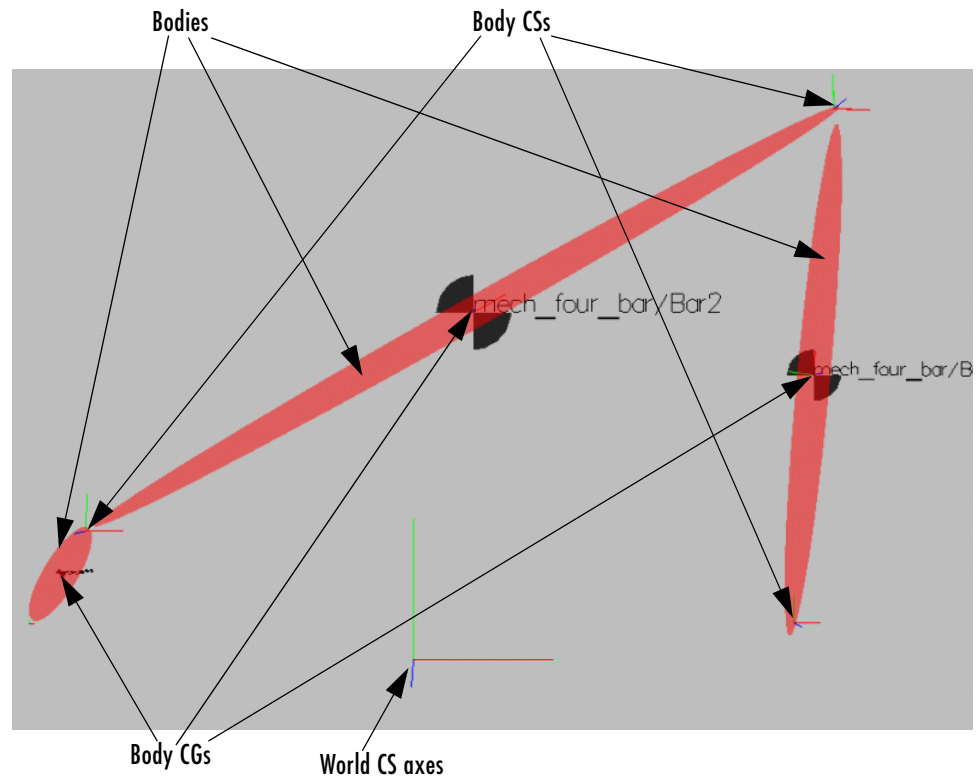
- 1 In the **Draw machine using** menu, choose Virtual Reality Toolbox.
- 2 Select the **Draw machine in initial state** and/or **Animate machine during simulation** check boxes.
- 3 Click **Apply** or **OK**. The viewer opens with an embedded virtual scene.

What you see in the virtual reality scene is a *virtual world* containing *virtual objects*, the bodies of your machine. The virtual scene displays the bodies along with their associated CSs.

## Interpreting Special Virtual Reality Symbols

Refer to the tutorial “A Four Bar Mechanism” on page 2-36. Open the `mech_fourbar` model from the Demos library and start the virtual reality visualization.


The machine bodies are rendered with special symbols, as shown in this virtual scene.



### Virtual Reality Scene: Four-Bar Mechanism

The major features of virtual reality visualization are as follows:

- The virtual scene displays all the machine’s bodies. If a body has an associated surface,
  - Equivalent ellipsoid

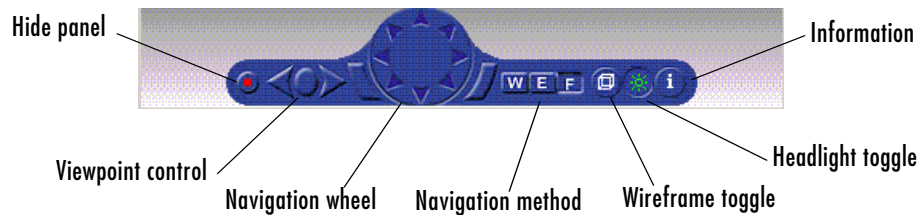
- Convex hull of one or more surface patches (a triangle of coplanar points or an enclosing surface of four or more points)  
the associated surface is red. A line convex hull is a black stick figure.
- The center of gravity (CG) point of each body is marked by a Body CG icon, .
- The World CS is marked by a large coordinate axis triad and Body CSs by small triads. The color coding is X-Y-Z axes = RGB = red-green-blue.

## Changing the Observer's Viewpoint

The Virtual Reality Toolbox contains a viewer as the default method for viewing virtual worlds. This section reviews the features and controls of the viewer.

The creation, configuration, and removal of virtual objects in the virtual world is entirely under the control of SimMechanics. You can add, change, or eliminate a virtual object only by making the corresponding change to the mechanical model in your Simulink model window.

The user's view of the virtual world, however, is under your control, through the viewer control panel at the bottom of the virtual scene.



### Virtual Reality Toolbox Viewer Control Panel

#### Viewpoint Control

There are three buttons on the control panel that affect the viewpoint of the scene. The center circular button resets the view to the current viewpoint. This button is useful when you have been moving about the scene and need to reorient yourself.

You can use the right and left arrows associated with viewpoint control to browse through predefined viewpoints. These buttons are inactive if other viewpoints are not specified by the author. You can also use the **Page Up** and **Page Down** keys to navigate through these viewpoints.

## Control Menu

Access the control menu by right-clicking in the viewer window. You can use the control menu to specify a predefined viewpoint or change the appearance of the control panel. You can also control the navigation method, speed, and rendering of the virtual world. For more information about navigation methods, see “Navigation” on page 6-28. For more information about rendering, see “Rendering” on page 6-28.

## Changing the Navigation Speed

- 1 In the viewer window, right-click.

A menu similar to this one appears.



- 2 Select **Navigation**.
- 3 Select **Speed**, then **Very Slow**.

Your navigation speed within the virtual world is much slower than before.

---

**Note** Your navigation speed controls the distance you move with each keystroke. It does not affect animation speed.

---

Set a higher speed for large machines and a slower speed for more controlled navigation in smaller machines.

### Rendering

You can change the rendering of the scene through the control panel or the control menu. The virtual scene is illuminated both by ambient lighting and a “headlight” shining directly ahead from your viewpoint. The most basic rendering operation is turning the camera headlight and the lighting of the scene on or off. When **Headlight** is off, the scene can appear dark.

The ambient lighting from all directions is controlled by **Lighting**. When **Lighting** is off, the virtual world appears as if lit in all directions. Shadows disappear and the scene loses some of its three-dimensional quality.

If **Transparency** is off, transparent objects are rendered as solid objects.

Turning **Wireframe** on changes the scene’s objects from solid to wireframe renderings.

### Navigation

You can navigate through the virtual scene with either the control panel, control menu, or your mouse.

**Control Panel.** The center navigation wheel and two curved buttons on either side move you about the scene. Experiment by moving backward and forward and side to side until you become comfortable with the controls.

**Control Menu.** Right click in the viewer window to access the control menu. Select **Navigation**, and the control menu appears.

From this menu, you can reset the viewpoint so that it is pointed straight ahead by choosing **Straighten up**. You can also return to the current viewpoint by choosing **Go to current viewpoint**.

**Mouse 3-D Roll.** You can execute a 3-D roll in your viewpoint at any time by clicking and holding anywhere in the virtual scene and then dragging the cursor. Your viewpoint rolls in the direction you drag the cursor.

### Learning More About Virtual Reality Toolbox

The best way to become comfortable moving around in a virtual world is to practice with a variety of movement modes and viewpoints. Consult the Virtual Reality Toolbox documentation for more details on controlling your virtual reality scene.

---

**Note to Users of Virtual Reality Toolbox** Using the internal SimMechanics virtual reality feature adds no new blocks to your model. SimMechanics feeds all virtual reality signals to the virtual scene internally, and the Virtual Reality Toolbox interface is not visible in the model window.

---

## Creating Custom Machine Visualizations

---

**Note** This section assumes that Virtual Reality Toolbox is installed on your system and that you are familiar with it and with creating virtual worlds. Refer to the Virtual Reality Toolbox User's Guide for full details on installing and using this toolbox.

---

You can bypass the visualization features built into SimMechanics and create a machine animation in a virtual world of your own design. This gives you the power to animate a more realistic visualization of your machine. You create a virtual world, populate it with bodies represented as virtual objects using the Virtual Reality Modeling Language (VRML), then interface the virtual world with your SimMechanics model. Creating your own virtual animation requires a new or existing virtual world for your model and an interface between them.

This section explains how to use a separately created virtual world with SimMechanics:

- “Creating Virtual Worlds for SimMechanics Models”
- “Interfacing SimMechanics with Virtual Worlds” on page 6-34

### Creating Virtual Worlds for SimMechanics Models

The Virtual Reality Toolbox User's Guide and VRML books such as Marrin and Campbell [8] explain how to create virtual objects and assemble them into virtual worlds. This section highlights the special requirements to make a virtual world usable as a visualization for a SimMechanics model.

As you create a virtual world populated by virtual bodies, you must create each component body, then plan and implement the geometry of the machine's initial state. Doing this yourself is a major difference from the SimMechanics virtual reality visualization, where the bodies are created, placed, and oriented automatically. Use a VRML authoring tool that can read VRML as a native format to create and edit virtual reality `.wr1` files. If you are familiar with raw VRML source code, you can use a plain text editor or the MATLAB editor to edit the files.

You represent each body by a virtual object encoded in a `.wr1` file. You also create a master `.wr1` file to represent the virtual world that refers to body `.wr1`



files, placing and orienting these body in the larger scene. You can define a body's position and orientation with respect to:

- The overall virtual world, corresponding to the SimMechanics coordinate system World
- Another body in the machine, corresponding to Body coordinate systems in SimMechanics

You can nest body references to other bodies in VRML hierarchies, but you must define at least one body's position and orientation with respect to the overall virtual world. Place and orient the bodies in their initial states, corresponding with the initial state of the SimMechanics simulation.


Each body's .wrl file contains a hierarchical tree starting with the Transform node. Among Transform's fields must be translation and rotation fields to specify the body's position and orientation in space. If a body is nested below another body, its position and orientation are defined with respect to the next body up the hierarchy.

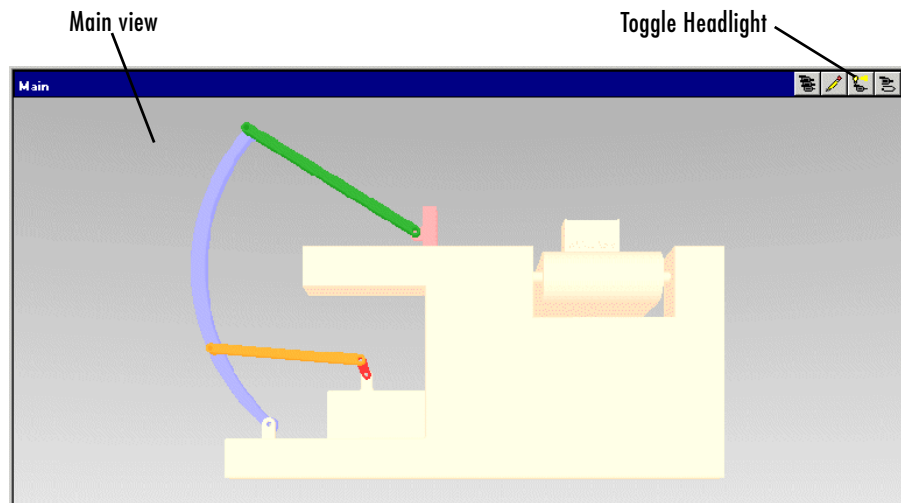
Creating your own virtual world gives you great flexibility in representing your machine:

- You can render bodies in as much or as little detail as you want, with shapes, colors, textures, etc., of your own choosing.
- You can include or omit bodies that you do not animate.
- You can create a computer-aided design (CAD) representation of your machine and export it into VRML files.
- If you only translate a body, you can omit the rotation field from its Transform node.
- If you only rotate a body, you can omit the translation field from its Transform node.

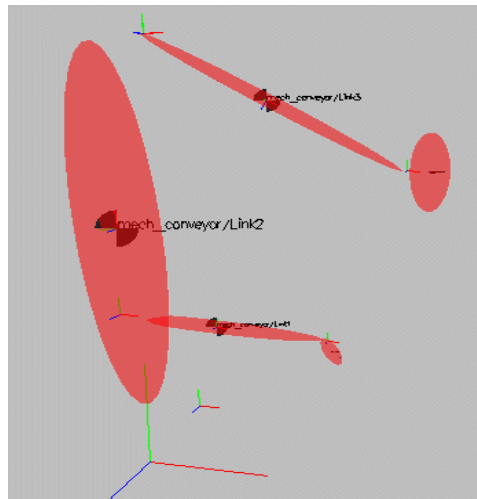
### **Example: Viewing Custom External VRML Files for the Conveyor Loader**

The demo model `mech_conveyor_vr` is a modified version of the original conveyor model `mech_conveyor` and comes with external VRML files containing static renderings of the machine parts in their initial positions. This example uses the V-Realm Builder<sup>®</sup> VRML authoring tool in the Virtual Reality Toolbox to view the files.

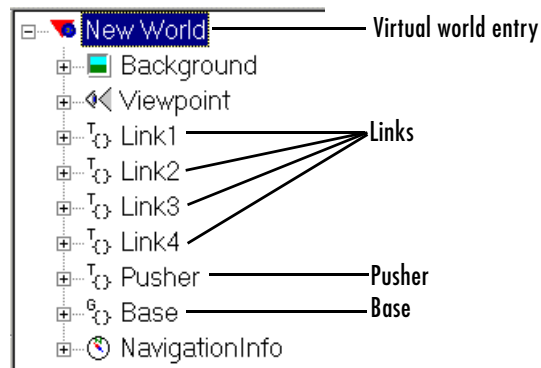
- 1 In the `<matlabroot>\toolbox\physmod\mech\mechdemos\` directory, open these external VRML files with V-Realm Builder:  
base.wrl, convmech.wrl, link1.wrl, link2.wrl, link3.wrl,  
link4.wrl, pusher.wrl
- 2 Click the **Test Mode** button  in the V-Realm Builder toolbar and view the complete machine in the **Main** view. Right-click in this window to configure the navigation. If the colors seem washed out, toggle off the headlight.



These conveyor parts are more realistic than the equivalent ellipsoids or convex hulls available with the built-in SimMechanics virtual reality tool:

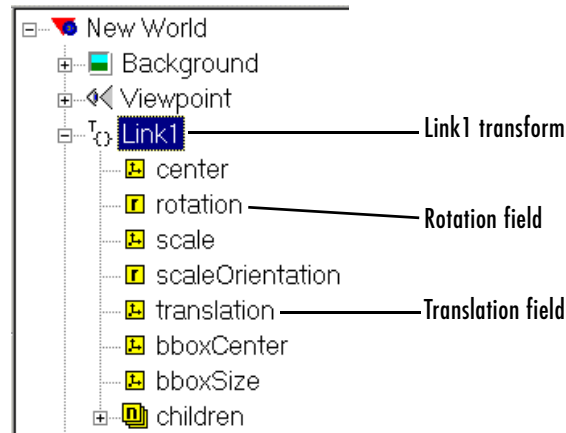


- 3 On the left side of your VRML editor window, examine the node tree of convmech.wr1 that refers to the six VRML files representing each component body:



The hierarchy of body position and orientation references is flat in this model. Each body is separately referenced to the top level of the hierarchy, New World.

- 4 Expand one of the nodes. Each body node has, among others, rotation and translation fields:



The exception is the Pusher transform, which has only a translation field. In the SimMechanics model, the pusher only translates along one axis.

## Interfacing SimMechanics with Virtual Worlds

To animate a body, you need to measure its motion in your SimMechanics simulation and export that information to the virtual world. This requires connecting Body Sensor blocks to the Bodies you want to animate in your model, then creating an interface that animates the virtual bodies with the body sensor motion signals. “Example: Interfacing the Conveyor Loader Model and Virtual World” on page 6-36 applies these steps to the mech\_conveyor\_vr demo.

### Adding and Configuring Body Sensors

Refer to “Sensing Body Motions” on page 4-60 for general information on how to use Body Sensors. Connect the Body Sensors to Body coordinate systems (CSs) on the bodies whose motions you want to animate. The Body block reference discusses how to create and configure Body CSs. You need to take these special steps to export the signals of a body sensor to your virtual world:

- 1 Make sure the Body Sensor's Body CS reference origin and orientation follow the body's defining VRML hierarchy.

*Example:* You define a new Body CS on a body to connect the Body Sensor. If you defined the VRML body's position with respect to the center of gravity (CG) of a second, neighboring body in your VRML files, you should set the **Translated from origin of** field of the new Body CS to the origin of the CG CS of the second body.

- 2 In the Body Sensor dialog, select the **[x; y; z] Position** check box if you want to animate the body's translational motion.

Select the **[3 x 3] Rotation matrix** check box if you want to animate the body's rotational motion.

- 3 Choose the coordinate system in which the body motions are measured in the **With respect to coordinate system** pull-down menu. You can pick *Absolute (World)* or *Local (Body CS)*. This coordinate system should be the same as the coordinate system used to define the body's position and orientation in the VRML files.

A Simulink output port  $\gt$  appears on the block for each of the motion signals. The translational signal is a 3-vector of spatial coordinates:  $(x, y, z)$ . The rotational signal is a 9-vector, column-wise representation of the 3-by-3 orthogonal rotation matrix  $R$ :  $(R_{11}, R_{21}, R_{31}, R_{12}, \dots)$ .

### Animating the Virtual World Bodies

Animating the virtual bodies requires interfacing the body sensor signals in the SimMechanics model with the VRML translation and/or rotation fields in the .wrl files. You accomplish this with the VR Sink block, which you can find in the Virtual Reality Toolbox block library. Enter

```
vrlib
```

at the command line. Drag a copy of the VR Sink block into your model.

Open the VR Sink dialog box. (The figure "Conveyor Loader Model: VR Sink Dialog Box" on page 6-39 displays an example of the dialog.) In the **Source file** field in the **World properties** area, enter the name of the VRML file that represents your model's virtual world. This is the file that refers to the other

.wrl files representing the component bodies of your machine. If the virtual world VRML file is not the same directory as your model, enter the file's path relative to the model. Click **Apply**.

In the **VRML tree** window, the node list of the virtual world .wrl file appears. Expand the tree of each component body in the list to view that body's check box list. Select the **rotation** and/or **translation** check boxes as needed for each body. Simulink input ports > appear on the block icon for each of these selected check boxes. The ports are labeled **node.field**. The **node** is the name for the body. The **field** is named either rotation or translation.

### Converting Body Sensor Signals into VRML Format

You are now ready to connect the Body Sensor output signals to the VR Sink block. But you might need to modify those signals for valid use in VRML.

- You can connect the translational motion signal line directly from the output port of the Body Sensor to the **node.translation** input port on the VR Sink. The VRML node tree directly accepts translation motion as a 3-vector signal of rectangular coordinates  $(x,y,z)$ .

Make sure that the translational motion signal refers to the same coordinate system used to define the body's position in the VRML files.

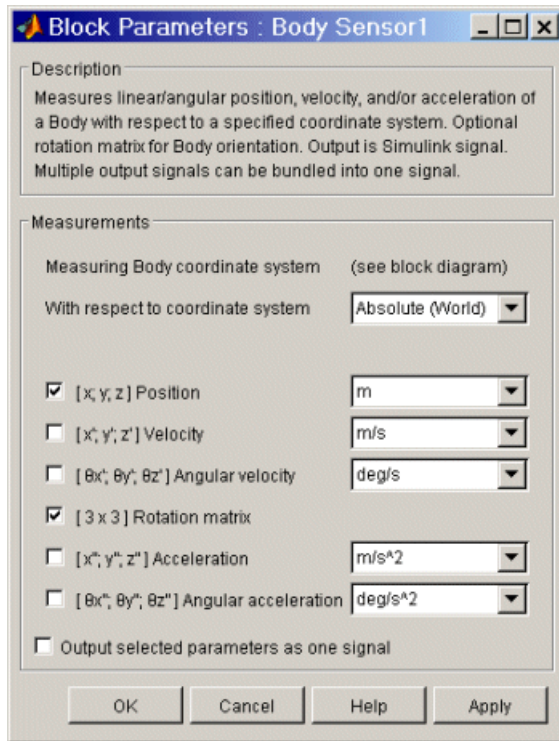
- You cannot directly connect the rotational motion signal line to the VR Sink. The Body Sensor output represents orientation with a 3-by-3 rotation matrix  $R$ , while VRML accepts orientation represented as the axis-angle 4-vector form  $[\mathbf{n} \theta]$ , where  $\mathbf{n} = (n_x, n_y, n_z)$  is a 3-vector representing the rotation axis and  $\theta$  is the rotation angle.

Open the SimMechanics Utilities library. For each rotational motion signal, drag a RotationMatrix2VR block into your model. Connect the rotation signal from the Body Sensor block to the RotationMatrix2VR block. Then connect the latter block to the corresponding **node.rotation** input port on VR Sink for that body. This block converts the 3-by-3  $R$  matrix signal into the 4-vector VRML form.

Close the VR Sink block dialog. Your SimMechanics model now animates the virtual world.

### Example: Interfacing the Conveyor Loader Model and Virtual World

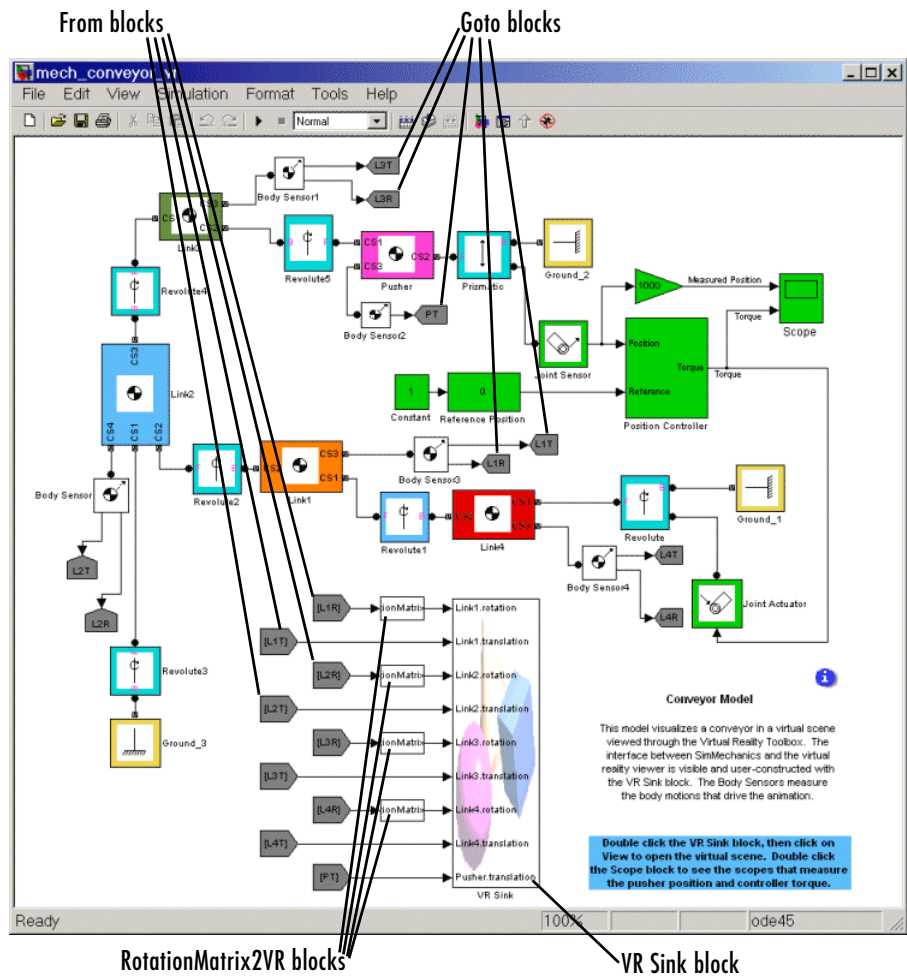
In the `mech_conveyor_vr` demo model, open the Body Sensor1 block. The block measures the translational and rotational motion of Link3 in the conveyor:



The Body Sensor1 block has two Simulink output signals.

The other Body Sensor blocks are similar, except for Body Sensor2, which measures only the translational motion of the pusher. All the Body Sensors measure body motions with respect to World, the frame in which the conveyor base is at rest. Each motion signal represents the body's displacement relative to its initial position.

The mech\_conveyor\_vr model contains a Virtual Reality Toolbox interface to the model's custom VRML files.

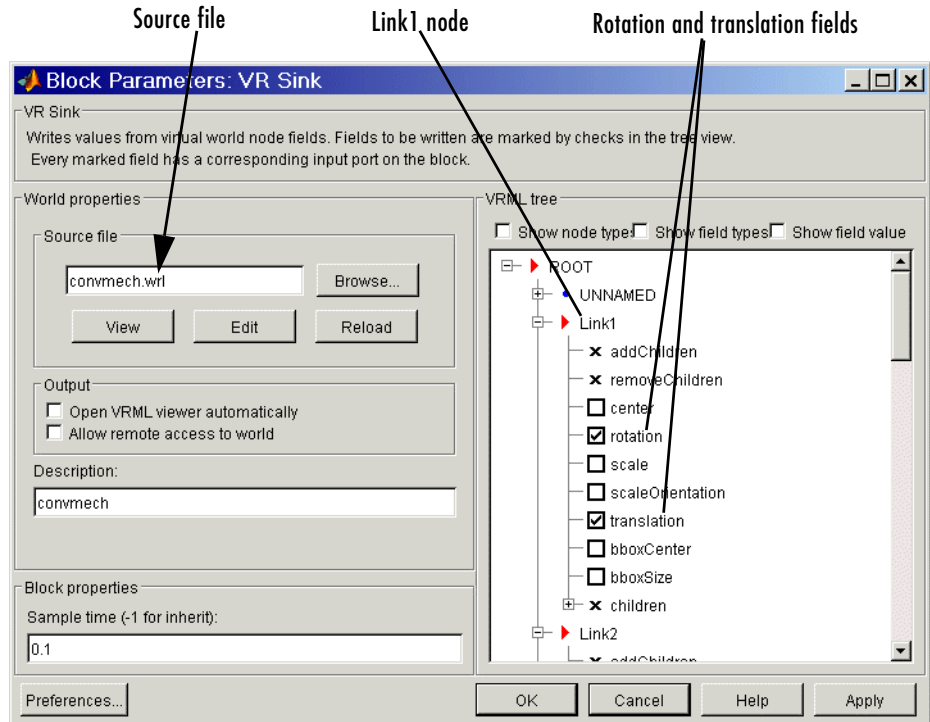


### Conveyor Loader Model with Custom Virtual Reality Interface

- 1 Trace each body sensor signal through the model. The signals are routed through pairs of Simulink Goto and From blocks.



- 2 Open the VR Sink block. The **Source file** is `convmech.wrl`, the master file for this virtual world. The **VRML tree** on the right reproduces the node tree visible in the VRML editor for `convmech.wrl`.



### Conveyor Loader Model: VR Sink Dialog Box

- 3 Expand and scroll down the VRML trees. The trees for Link1, Link2, Link3, Link4, and Pusher list the field inputs for accepting motion signals.
- The Link component bodies require both translational and rotational motions. All the Links have actively selected check boxes for their **rotation** and **translation** field inputs.

- The Pusher body requires only translational motion. Only the **translation** field check box is selected for the Pusher.

Each of the nine Simulink input ports on the VR Sink block is named **node.field**. The Base of the conveyor does not move, so its node has no motion input fields.

- 4 In the VR Sink dialog, click **View** in the **World properties / Source file** area.

Your Virtual Reality Toolbox viewer opens, displaying the conveyor machine scene. The scene is identical to that visible in the VRML editor (see “Example: Viewing Custom External VRML Files for the Conveyor Loader” on page 6-31).

- 5 Close all the dialog boxes by clicking **OK**, leaving the viewer open.
- 6 Click the **Start** button in the model window.

As in the original mech\_conveyor demo, starting the model opens the **Reference Position** slider bar that you can move from side to side. As you do so, watch the pusher in the viewer move in parallel.

# Case Studies

---

SimMechanics features important advanced modes for analyzing machine motion beyond Forward Dynamics integration of forces. This chapter explains how to specify machine motion, then deduce the necessary forces and torques, with the inverse dynamics and kinematic analysis modes. You can also analyze perturbations about a given machine trajectory by trimming and linearizing your model.

Finding Forces from Motions (p. 7-2)

Examples of the Inverse Dynamics and Kinematics analysis modes of SimMechanics

Trimming SimMechanics Models  
(p. 7-12)

Examples of finding machine steady states with the Trimming mode of SimMechanics

Linearizing SimMechanics Models  
(p. 7-23)

Determining the linear response of trimmed mechanical systems with SimMechanics and Simulink

## Finding Forces from Motions

The SimMechanics Kinematics and Inverse Dynamics modes (see “Choosing an Analysis Mode” on page 5-4) enable you to find all the forces on a closed-loop system or an open system, respectively, given a model that completely specifies the system’s motions. Because the model specifies the positions, velocities, and accelerations of the model’s components, these modes, unlike Forward Dynamics mode, do not need to compute these quantities. Consequently, Kinematics and Inverse Dynamics modes take less time to compute the forces on a system. The time saving depends on the size and complexity of the system being simulated.

To use these modes, you must first build a model of the system that specifies completely the positions, velocities, and accelerations of the system’s joints and bodies. Such a model is called a *kinematic* model. You create a kinematic model by creating and interconnecting blocks representing the bodies and joints of the system and then connecting actuators to the joints to specify the motions of the bodies.

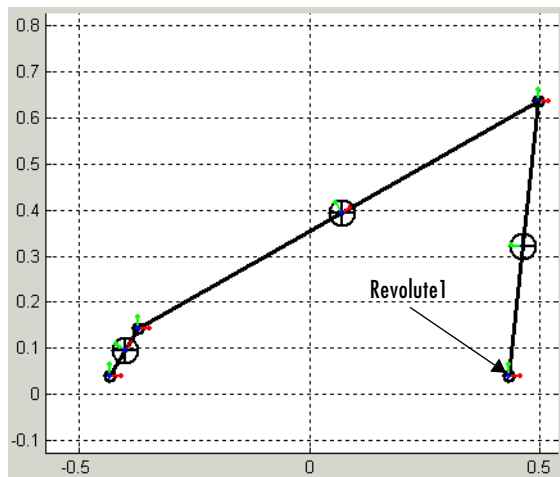
A model does not have to actuate every joint to specify completely the motions of a system. In fact, the model need actuate only as many joints as there are independent degrees of freedom in the system. (See “Counting Degrees of Freedom” on page 4-67.) For example, a model of a four-bar mechanism need actuate only one of the mechanism’s joints, because a four-bar mechanism has only one degree of freedom. To avoid overconstraining the model’s solution, the number of actuated joints should not exceed the number of degrees of freedom. Attempting to simulate an overconstrained model causes Simulink to halt the simulation and display an error message.

The following sections illustrate use of Kinematics and Inverse Dynamics modes to find the forces on the joints of a closed- and an open-loop system, respectively.

## Closed-Loop Example: Four-Bar System

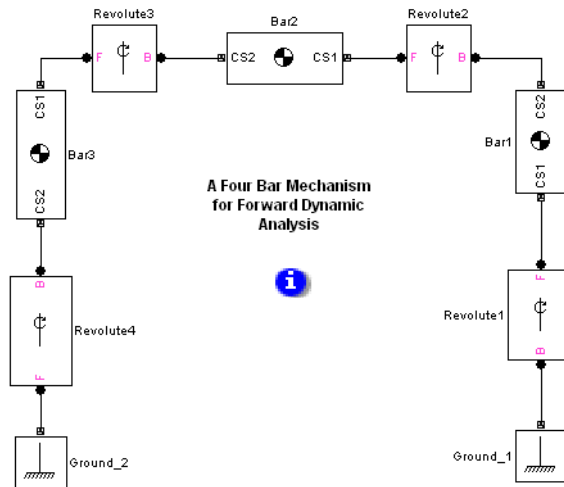
**Note** The Kinematics mode works only on closed topologies and requires motion-actuating every independent DoF (see “Counting Degrees of Freedom” on page 4-67). There must also be no Joint Stiction Actuators and no nonholonomic constraints.

Consider the four-bar system used to illustrate model building in the tutorial titled “A Four Bar Mechanism” on page 2-36.



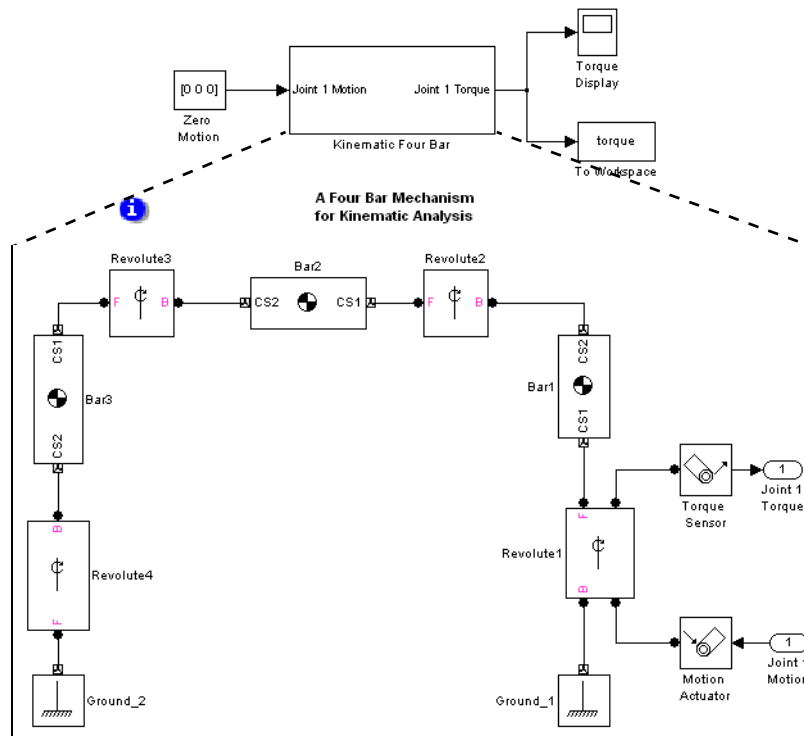
Suppose that you want to keep this system from collapsing under its own weight. Because this system has only one degree of freedom, applying a counterclockwise torque to the joint labeled Revolute1 would accomplish this objective. But how much torque is sufficient?

To answer this question, you must first build a kinematic model of the stationary four-bar system. The kinematic model must specify the structure of the four-bar system and its motion over time. The four-bar model from the four-bar tutorial specifies the structure of the system. You can therefore use the tutorial model as a starting point for creating the kinematic model. You can find this version of the tutorial in the model `mech_four_bar_forw`.



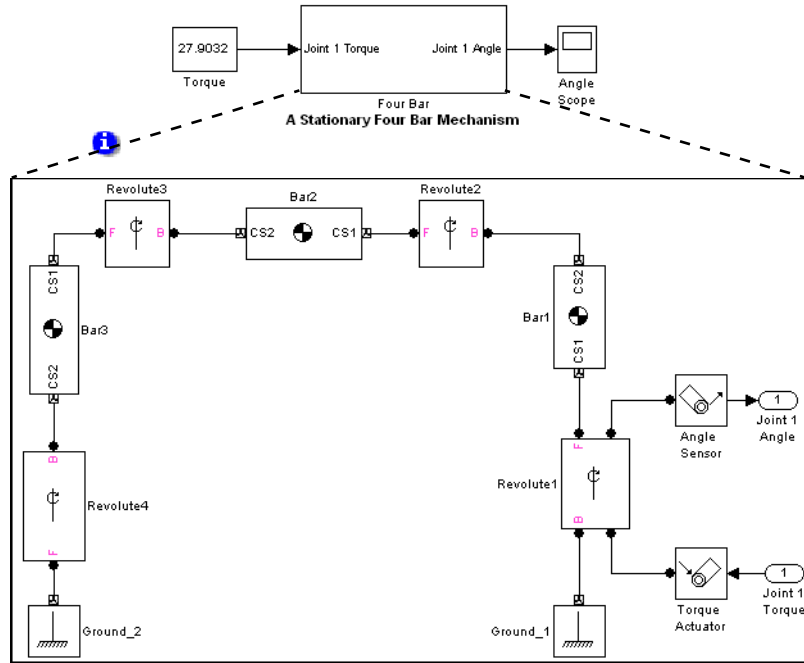
To turn the tutorial model into a kinematic model, you must specify how the system moves over time. In particular, you want the model to specify that the system remains stationary. Because a four-bar system has only one degree of freedom, you need specify only that one of the joints remains stationary. You can use a Joint Actuator to accomplish this task.

The following diagram shows a kinematic model derived from the tutorial model in this manner. This system is modeled in the mech\_four\_bar\_kin.

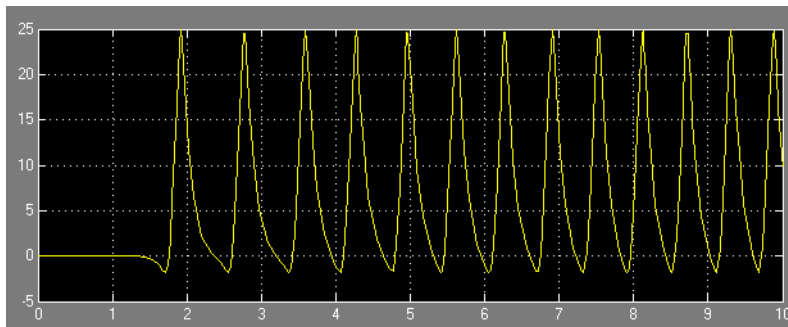


The model uses a Joint Actuator block driven by a Constant block to specify the motion on the Revolute1 joint. The Constant block outputs a three-element vector that specifies the angular position, velocity, and acceleration, respectively, of the joint as 0. The model uses a Joint Sensor block connected to a Scope block to display the resulting torque on the joint and a To Workspace block to save the torque signal to the MATLAB workspace. Running this model in Kinematics mode (see “Choosing an Analysis Mode” on page 5-4) reveals that the torque on the Revolute1 joint is 27.9032 n-m.

To verify that the computed torque is, indeed, the torque required to keep the system stationary, create a forward-dynamics model that applies the computed torque to the Revolute1 joint. This model is contained in `mech_four_bar_stat`.



Now run the model in Forward Dynamics mode, with the Revolute1 Angle Scope open.



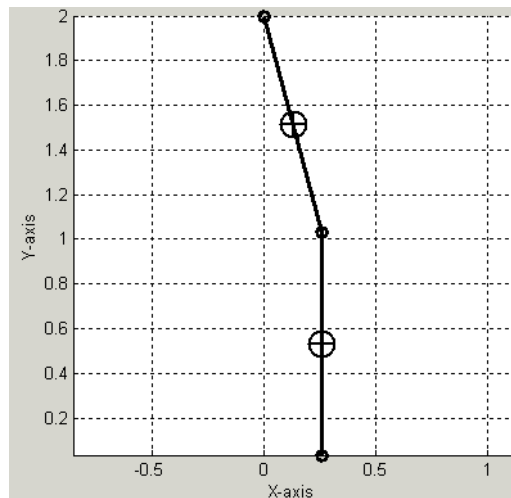
The Scope display reveals that the model does, indeed, remain stationary, although only for about 1.5 seconds. The model is nonlinear and unstable, and the computed force value is not copied exactly in the new model.



## Open-Topology Example: Double Pendulum

**Note** The Inverse Dynamics mode works only on open topologies and requires motion-actuating every independent DoF (see “Counting Degrees of Freedom” on page 4-67).

Consider a double pendulum consisting of two thin rods each one meter long and weighing one kilogram. Suppose that the upper rod is initially rotated 15 degrees from the perpendicular.

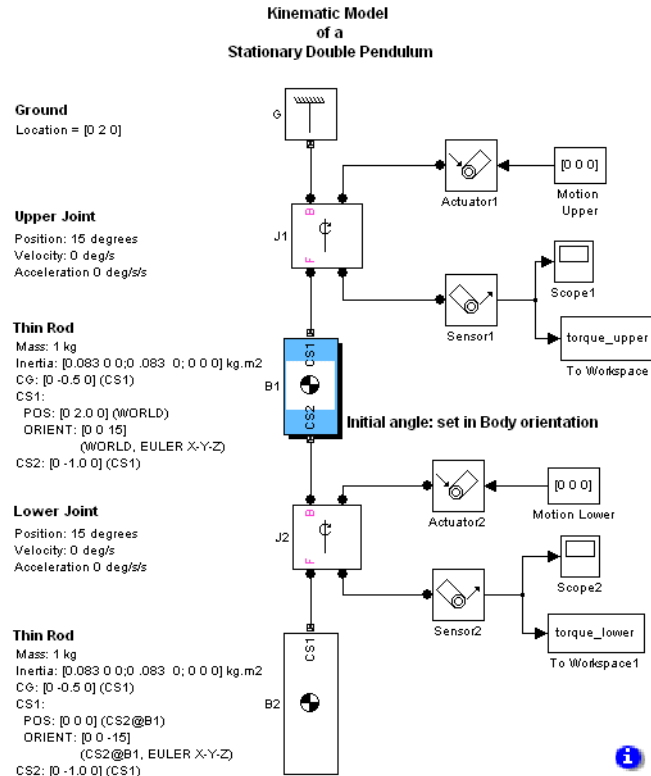


How much torque is required to keep the pendulum stationary? Solving this problem entails building a kinematic model of the stationary pendulum. The model must represent the geometry of the double pendulum and specify that it remains stationary throughout the simulation.

The kinematic model can take different approaches to specifying the initial state of the pendulum. One approach uses Body block parameters to specify the initial states. Another approach uses Actuator block parameters.

## Using Body Block Parameters to Specify Initial Conditions

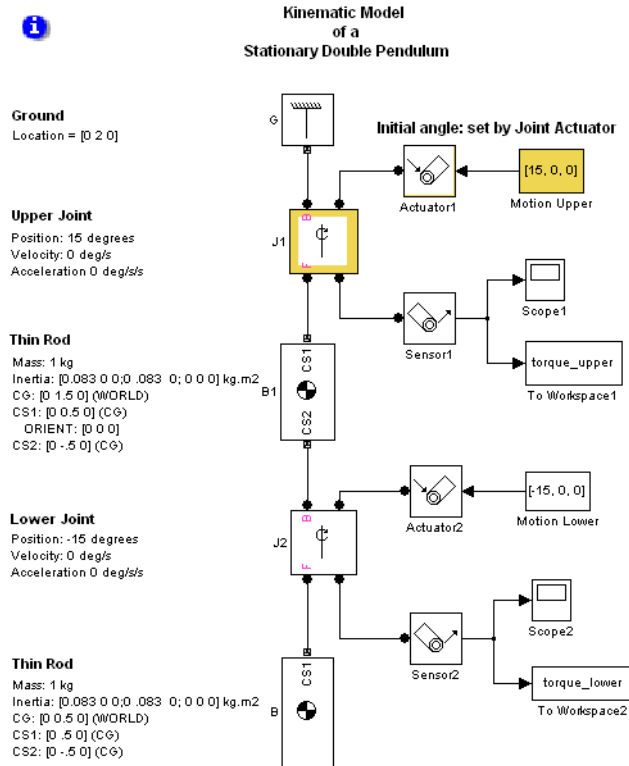
The following diagram illustrates the Body block approach to modeling initial states. The model is mech\_dpend\_invdyn1.



This model represents the pendulum by two Body blocks and two Revolute Joint blocks. The CS1 axis of the upper body (B1) of the pendulum is rotated 15 degrees from the perpendicular (see annotation for block B1). The coordinate systems for the lower block (B2) are aligned with CS1 of the upper block. The CS1 of B2 is rotated -15 degrees relative to CS1 of B1; i.e., it is perpendicular to the world coordinate system. Actuator blocks connected to the joint blocks specify that the pendulum should not move from its initial position. The model uses sensor blocks connected to To Workspace blocks to output the torques on the upper and lower joints as MATLAB workspace variables torque\_upper and torque\_lower, respectively.

## Using Actuator Blocks to Specify the Initial States

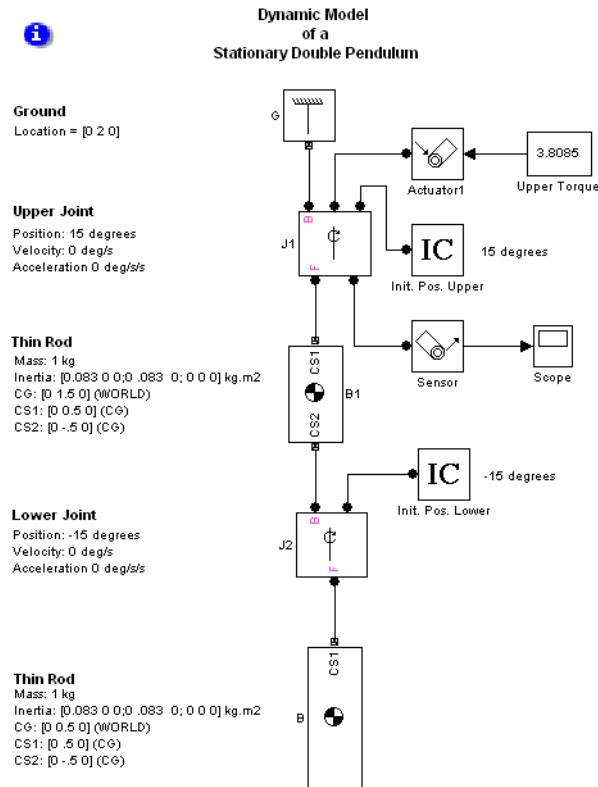
The following diagram shows the use of Actuator blocks to specify the initial kinematic state. The system is modeled in the `mech_epend_invdyn2`.



Using the actuators to specify the displacement slightly simplifies the configuration of the body blocks.

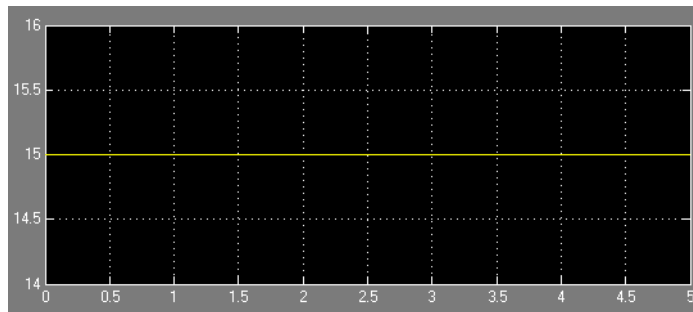
Simulating either model in Inverse Dynamics mode (required for open-loop models; see “Choosing an Analysis Mode” on page 5-4) causes Simulink to compute the joint torques required to maintain the pendulum in its initial position. The torques are 3.8085 and 0 newton-meters, respectively. You can verify that these are the correct answers by creating a version of the model that applies the computed torques to the joints and simulating the model in Forward Dynamics mode. For example, the following diagram illustrates a

forward dynamics version of the kinematic model that uses the joint actuators to specify the initial angular displacement of the pendulum bodies. Open the model mech\_dpend\_stat.



Note that this body uses Initial Condition blocks to specify the initial 15 degree displacement of the upper body from the vertical in the world coordinate system and the corresponding initial -15 degree displacement of the lower body from the vertical in the coordinate system of the upper body. The negative displacement of the lower body is equivalent to positioning it as vertical in the world coordinate system.

Simulating this model in Forward Dynamics mode results in the following display on the upper joint scope.



The scope reveals that the upper joint never moves from its initial 15 degree displacement, thus confirming that the computed torque is correct.

## Trimming SimMechanics Models

Trimming a mechanical system refers to the process of finding solutions for the model that satisfy conditions on its inputs, outputs, and states that you specify; for example, steady-state solutions where some or all of the derivatives of the system's states are zero. To use the Simulink `trim` command on a system represented by a SimMechanics model, you must select the SimMechanics Trimming mode (see “Trimming Mode” on page 5-4). You must also specify the initial states of the system and the conditions that the solutions must satisfy. The following examples illustrate the process of trimming unconstrained and constrained mechanical systems, respectively.

Consult the Simulink user's guide for more on trimming models.

---

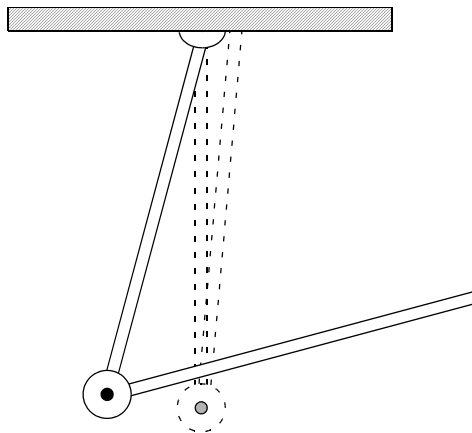
**Caution** You cannot use Driver or Joint Initial Condition Actuator blocks while trimming a model.

Always run a model you plan to trim at least once in Forward Dynamics first.

---

### Unconstrained Example: Spring-Loaded Double Pendulum

Consider the following spring-loaded double pendulum.

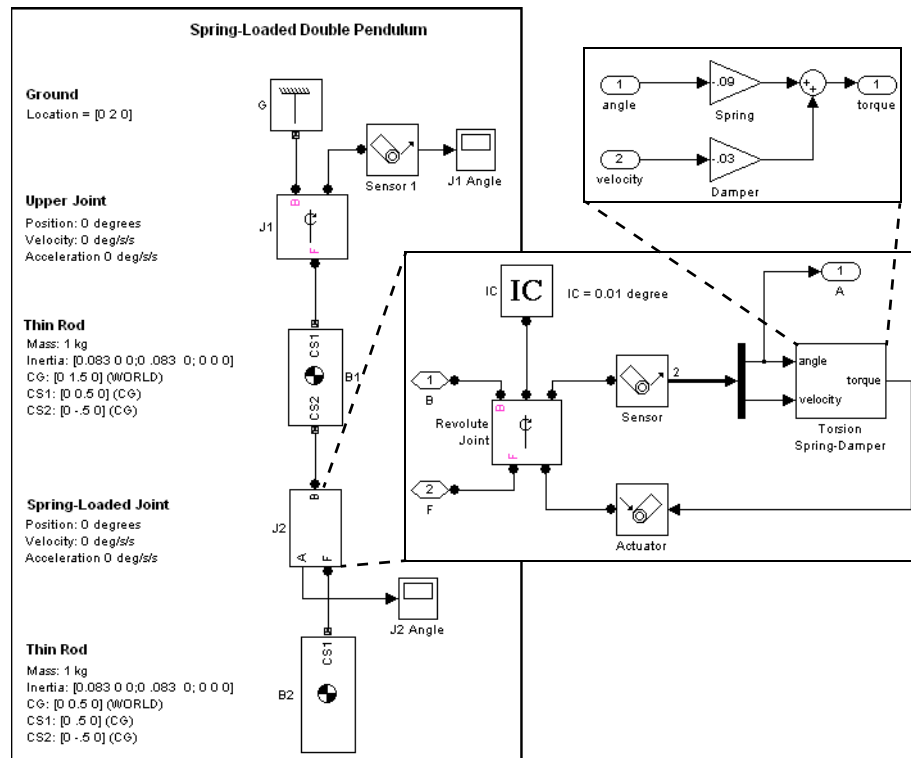


The joint connecting the upper and lower arms of this pendulum contains a torsional spring and damper system that exert a counterclockwise force directly proportional to the angular displacement and velocity of the joint, respectively. Suppose that the lower arm is folded upwards almost vertically and then allowed to fall under the force of gravity. At what point does the spring-damper system reach equilibrium; i.e., at what point does it cease to unfold?

To find an equilibrium point for this system:

- 1 Build and run a SimMechanics model of the system.

This diagram shows an example of such a model, `mech_depend_trim`.



This model uses Body blocks to model the upper and lower arms of the pendulum and a Revolute Joint block (J1) to model the connection between

the pendulum and ground. The model uses a Subsystem block (J2) to model the spring-loaded revolute joint between the arms. The subsystem in turn uses a negative feedback loop consisting of Revolute Joint, Joint Sensor, Joint Actuator, Gain, and Sum blocks to model the spring-loaded joint. The feedback loop models a torsional spring and damper by multiplying the angular displacement and velocity, respectively, of the joint by constants. The loop sums the resulting torques and feeds them back into the joint via the Joint Actuator block. The result is that the joint experiences a torque that opposes its motion and is proportional to its angular displacement and velocity.

The spring and damper constants used in this model were chosen by running the model with various candidate values and choosing the ones that resulted in a moderate deflection of the pendulum.

## 2 Determine the layout of the model's state vector.

You need to determine the layout of the model's state vector in order to tell the `trim` command where in the model's state space to start its search for the pendulum's equilibrium point (the point where it stops unfolding). Use the SimMechanics `mech_stateVectorMgr` command to perform this task. Refer to the Ground block, G.

```
>> v = mech_stateVectorMgr('mech_dpend_trim/G');
>> v.StateNames

ans =

    'mech_dpend_trim/J2/RevoluteJoint:R1:Position'
    'mech_dpend_trim/J1:R1:Position'
    'mech_dpend_trim/J2/RevoluteJoint:R1:Velocity'
    'mech_dpend_trim/J1:R1:Velocity'
```

The `StateNames` field of the state vector object returned by `mech_stateVectorMgr` lists the names of the model's states in the order in which they appear in the model's state vector. Thus the field reveals that the model's state vector has the following structure:

```
x(1) = position of lower joint (J2)
x(2) = position of upper joint (J1)
x(3) = velocity of lower joint (J2)
```



`x(4) = velocity of upper joint (J1)`

### 3 Determine an initial state vector.

The initial state vector specifies the point in a system's state space where the `trim` command starts its search for an equilibrium point. The `trim` command searches the state space outward from the starting point, returning the first equilibrium point that it encounters. Thus, the starting point should not be at or near any of a system's trivial equilibrium points. In the case of the double pendulum, the point `[0; 0; 0; 0]` (i.e., the pendulum initially folded up and stationary) is a trivial equilibrium point and therefore to be avoided. The initial state vector must also be a column vector and must specify angular states in radians.

Often, the choice of a good starting point can be found only by experiment, that is, by running the `trim` command repeatedly from different starting points to find a nontrivial equilibrium point. This is true of the double pendulum of this example. Experiment reveals that the following starting point

```
ix(1) = J2 (lower joint) angle = -35 degrees = -0.6109 radians
ix(2) = J1 (upper joint) angle = -10 degrees = -0.1745 radians
ix(3) = J2 angular velocity = 0 radians/sec
ix(4) = J1 angular velocity = 0 radians/sec
```

yields a nontrivial equilibrium point. Therefore you can save time by creating an initial state vector set to these values.

```
>> ix = [-0.6109; -0.1745; 0; 0];
```

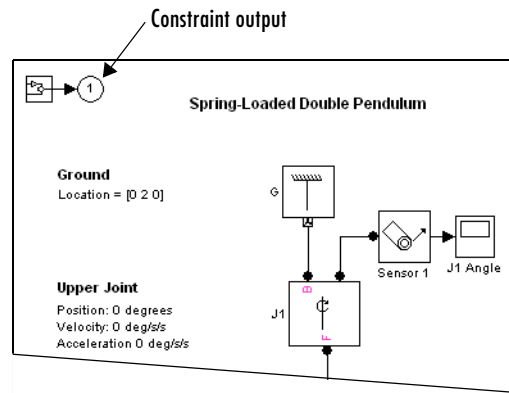
---

**Note** The `trim` command ignores initial states specified by Initial Condition blocks. Thus, you cannot use these blocks to specify the starting point for trimming a model. If your model contains IC blocks, create the initial state vector as if the blocks did not exist.

---

- 4 Set Trimming as the analysis type on the **Parameters** pane of the **Mechanical Environment Settings** dialog (see “Choosing an Analysis Mode” on page 5-4).

This option inserts a constraints subsystem and associated output at the top level of the model.



SimMechanics inserts the constraint output in order to make the constraints available to the `trim` command. The spring-loaded double pendulum has no constraints. Hence the constraint output port outputs nothing and is not needed to trim the pendulum.

- 5 Enter the following commands to find the equilibrium point nearest to the starting point.

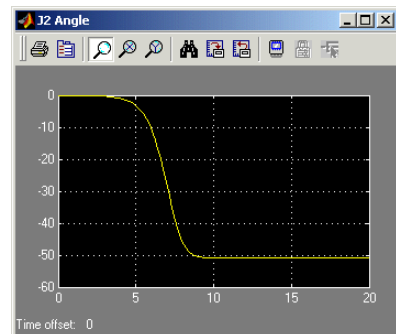
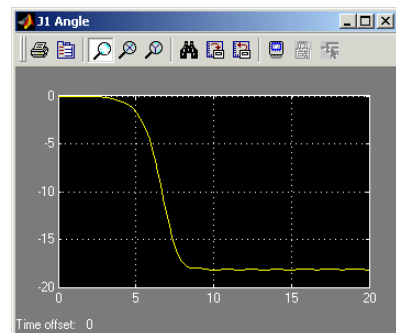
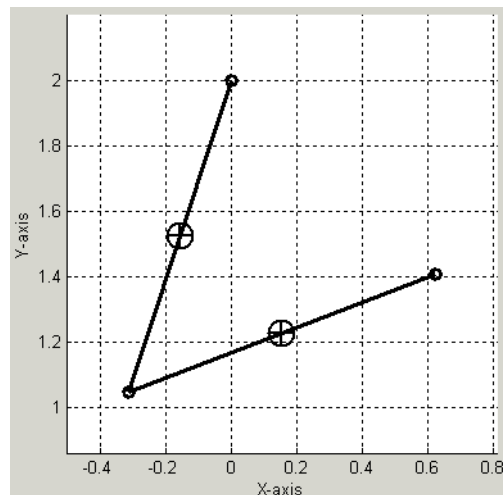
```
ix = [-0.6109; -0.1745; 0; 0];
iu = [];
[x,u,y,dx]=trim('mech_dpend_trim',ix,iu);
```

The array `ix` specifies the starting point determined in step 3. The array `iu` specifies the initial inputs of the system. Its value is null because the system has no inputs. (Thus the `u` and `y` outputs are null.) The third command executes the form of the `trim` command that finds a system’s steady-state (equilibrium) points, i.e., the points where the system’s state derivatives are zero. The array `x` contains the state vector corresponding to the first equilibrium point that the `trim` command finds:

$$x = \begin{bmatrix} -0.8882 \\ -0.3165 \\ -0.0000 \\ 0.0000 \end{bmatrix}$$

The resulting states are angular positions and velocities expressed in units of radians. Based on the layout of the model's state vector determined in step 2, the pendulum reaches equilibrium when its upper joint has deflected to an angle of  $-18.1341$  degrees and its lower joint to an angle of  $-50.8901$  degrees. The system state derivatives  $dx$  are zero, within tolerances.

- 6 Running the model in Forward Dynamics mode confirms that this is indeed an equilibrium point of the pendulum.

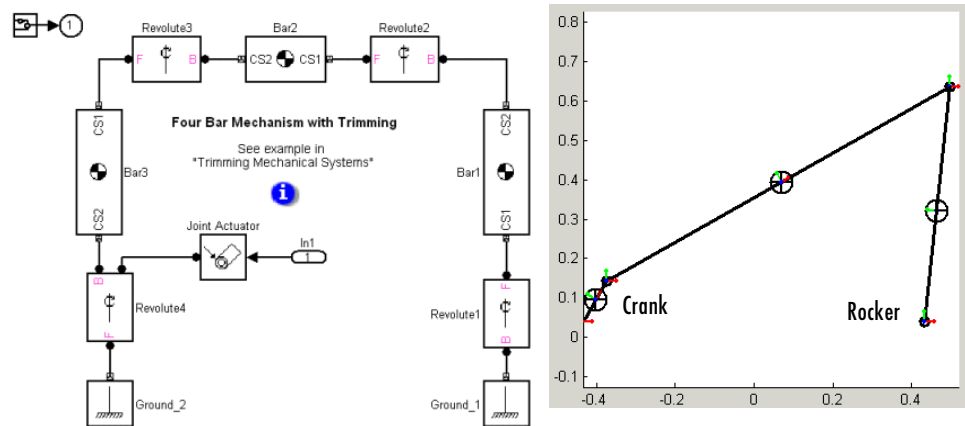


The simulation reveals that the spring stops unfolding after about nine seconds, that is, it reaches a steady-state point. At this point the angles of

the upper and lower joints are about  $-18$  and  $-50$  degrees, respectively, and the velocities are zero. The trim command can find the values of these states precisely.

## Constrained Example: Four-Bar System

Consider a planar four-bar system consisting of a crank, a coupler, and a rocker. The following figure shows a block diagram and a convex hull diagram of the four-bar system. The model is `mech_four_bar_trim`.



Suppose you want to find out the torque required to turn the crank at a constant angular velocity of 1 radian/sec over a range of crank angles. Run the model first in Forward Dynamics, then follow this procedure with the trim command and the SimMechanics Trimming mode to determine the torque:

- 1 Cut the closed loop that represents the four-bar system at the joint (Revolute1) connecting the rocker to ground (see "Modeling Bodies" on page 4-8).

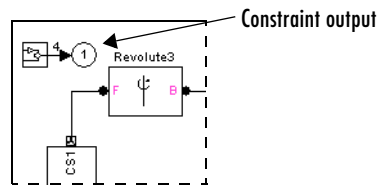
Manually cutting the rocker joint ensures that SimMechanics does not cut the four-bar loop at the crank joint and thereby eliminate the crank's position and velocity from the system's state vector.

**2** Select **Signal Dimensions** from the model's **Format** menu.

This causes Simulink to display the width of signals on the model diagram and hence enables you to read the number of constraints on the four-bar system from the diagram in the next step.

**3** Set the analysis mode for the model to Trimming (see “Trimming Mode” on page 5-4).

This causes SimMechanics to insert a subsystem and an output block that output a signal representing the mechanical constraints on the four-bar system.



The width of the constraint signal (4) reflects the fact that the four-bar system is constrained to move in a plane and thus has only four constraints: two position constraints and two velocity constraints.

**4** Determine the layout of the system's state vector.

Use the state vector command `mech_stateVectorMgr` to perform this task:

```
Handle = get_param(find_system('mech_four_bar_trim/
Revolute2'),'handle');
StateManager = mech_stateVectorMgr(Handle{1});
StateManager.StateNames
```

ans =

```
'mech_four_bar_trim/Revolute2:R1:Position'
'mech_four_bar_trim/Revolute3:R1:Position'
'mech_four_bar_trim/Revolute4:R1:Position'
'mech_four_bar_trim/Revolute2:R1:Velocity'
'mech_four_bar_trim/Revolute3:R1:Velocity'
```

```
'mech_four_bar_trim/Revolute4:R1:Velocity'
```

- 5** Specify the initial state vector  $x_0$  and the index array  $ix$ :

```
x0 = [0;0;0;0;0;1];  
ix = [3;6];
```

The array  $x_0$  specifies that the trim command should start its search for a solution with the four-bar system in its initial position and with the crank moving at an angular velocity (state 6) of 1 radian/sec. The array  $ix$  specifies that the angular position (state 3) and velocity (state 6) of the crank must equal their initial values, 0 radians and 1 radian/sec, respectively, at the equilibrium point. It is not necessary to constrain the other states because the four-bar system has only one global position DoF and only one global velocity DoF.

- 6** Specify zero as the initial estimate for the crank torque:

```
u0 = 0;
```

- 7** Set up the trim command to drive the constraint outputs to 0:

```
y0 = [0;0;0;0];  
iy = [1;2;3;4];
```

The  $y_0$  array specifies that the initial values of the constraint outputs are 0. The  $iy$  array specifies that the constraint outputs at the solution point must equal their initial values (0). This assures that the solution satisfies the mechanical constraints on the system.

---

**Note** The four-bar system has only constraint outputs. If you were trimming a system with nonconstraint outputs, you would have to include the nonconstraint outputs in the initial output vector.

---

- 8** Specify the state derivatives to be trimmed:

```
dx0 = [0;0;1;0;0;0];
```

```
idx = [6];
```

The dx0 array specifies the initial derivatives of the four-bar system's states. In particular, it specifies that the initial derivative of the crank angle (i.e., the crank angle velocity) is 1 radian/sec and all the other derivatives (i.e., velocities and accelerations) are 0. The idx array specifies that the acceleration of the crank at the solution point must be 0, i.e., the crank must be moving at a constant velocity. It is not necessary to constrain the accelerations of the other states because the system has only one velocity DoF.

---

**Note** The four-bar system has only mechanical states. If you were trimming a system that has nonmechanical states, you would have to include the nonmechanical states in the initial state vector.

---

- 9 Trim the system at the initial crank angle to verify that you have correctly set up the trim operation:

```
[x,u,y,dx] =
trim('mech_four_bar_trim',x0,u0,y0,ix,[],iy,dx0,idx);
```

Trim the system over a range of angles. Use the following program to perform this task.

```
Angle = [];
Input = [];
State = [];
dAngle = 2*pi/10;
Constraint = [];

for i=1:11;
    x0(3) = (i-1)*dAngle;
    x0(6) = 1;
    [x,u,y,dx] =
trim('mech_four_bar_trim',x0,u0,y0,ix,[],iy,dx0,idx);
    disp(['Iteration: ', num2str(i), ' completed.']);
    Angle(i) = x0(3);
    Input(:,i) = u;
    State(:,i) = x;
```

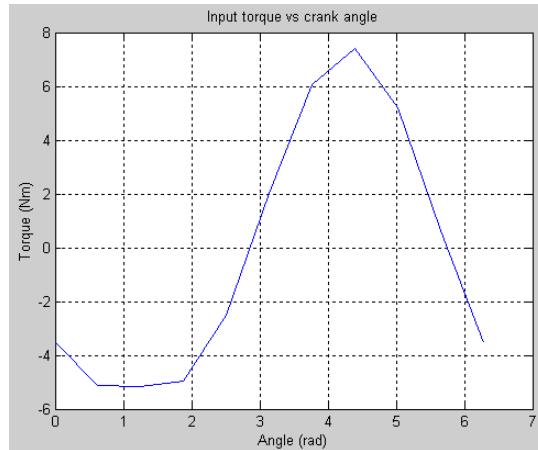
```
Constraint(:,i) = y;  
if (i>3),  
    u0 = spline(Angle,Input,Angle(end) + dAngle);  
    x0 = spline(Angle,State,Angle(end) + dAngle);  
else  
    x0 = x;  
    u0 = u;  
end;  
end;  
end;
```

**10** Plot the results.

Use the following program to perform this task:

```
figure(1);  
plot(Angle,Input);  
grid;  
xlabel('Angle (rad)');  
ylabel('Torque (Nm)');  
title('Input torque vs crank angle');
```

The following figure shows the resulting plot.





## Linearizing SimMechanics Models

The Simulink `linmod` command creates linear time-invariant (LTI) state-space models from Simulink models. You can use this command to generate an LTI state-space model from a SimMechanics model, for example, to serve as input to Control System Toolbox commands that generate controller models. The `linmod` command allows you to specify the point in state space about which it linearizes the model (the *operating point*). You should choose a point where your model is in equilibrium, i.e., where the net force on the model is zero. You can use the Simulink `trim` command to find a suitable operating point (see “Trimming SimMechanics Models” on page 7-12). By default, `linmod` uses an adaptive perturbation method to linearize a SimMechanics model. The **Mechanical Environment Settings** dialog box allows you to require that `linmod` use a fixed perturbation method instead (see “Linearization Pane” on page 5-10). The following example illustrates the use of `linmod` to linearize a SimMechanics model.

Consult the Simulink user’s guide for more on linearizing models.

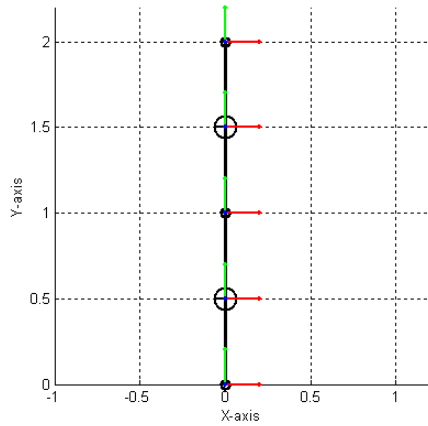
---

**Caution** Before linearizing a model, perform at least one simulation in Forward Dynamics mode. See “Choosing an Analysis Mode” on page 5-4.

---

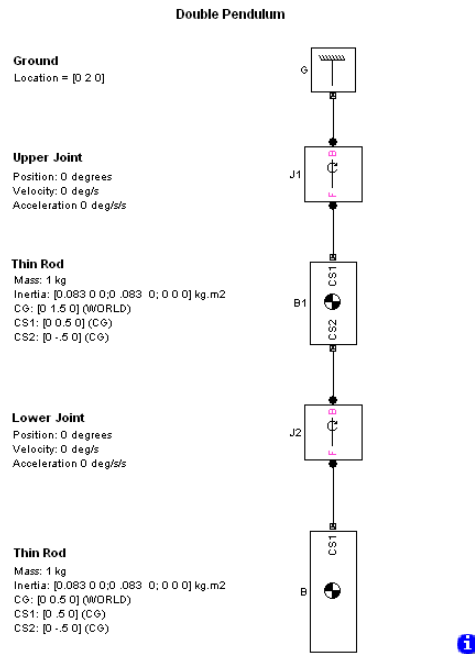
### Model Linearization Example: Double Pendulum

Consider a double pendulum initially hanging straight up and down.



The net force on the pendulum is zero in this configuration. The pendulum is thus in equilibrium.

The following figure shows a SimMechanics model of the pendulum, `mech_dpend_forw`.



To linearize this model, enter

```
[A B C D] = linmod('mech_dpend_forw');
```

at the MATLAB command line. This form of the `linmod` command linearizes the model about the model's initial states.

---

**Note** If you specify any joint primitive conditions with IC blocks, these initial condition values always override any state vector initial values specified via the `linmod` command.

The double pendulum model in this example contains no IC blocks. The initial conditions specified via the `linmod` command are therefore implemented without modification.

---

The matrices  $A$ ,  $B$ ,  $C$ ,  $D$  returned by the `linmod` command correspond to the standard mathematical representation of an LTI state-space model:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

where  $\mathbf{x}$  is the model's state vector,  $\mathbf{y}$  is its outputs, and  $\mathbf{u}$  is its inputs. The double pendulum model has no inputs or outputs. Consequently, only  $A$  is not null. This reduces the state-space model for the double pendulum to

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t)$$

where

$A =$

$$\begin{bmatrix} 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ -137.3400 & 39.2400 & 0 & 0 \\ 39.2400 & -19.6200 & 0 & 0 \end{bmatrix}$$

This model specifies the relationship between the state derivatives and the states of the double pendulum. The state vector of the LTI model has the same format as the state vector of the SimMechanics model. The SimMechanics `mech_stateVectorMgr` command gives the format of the state vector as follows:

```
vm = mech_stateVectorMgr('mech_dpend_forw/G');
```

ans =

```
'mech_dpend_forw/J2:R1:Position'  
'mech_dpend_forw/J1:R1:Position'  
'mech_dpend_forw/J2:R1:Velocity'  
'mech_dpend_forw/J1:R1:Velocity'
```

Multiplying  $A$  by the state vector  $\mathbf{x}$  yields the differential state equations corresponding to the LTI model of the double pendulum.

$$\dot{\theta}_1 = -19.6200 \cdot \theta_1 + 39.2400 \cdot \theta_2$$

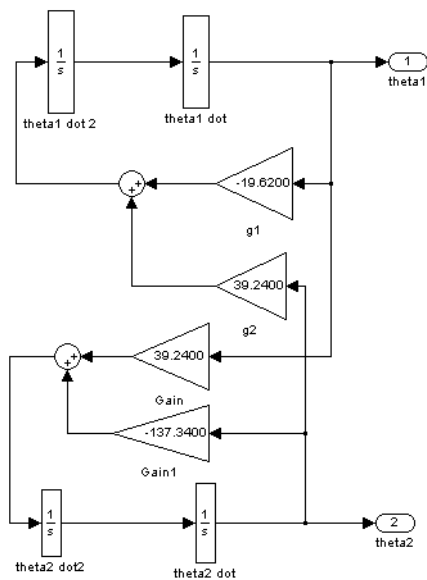
$$\dot{\theta}_2 = 39.2400 \cdot \theta_1 - 132.66033 \cdot \theta_2$$

where

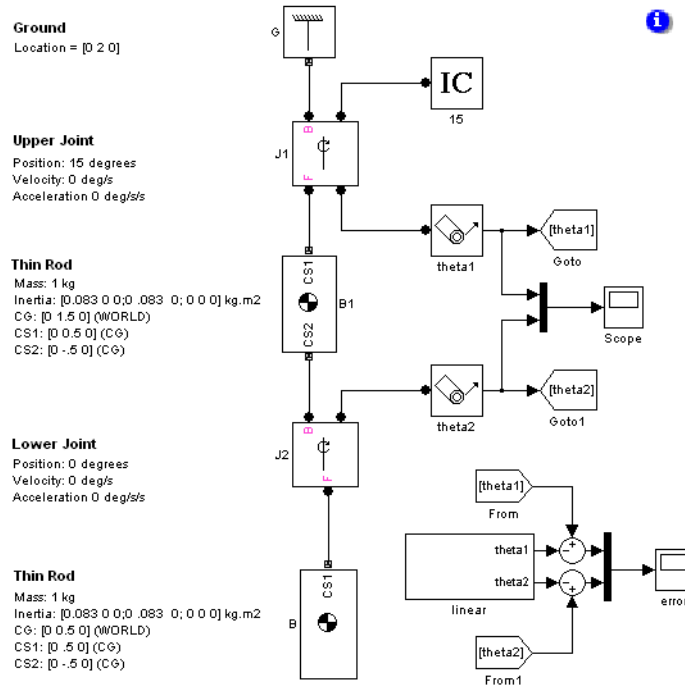
$\theta_1$  = position of top joint (J1)

$\theta_2$  = position of bottom joint (J2)

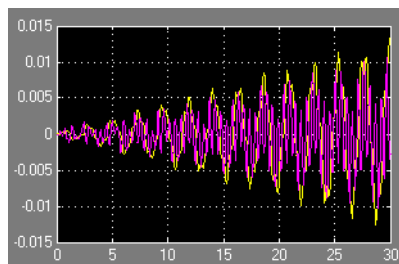
The following Simulink model implements the state space model represented by these equations.



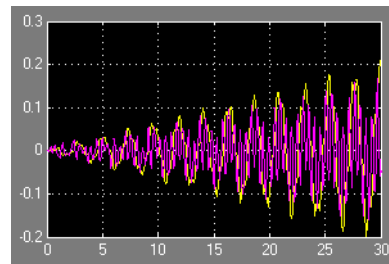
This model in turn allows creation of a model, located in `mech_depend_lin`, that computes the LTI approximation error.



Running the model twice with the upper joint deflected 2 degrees and 5 degrees, respectively, shows an increase in error as the initial state of the system strays from the pendulum's equilibrium position and as time elapses. This is the expected behavior of a linear state-space approximation.



2 degrees



5 degrees

# SimMechanics Block Reference

---

This chapter contains complete information on every block in SimMechanics. Refer to this chapter when you need to find detailed information on a particular block.

Blocks – By Category (p. 8-2)

The SimMechanics blocks summarized by block library

Blocks – Alphabetical List (p. 8-8)

The SimMechanics blocks listed alphabetically by name

## **Blocks – By Category**

This section consists of the block library hierarchy, a structured list of the SimMechanics libraries; and the block library contents, a listing of all SimMechanics blocks arranged by library.

Use the Simulink Library Browser or the SimMechanics library to access the blocks directly, guided by this hierarchical library list. The subsequent pages contain reference information for all blocks in SimMechanics, arranged by library and in alphabetical order by block name.

### **Bodies Library**

Contains blocks representing rigid bodies and ground points.

### **Joints Library**

Contains blocks representing relative motion degrees of freedom (DoFs) between bodies.

#### **Disassembled Joints Sublibrary**

Contains blocks representing initially misaligned DoFs between bodies.

#### **Massless Connectors Sublibrary**

Contains blocks representing rigidly separated DoF pairs between bodies.

### **Constraints & Drivers Library**

Contains blocks to restrict or externally drive relative DoFs between bodies.

### **Sensors & Actuators Library**

Contains blocks to initiate and measure motion, and to interface between SimMechanics and Simulink.

### **Force Elements Library**

Contains blocks to model forces/torques between bodies

### **Utilities Library**

Contains additional useful blocks for modeling and simulating machines.



---

## **Demos Library**

Contains Simulink model files with SimMechanics blocks. Online Help users can click this library name to browse the demos.

## Creating Bodies and Grounds

Body	Customizable rigid body
Ground	Immobile point at rest in World

## Creating Joints

### Creating Assembled Joints

Bearing	Composite: one prismatic, three revolute
Bushing	Composite: three prismatics, three revolute
Custom Joint	Customizable composite joint: up to three translational and three rotational DoFs
Cylindrical	Composite: one prismatic, one revolute (axes aligned)
Gimbal	Composite: three revolute
In-Plane	Composite: two prismatics
Planar	Composite: two prismatics, one revolute
Prismatic	Primitive: one translational DoF
Revolute	Primitive: one rotational DoF
Screw	Composite: one prismatic, one revolute (constrained)
Six-DoF	Composite: three prismatics, one spherical
Spherical	Primitive: three rotational DoFs at pivot
Telescoping	Composite: one prismatic, one spherical
Universal	Composite: two revolute
Weld	Primitive: no DoFs (rigid)

---

### **Creating Disassembled Joints**

Disassembled Cylindrical	Misaligned translational-rotational DoFs
Disassembled Prismatic	Misaligned translational DoF
Disassembled Revolute	Misaligned rotational DoF
Disassembled Spherical	Dislocated spherical DoF

### **Creating Massless Connectors**

Revolute-Revolute	Composite: revolute separated revolute
Revolute-Spherical	Composite: revolute separated spherical
Spherical-Spherical	Composite: spherical separated spherical

## Constraining and Driving Motion

Angle Driver	Specify angle between two body axes as function of time
Distance Driver	Specify distance between two body CS origins as function of time
Gear Constraint	Constrain two bodies rotating along two tangent pitch circles
Linear Driver	Specify vector component between two Body CS origins as function of time
Parallel Constraint	Constrain two body axes to be parallel
Point-Curve Constraint	Constrain motion of one body along curve on another body
Velocity Driver	Specify projected linear and angular velocities of two bodies as a function of time

## Actuating and Sensing Motion

Body Actuator	Apply force/torque or motion to a body
Body Sensor	Measure body motion
Constraint & Driver Sensor	Measure constraint force/torque between pair of constrained bodies
Driver Actuator	Apply motion to pair of constrained bodies
Joint Actuator	Apply force/torque or motion to a joint primitive
Joint Initial Condition Actuator	Apply initial positions and velocities to Joint primitives
Joint Sensor	Measure joint motion and force/torque
Joint Stiction Actuator	Apply friction to joint primitive
Variable Mass & Inertia Actuator	Vary the mass and inertia on a body (does <i>not</i> include thrust force or torque)

---

## Applying Force Elements

Body Spring & Damper	Model damped linear oscillator force between two bodies
Joint Spring & Damper	Model damped linear oscillator force or torque on a joint between two bodies

## Additional Useful Blocks

Connection Port	Connector port for a subsystem
Continuous Angle	Convert bounded angular signals to unbounded signals
Mechanical Branching Bar	Map multiple sensor/actuator lines to one sensor/actuator port
RotationMatrix2VR	Convert rotation matrix to rotation axis and angle

## Blocks – Alphabetical List

Angle Driver .....	8-10
Bearing .....	8-13
Body .....	8-18
Body Actuator .....	8-29
Body Sensor .....	8-32
Body Spring & Damper .....	8-36
Bushing .....	8-39
Connection Port .....	8-44
Constraint & Driver Sensor .....	8-45
Continuous Angle .....	8-48
Custom Joint .....	8-51
Cylindrical .....	8-58
Disassembled Cylindrical .....	8-62
Disassembled Prismatic .....	8-67
Disassembled Revolute .....	8-71
Disassembled Spherical .....	8-75
Distance Driver .....	8-78
Driver Actuator .....	8-81
Gear Constraint .....	8-84
Gimbal .....	8-88
Ground .....	8-93
In-Plane .....	8-95
Joint Actuator .....	8-100
Joint Initial Condition Actuator .....	8-105
Joint Sensor .....	8-109
Joint Spring & Damper .....	8-117
Joint Stiction Actuator .....	8-121
Linear Driver .....	8-128
Mechanical Branching Bar .....	8-131
Parallel Constraint .....	8-135
Planar .....	8-138
Point-Curve Constraint .....	8-143
Prismatic .....	8-152
Revolute .....	8-156
Revolute-Revolute .....	8-160

---

Revolute-Spherical .....	8-165
RotationMatrix2VR .....	8-170
Screw .....	8-172
Six-DoF .....	8-177
Spherical .....	8-181
Spherical-Spherical .....	8-186
Telescoping .....	8-190
Universal .....	8-194
Variable Mass & Inertia Actuator .....	8-199
Velocity Driver .....	8-203
Weld .....	8-207
mech_stateVectorMgr .....	9-3

# Angle Driver

---

**Purpose** Specify the angle between two body axis vectors as a function of time

**Library** Constraints & Drivers

## Description



The Angle Driver block drives axis vectors defined on two Bodies. You specify fixed base and fixed follower body axis vectors  $\mathbf{a}_B$ ,  $\mathbf{a}_F$  in the Body CS on either side of the Driver on each body, then drive the angle between the body axis vectors as a function of time.

The Angle Driver block specifies the angle  $\theta$  defined by

$$\cos \theta = |\mathbf{a}_B \cdot \mathbf{a}_F| / (|\mathbf{a}_B| |\mathbf{a}_F|)$$

as a function of time:  $\theta = f(t)$ . You connect the Angle Driver to a Driver Actuator block.

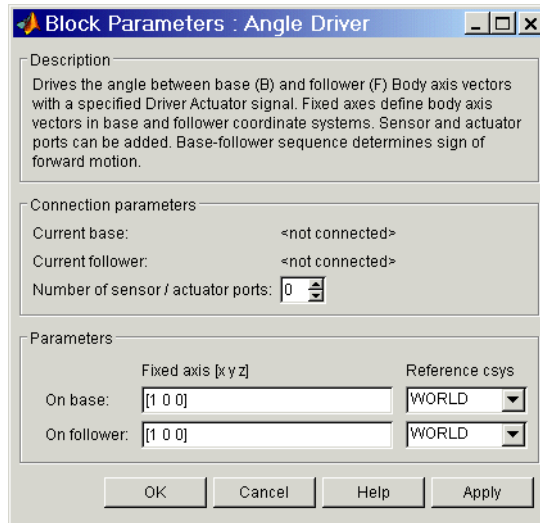
The Simulink input signal into the Driver Actuator specifies the time-dependent driving function  $f(t)$  and its first two derivatives, as well as their units. If you do not actuate Angle Driver, this block acts as a time-independent constraint that freezes the angle between the two body axes at its initial value during the simulation.

Drivers restrict relative degrees of freedom (DoFs) between a pair of bodies as specified functions of time. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Driver blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Drivers are ordered as base and follower, fixing the direction of relative motion.

You can also connect a Constraint & Driver Sensor to any Driver measure the reaction forces/torques between the driven bodies.



## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Angle Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Angle Driver base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Angle Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Angle Driver base and follower Body connector ports” figure.

### Number of sensor/actuator ports

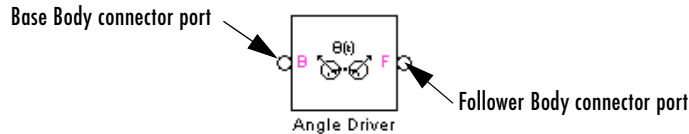
Using this spinner menu, you can set the number of extra connector ports needed for connecting Driver Actuator and Constraint & Driver Sensor blocks to this Driver. The default is 0.

To activate the Driver, connect a Driver Actuator.

# Angle Driver

---

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-handed sense about the rotation axis.



## Angle Driver base and follower Body connector ports

### Parameters

#### Fixed axis

For the **Base** and **Follower** bodies, respectively, enter the body axis vectors. The defaults are [1 0 0].

#### Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the **Base** and **Follower** body axis vectors are oriented with respect to. This CS also determines the absolute meaning of reaction forces/torques at this Driver. The defaults are WORLD.

### See Also

Constraint & Driver Sensor, Driver Actuator, Parallel Constraint, Velocity Driver

See “Modeling Constraints and Drivers” on page 4-34 for more on restricting DoFs with Drivers.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on using drivers in closed loops.

See “Constraining and Driving Motion” on page 8-6.

**Purpose** Represent a composite joint with one translational and three rotational DoFs

**Library** Joints

**Description** The Bearing block represents a composite joint with one translational degree of freedom (DoF) as one prismatic primitive and three rotational DoFs as three revolute primitives. There are no constraints among the primitives. Unlike Telescoping, Bearing represents the rotational DoFs as three revolutes, rather than as one spherical.



---

**Caution** A joint with three revolute primitives becomes singular if two or three of the rotation axes become parallel (“gimbal lock”). The simulation stops with an error in this case.

---

You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Bearing block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

---

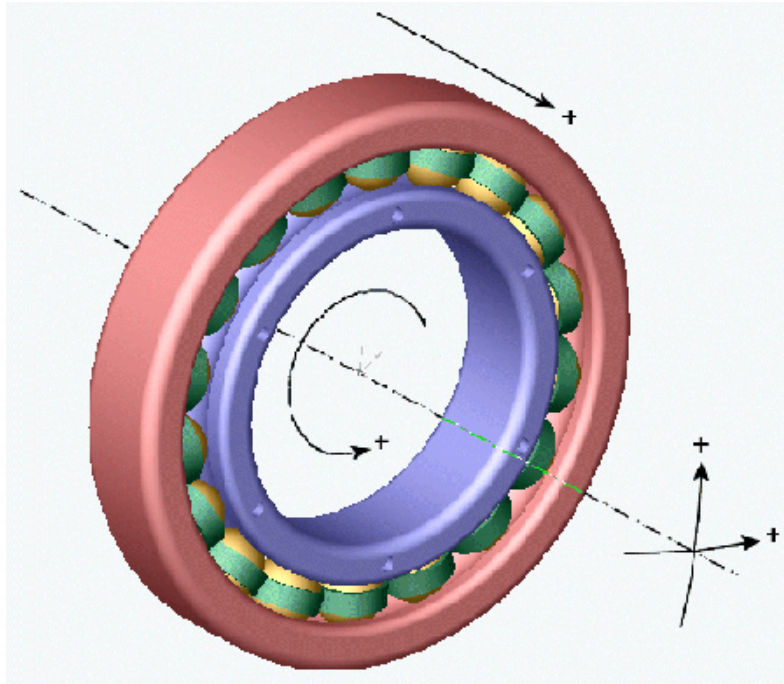
**Note** Bearings are often represented by one translational and one rotational DoF. The Bearing block has *three* rotational degrees of freedom, rather than one, in order to represent transverse “play” in the joint.

---

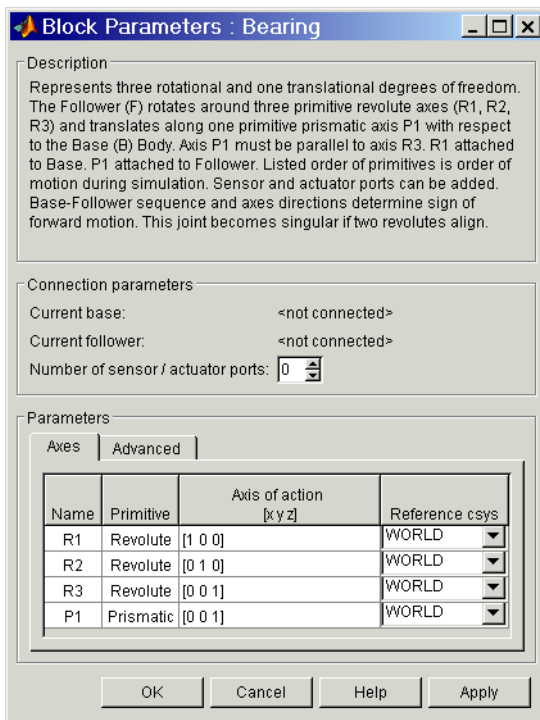
# Bearing

---

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Bearing block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Bearing base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive R1 in the primitive list in **Parameters**.

### Current follower

When you connect the follower (F) connector port on the Bearing block to a Body CS Port on a Body, this parameter is automatically reset to the name

# Bearing

of this Body CS. See the following “Bearing base and follower Body connector ports” figure.

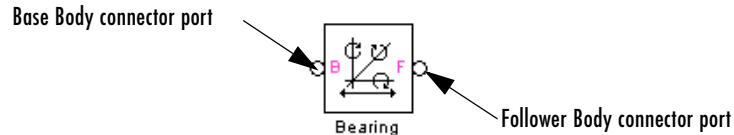
The follower Body is automatically connected to the last joint primitive P1 in the primitive list in **Parameters**.

## Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motions of prismatic and revolute primitives are specified in linear and angular units, respectively.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower moving around the rotational axis following the right-hand rule.



## Bearing base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Bearing has an entry line. These lines specify the direction of the axes of action of the DoFs that the Bearing represents.

### Name - Primitive

The primitive list states the names and types of joint primitives that make up the Bearing block: revolute primitives R1, R2, R3, and prismatic primitive P1.

### Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Prismatic: axis of translation
- Revolute: axis of rotation

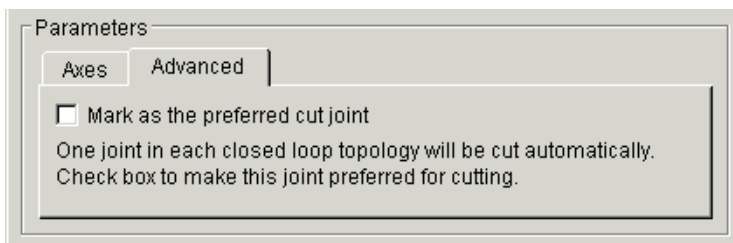
The default vectors are shown in the dialog box above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the revolute axes can be parallel.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Bushing, Cylindrical, Gimbal, Prismatic, Revolute

See "Modeling Joints" on page 4-17 for more on representing DoFs with Joints.

See "Checking Schematic Topology" on page 4-64 and "How SimMechanics Works" on page 5-14 for more on closed loops and cutting.

# Body

---

**Purpose** Represent a customizable rigid body

**Library** Bodies

**Description** The Body block represents a rigid body whose properties you customize. The representation you specify includes:



- The body's *mass* and *moment of inertia tensor*
- The coordinates for the body's *center of gravity* (CG)
- Any number of optional *Body coordinate systems* (CSs)


A rigid body is defined in space by the position of its CG (or center of mass) and its orientation in some CS.

---

**Setting Body Initial Conditions** The initial position and orientation of a body are set by the entries in its Body dialog. These initial conditions remain unchanged; unless, with a Joint Initial Condition Actuator, you change the initial conditions of the Joint(s) connected to the Body prior to starting the simulation, or you actuate the Body with a Body Actuator.

---

In SimMechanics, you enter the body's properties in two classes, the geometric properties and the mass properties:

- The *geometric properties* are defined by the body's Body CSs.
  - The minimum required Body CS is the CS with its origin at the CG. The CG point specifies both the initial position of the whole body and the origin of the CG CS. You must also orient the CG CS axes.
  - You can place any number of additional Body CSs on a body. You must define each Body CS by the position of its origin and the orientation of its CS axes.
  - Each connection of a Joint, Constraint/Drive, Actuator, or Sensor block to a Body requires an anchor point on the Body. This anchor point is one of the Body CS origins.
  - Body CSs on the block available for connections are shown by Body CS ports  on the sides of the block. You can show or hide each Body CS on the block sides.



- The set of a body's Body CS origins (including the CG CS) defines the body's *convex hull*, one of the visualization shapes available for representing a body in space.
- The *mass properties* are defined by the body's mass and inertia tensor.
  - The mass is the body's inertia and controls the translational acceleration of the CG in response to an applied force.
  - The inertia tensor measures the distribution of mass density in the body and controls the rotational acceleration of the body about the CG in response to an applied torque.
  - The components of the inertia tensor control the initial orientation of the body and are always interpreted as being in the CG CS axes. The orientation of the CG CS axes with respect to another CS external to the body (the World CS, a CS on a Ground, or a CS on another Body) then determines the orientation of the body with respect to other bodies or with respect to World.
  - The body's inertia tensor defines its *principal axes* and *moments* and its *equivalent ellipsoid*, one of the visualization shapes available for representing a body in space.

## Default Initial State of a Body

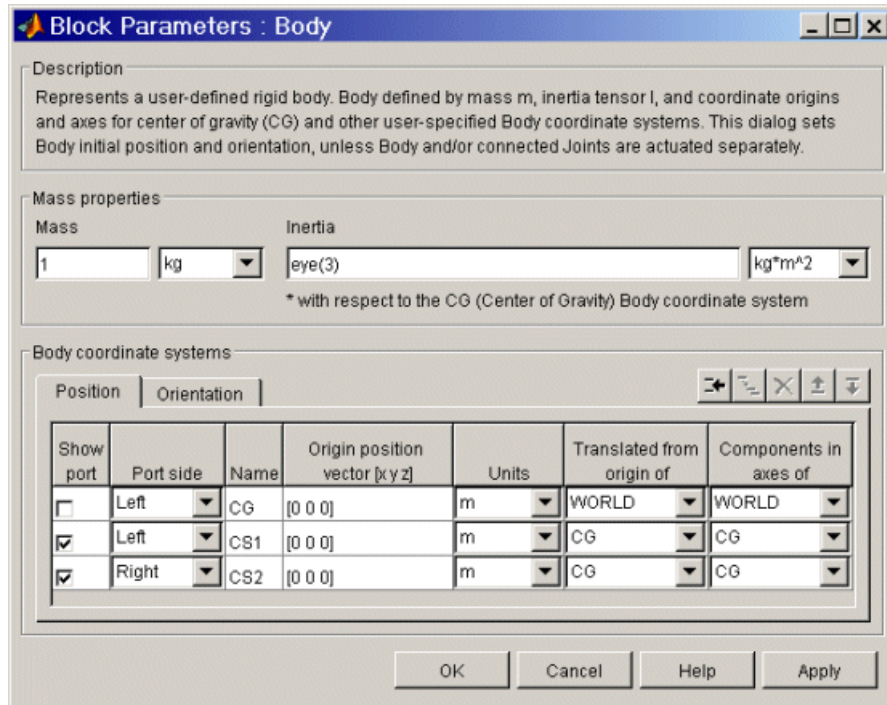
These two sets of properties determine a body's initial position and orientation:

- The initial position of a body is set by the position of its CG.
- The initial orientation is set by its inertia tensor components (in the CG CS) and the orientation of the CG CS axes with respect to other CSs in the machine.

The initial conditions of a machine can be changed with Joint Initial Condition Actuator blocks before starting a simulation. If you do not change the initial state of a Body before simulation, SimMechanics sets its initial position/orientation to its Body dialog box entries. SimMechanics also sets the Body's initial linear/angular velocities to zero in this case.

# Body

## Dialog Box and Parameters



The dialog box has two active areas, **Mass Properties** and **Body Coordinate Systems**.

## Mass Properties

### Mass

Enter the mass of the body in the first field and choose units in the pull-down menu to the right. The mass must be a positive, real number or MATLAB equivalent expression. The defaults are 1 and kg (kilograms).

### Inertia tensor

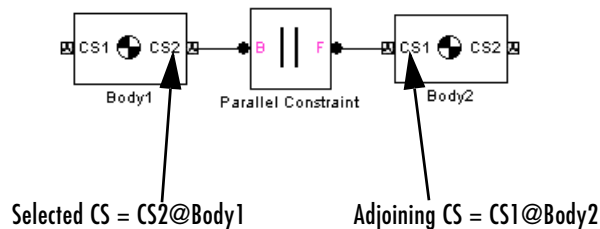
Enter the inertia tensor (with respect to the Body CG CS axes) in the first field and choose units in the pull-down menu to the right. The tensor must be a 3-by-3 real, symmetric matrix. The default tensor is `eye(3)`, the MATLAB 3-by-3 identity matrix. A zero tensor `zeros(3,3)` defines a point mass. The units default is  $\text{kg} \cdot \text{m}^2$  (kilograms-meters<sup>2</sup>).

## Body Coordinate Systems

### Configuring a Body Coordinate System

You set up Body CSs in the **Body coordinate systems** area:

- The default configuration consists of three Body CSs: the required CG CS attached to the body's CG and two other optional Body CSs, called "CS1" and "CS2", for connecting Joints, Constraints, or Drivers.
- You can configure the CG CS but not delete it. You also cannot create additional CG CSs, although you can duplicate the CG CS with a different name. (See following.)
- The other CSs can be configured or deleted as you want.
- Configuring a Body CSs requires two groups of steps:
  - positioning the Body CS origin in the **Position** panel
  - orienting the Body CS axes in the **Orientation** panel.
- Defining Body CSs requires referring to other, pre-existing CSs in the model. In a given Body block, you can refer to Body and grounded CSs in three ways. The references must be to:
  - World
  - Other Body CSs on the same body
  - The *Adjoining CS*, the coordinate system on a neighboring body or ground directly connected to the selected Body CS by a Joint, Constraint, or Driver.



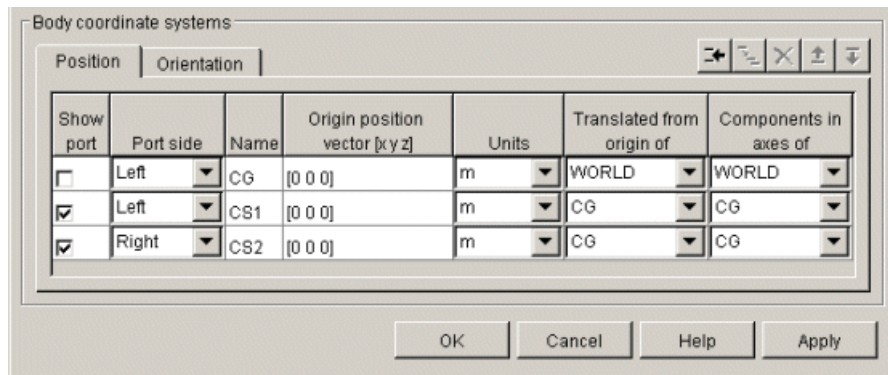
- Toggle between the **Position** or **Orientation** panels with the tabs. Each Body CS is labeled with a name, CG for the CG CS, and CS1, CS2, etc., for additional CSs.

## Configuring the Position fields

The **Position** fields for each Body CS specify the position of that CS's origin as a translation vector:

- The numerical components of the vector carry units.
- The origin is displaced from the origin of another, pre-existing CS in your machine by this translation vector.
- The translation vector's components are oriented with respect to another set of CS axes.

Highlight each Body CS to configure it.



### Origin position vector [x y z]

Enter the translation vector that defines the position of the Body CS's origin.

The special entry for the CG CS origin places the entire body.

### Units

Choose linear units for the translation vector. The default is m (meters).

### Translated from the origin of

In the pull-down menu, choose the other, pre-existing CS in your machine that defines the starting point for the translation vector. The choices are WORLD, ADJOINING, and the other Body CSs on this Body. The ending point of the translation vector is this Body CS's origin.

For the CG CS, the default starting-point CS is WORLD. For the additional Body CSs (CS1, CS2, etc.), the default starting point CS is this Body's CG.

## Components in the axes of

In the pull-down menu, choose the CS whose axes define the orientation of the translation vector's components. The choices are WORLD, ADJOINING, and the other Body CSs on this Body. The translation vector's components are projected along the axes of the CS chosen in this column.

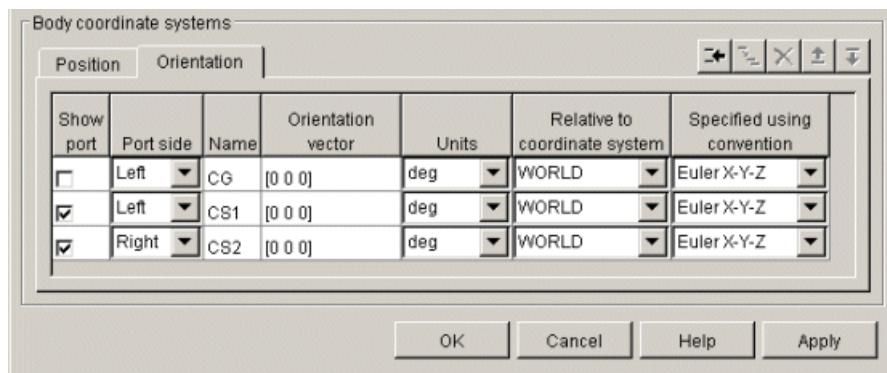
For the CG CS, the default orientation CS is WORLD. For the additional Body CSs (CS1, CS2, etc.), the default orientation CS is this Body's CG.

## Configuring the Orientation Fields

The **Orientation** fields for each Body CS specify the orientation of that CS's triad of axes as a rotation:

- The orientation vector specifying the rotation vector has three components.
- The numerical components of the vector carry units.
- The rotation is oriented with respect to some other, pre-existing set of CS coordinate axes in your machine.
- The orientation vector's components are interpreted in the convention of a rotation representation.

Highlight each Body CS to configure it.



## Orientation vector

Enter the components of the rotation that defines the orientation of the Body CS's axes. The geometric meaning of these components is determined by the **Specified using convention** column.

The special entry for the CG CS orients the CG CS axes. Together with the **Inertia tensor** entry in **Mass properties**, the CG CS axes orient the whole body with respect to another CS in your machine.

## Units

Choose angular units for the rotation, degrees or radians. The default is deg (degrees).

## Relative to coordinate system

In the pull-down menu, choose the other, pre-existing CS in your machine that defines the starting orientation for the rotation. The choices are WORLD, ADJOINING, and the other Body CSs on this Body.

## Specified using convention

In the pull-down menu, choose the representation type for the rotation:

Rotation Type	Orientation vector components
Quaternion	$[n_x \cdot \sin(\theta/2) \quad n_y \cdot \sin(\theta/2) \quad n_z \cdot \sin(\theta/2) \quad \cos(\theta/2)]$
3x3Transform	3-by-3 orthogonal rotation matrix $R$
Euler	Rotation angles about: [X-axis Y-axis Z-axis]

## Rotation Conventions

There are three generic conventions for representing rotations:

- Euler

The Euler angle convention specifies the rotation of the Body CS axes by rotating about three axes in a sequence. The components of the 1-by-3 row vector are the angles of rotation about the  $X$ -,  $Y$ -, and  $Z$ -axes, respectively, in degrees or radians.

For example, Euler  $X$ - $Y$ - $Z$  means, rotate about the original  $X$ -axis, then about the first intermediate  $Y$ -axis, and then about the second intermediate  $Z$ -axis. Another example: Euler  $X$ - $Z$ - $Y$  means, rotate about the original

X-axis, then about the first intermediate Z-axis, and then about the second intermediate Y-axis.

- 3-by-3 Transform

The transform convention specifies the rotation as a dimensionless 3-by-3 orthogonal rotation matrix. The inverse of an orthogonal matrix  $R$  is equal to its transpose:  $R^{-1} = R^T$ .

The columns of  $R$  are the  $(x,y,z)$  unit vectors of the Body CS axes. The units menu is inactive.

- Quaternion

The quaternion convention specifies the rotation in angle-axis form as a dimensionless 1-by-4 row vector:

$$[n_x * \sin(\theta/2) \quad n_y * \sin(\theta/2) \quad n_z * \sin(\theta/2) \quad \cos(\theta/2)]$$

$\mathbf{n} = (n_x, n_y, n_z)$  is a three-component vector of length unity:  $\mathbf{n} * \mathbf{n} = n_x^2 + n_y^2 + n_z^2 = 1$ .

The unit vector  $\mathbf{n}$  specifies the axis of rotation. The rotation angle about that axis is  $\theta$  and follows the right-hand rule.

## Managing the Body Coordinate Systems List

The Body coordinate system controls (see the following “Body coordinate systems controls” figure) allow you to add, duplicate, reorder, and delete Body CSs on a Body block:

- To add a Body CS to the list:

- Highlight an existing Body CS in the list.

- Click the **Add** button (see the following “See Also” figure).

A new Body CS will appear immediately below the Body CS you highlighted. New Body CSs are named in sequence after the current ones: CS3, CS4, etc.

- To duplicate a Body CS in the list:

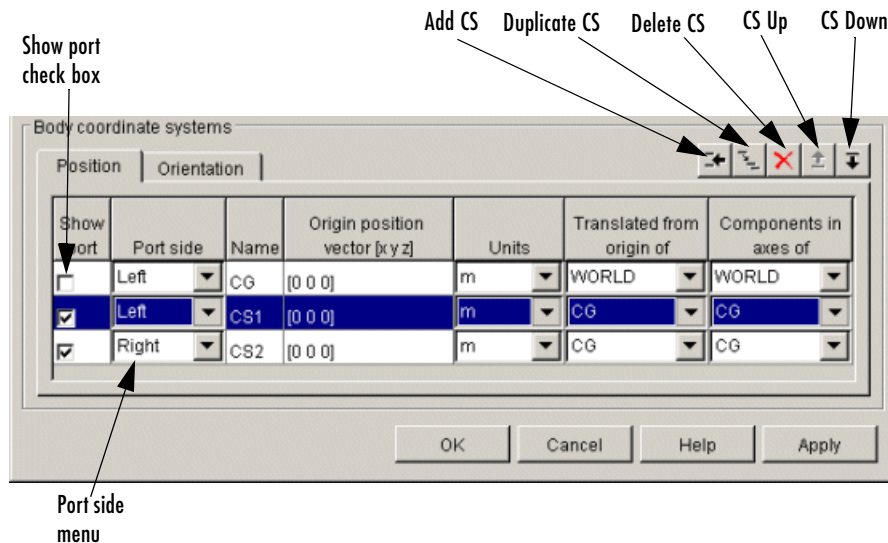
- Highlight the Body CS you want to duplicate.

- Click on the **Duplicate** button (see the following “See Also” figure).

A new Body CS with the same configuration but a different name will appear immediately below the Body CS you highlighted. New Body CSs are named in sequence after the current ones: CS3, CS4, etc.

# Body

- To change the position of a Body CS in the list:
  - Highlight the Body CS whose position you want to change.
  - Click on the **Up** or **Down** button (see the following “See Also” figure) until the Body CS is where you want it.
- To delete a Body CS from the list:
  - Highlight the Body CS you want to delete.
  - Click on the **Delete** button (see the following “See Also” figure).  
The Body CS you highlighted disappears.
- You cannot delete the Body’s CG CS.




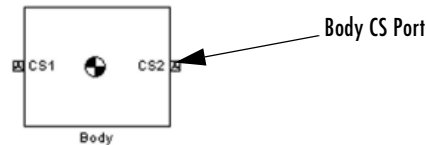
## Body coordinate systems controls

### Managing Body CS Ports on a Body Block

To connect a Joint, Constraint/Driver, Actuator, or Sensor block to a Body block requires an existing and configured Body CS in the Body to anchor the other block to:




- These other blocks define, constraint, impart, and measure the motion of bodies with respect to the origin and coordinate axes of Body CSs. Connect each of these blocks to a Body CS with a connection line.
- The actual connection line running from the other block to the Body block must be anchored to a displayed Body CS Port  on the side of the Body block in the model window.



- A displayed Body CS Port on a Body block indicates a Body CS with the displayed name configured internally within the Body block.
- Not all the Body CSs configured inside a Body block need to be displayed, however.

See the “Body coordinate systems controls” figure preceding.

### Show port

Select this check box for any Body CS to create a corresponding Body CS Port  on the side of the Body block. The Body CS on that line in the Body CS list is now accessible for connection to other blocks.

Unselect this check box to remove the Body CS Port corresponding to that Body CS on that line in the list.

The defaults are: unselected for CG, selected for CS1 and CS2.

To apply your choices to the displayed Body block, click **Apply**.

### Port side

From the pull-down menu, choose which side of the Body block you want the Body CS Port for that Body CS to be placed on, Left or Right.

The defaults are Left for CG and CS1 and Right for CS2.

To apply your choices to the displayed Body block, click **Apply**.

### See Also

Body Actuator, Body Sensor, Body Sensor, Ground, Mechanical Branching Bar

# Body

---

See “Representing Body Positions and Orientations” on page 3-2 for more details on body coordinate system rotations.

See “Modeling Bodies” on page 4-8, “Creating Body CS Ports” on page 4-16, and “Choosing Visualization Options in SimMechanics” on page 6-2 for more on setting up Bodies in machines.

See “Modeling Sensors” on page 4-60 for setting general initial conditions (positions and velocities) of DoFs in a machine.

**Purpose** Apply force/torque to a body

**Library** Sensors & Actuators

**Description** The Body Actuator block actuates a Body block with a generalized force signal, representing a force/torque applied to the body:



- Force for translational motion
- Torque for rotational motion

The generalized force is a function of time specified by a Simulink input signal. This signal can be any Simulink signal, including a signal feedback from a Sensor block.

The Body Actuator applies the actuation signal in the reference coordinate system (CS) specified in the block dialog box.

The inport is the Simulink input signal. The output is the connector port you connect to the Body block you want to actuate.

You should carefully distinguish the Body Actuator from the Driver blocks:

- The Body Actuator block applies generalized forces to one body in a specified reference CS.
- The Driver blocks drive relative *degrees of freedom* between pairs of bodies.

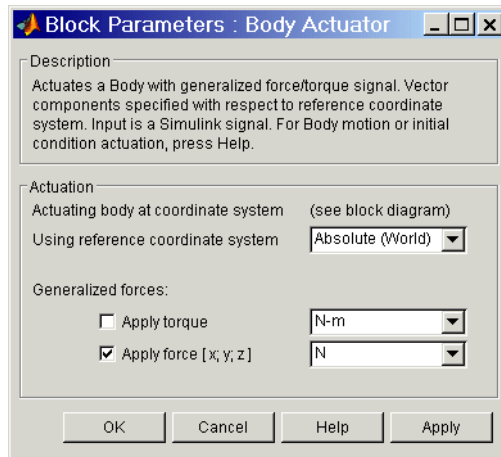
## Other Ways to Actuate Bodies

The Body Actuator block actuates a Body with force/torque signals only. To actuate a Body with motion signals or initial conditions, or to drive the relative degrees of freedom between a pair of Bodies, see “Actuating a Joint” on page 4-44 and “Joint Actuator Example: Body Driver” on page 4-46.

The `mech_body_driver` model from the Demos library shows how to drive the relative DoFs between a pair of bodies. To actuate one body alone, use this model and replace the second Body block with a Ground block. To set body initial conditions, replace the second Body block with a Ground block and the Joint Actuators with Joint Initial Condition Actuators.

# Body Actuator

## Dialog Box and Parameters



The dialog box has one active area, **Actuation**.

## Actuation

### Actuating body at coordinate system

This field is not active. You choose the Body CS at which to apply the actuation by connecting it to the Body Actuator.

### Using reference coordinate system

In the pull-down menu, choose the coordinate system (CS) in which the actuating force/torque is interpreted: either the `Local` (Body CS) to which the Actuator is connected or the default `Absolute (World)`.

## Generalized Forces

You can apply a force, a torque, or both generalized forces to a body.

If you apply both, you need to bundle the torque and force vectors into a 6-component signal, in the order shown in the dialog box.

### Apply torque

Select the check box if part or all of the actuating signal is a rotational torque. The default is unselected. The Simulink torque input is a 3-component bundled signal.

In the pull-down menu, choose units for the actuating torque. The default is N-m (Newton-meters).

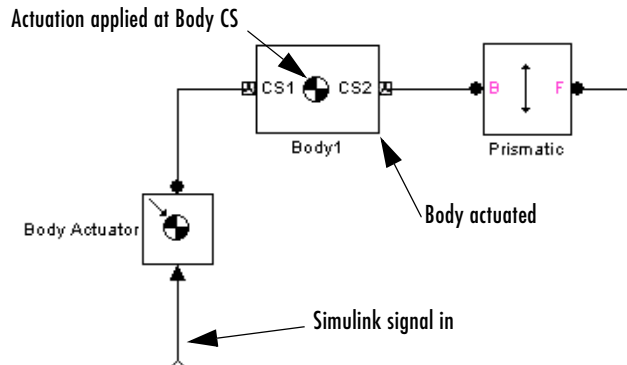
## Apply force

Select the check box if part or all of the actuating signal is a translational force. The default is selected. The Simulink force input is a 3-component bundled signal.

In the pull-down menu, choose units for the actuating force. The default is N (Newtons).

## Example

Here is a Body Actuator connected to a Body:



You must connect the Body Actuator to the Body at one of that Body's attached Body CSs, at the corresponding Body CS Port. The actuation signal acts on the Body at that Body CS's origin.

## See Also

Body, Body Sensor, Driver Actuator, Joint Actuator, Joint Initial Condition Actuator, Mechanical Branching Bar

See “Representing Body Positions and Orientations” on page 3-2 for more details on body coordinate system rotations.

See “Actuating a Joint” on page 4-44 and “Joint Actuator Example: Body Driver” on page 4-46.

See “Creating Bodies and Grounds” on page 8-4 and “Constraining and Driving Motion” on page 8-6.

In Simulink, see the Signal Routing Library and the Sources Library.

# Body Sensor

---

**Purpose** Measure body motion

**Library** Sensors & Actuators

**Description**



The Body Sensor block detects the position, velocity, and/or acceleration of a body represented by a Body block.

The Body Sensor measures the motion in the reference coordinate system (CS) specified in the block dialog box.

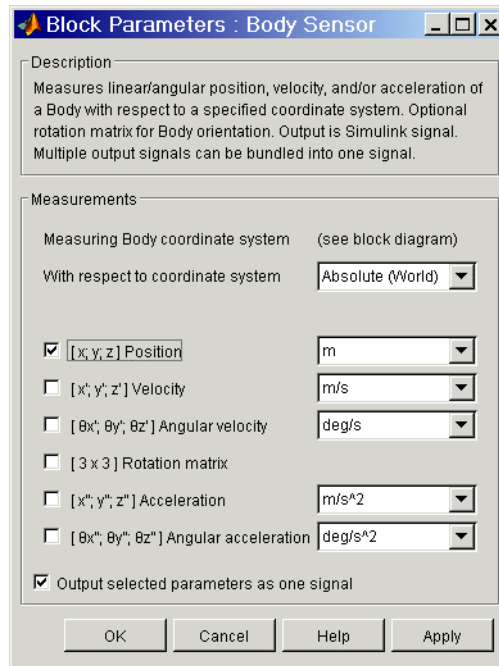
You can measure one, two, or all three of these motion types:

- Translational motion of a body, in terms of linear position, velocity, and/or acceleration vectors
- Rotational motion of a body, in terms of angular velocity and/or acceleration vectors
- Rotational motion of a body, in terms of a 3-by-3 rotation matrix  $R$

The input is the connector port connected to the Body being sensed. The output is a set of Simulink signals or one bundled Simulink signal of the position, velocity, and/or acceleration vector(s) and/or the rotation matrix of the body.

A body's orientation rotation matrix  $R$  relates vector components measured in the body CS and in the inertial World CS by  $[R] \cdot \mathbf{v}_b = \mathbf{v}_g$ . The column vector  $\mathbf{v}_b$  lists the vector  $\mathbf{v}$ 's three components measured in the body CS. The column vector  $\mathbf{v}_g$  lists the vector  $\mathbf{v}$ 's three components measured in the World CS.

## Dialog Box and Parameters



The dialog box has one active area, **Measurements**.

### Measurements **Measuring Body coordinate system**

This field is not active. You choose the Body CS at which to measure the motion by connecting it to the Body Sensor.

### **With respect to coordinate system**

In the pull-down menu, choose the coordinate system in which the body motion is measured: either the Local (Body CS) to which the Sensor is connected or the default Absolute (World).

In the Absolute case, the rotation matrix  $R$  and the motion vectors have components measured relative to the inertial World CS axes. In the Local case, the same body motion signals are premultiplied by the body's inverse orientation rotation matrix  $R^{-1} = R^T$ .

# Body Sensor

---

Select the check box(es) for each of the possible measurements you want to make:

- Linear motion: **Position**, **Velocity**, and **Acceleration** vectors
- Angular motion: **Angular velocity** and **Angular acceleration** vectors and **Rotation matrix**:
  - The **Rotation matrix** is the 3-by-3 orthogonal rotation matrix  $R$ :

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

representing rotational orientation and satisfying  $R^T R = R R^T = I$ . The components are output column-wise as a 9-component row vector:  $(R_{11}, R_{21}, R_{31}, R_{12}, R_{22}, R_{32}, R_{13}, R_{23}, R_{33})$ .

- If you choose the **With respect to coordinate system** as **Absolute (World)**, the **Rotation matrix** measures the body's rotational orientation with respect to the World CS, relative to its initial orientation. If you choose the **With respect to coordinate system** above as **Local (Body CS)**, the **Rotation matrix** returns the 3-by-3 identity matrix:  $R^T R = I$ .

Each vector measurement is a row vector in the Simulink output signal. The selected signals are ordered in the same sequence as the dialog box.

In the pull-down menus, choose the units for each of the measurements you want:

- Linear motion: the defaults are m (meters), m/s (meters/second),  $m/s^2$  (meters/second<sup>2</sup>), and N (Newtons), respectively, for **Position**, **Velocity**, and **Acceleration**.
- Angular motion: the defaults are deg/s (degrees/second) and  $deg/s^2$  (degrees/second<sup>2</sup>), respectively, for **Angular velocity** and **Angular acceleration**. The **Rotation matrix** is dimensionless.

## Output selected parameters as one signal

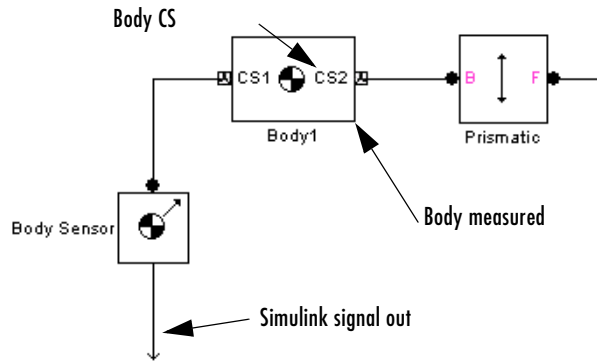
Select this check box to convert the output signals into a single bundled signal. The default is selected. If you unselect it, the Body Sensor block will grow as many Simulink outports as there are active signals selected, in the same order top to bottom, as in the dialog box.



If the check box is selected, the Simulink signal out has all the active (selected) signals ordered into a single row vector, in the same order you see in the dialog box. Unselected components are removed from the vector signal.

## Example

Here is a Body Sensor connected to a Body:



You must connect the Body Sensor to the Body at one of that Body's attached Body CSs, at the corresponding Body CS Port. The sensor measures the motion of that Body CS.

## See Also

Body, Body Actuator, Constraint & Driver Sensor, Joint Sensor, Mechanical Branching Bar

See “Representing Body Positions and Orientations” on page 3-2 for more details on body coordinate system rotations.

See “Representing Body Positions and Orientations” on page 3-2 and “Modeling Sensors” on page 4-60.

In Simulink, see the Signal Routing Library and the Sinks Library.

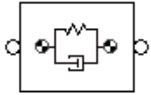
# Body Spring & Damper

---

**Purpose** Models a damped linear oscillator force between two bodies

**Library** Force Elements

## Description



The Body Spring & Damper block models the force of a damped spring acting between two bodies. By Newton's third law, the spring applies equal and opposite forces to the two bodies. You can use this Force Element block to model any linear (Hooke's law) force, with constant coefficients, that acts between a pair of bodies. One of the Bodies can be a Ground.

---

**Caution** Both the spring and the damper force act *only* along the axis connecting the two bodies.

The Body Spring & Damper block does *not* include the degree of freedom (DoF) needed to model the relative motion of the connected bodies. You need to add a Joint block between the Bodies to represent one, two, or three prismatic primitives. You can use Prismatic blocks or a Custom Joint block to accomplish this. The Body Spring & Damper block and Joints block must form a closed loop together with the two Bodies.

---

## Body Spring and Damper Theory

You connect this block to each Body,  $A$  or  $B$ , at a Body coordinate system (CS). If  $\mathbf{r}_A$  and  $\mathbf{r}_B$  are the positions of these Body CSs, the relative position vector connecting them is  $\mathbf{r} = \mathbf{r}_B - \mathbf{r}_A$ . The distance of separation is  $|\mathbf{r}|$ . The relative velocity is  $\mathbf{v} = d\mathbf{r}/dt$ . Then the vector force that body  $A$  exerts on body  $B$  is

$$\mathbf{F} = -k(|\mathbf{r}| - r_0)(\mathbf{r}/|\mathbf{r}|) - b(\mathbf{v} \cdot \mathbf{r})(\mathbf{r}/|\mathbf{r}|^2)$$

The first term represents the spring or linear displacement force. The second represents the damper or velocity dissipation force, which acts only along the direction of  $\mathbf{r}$ . Thus the damper is equivalent to a dashpot, not a viscous medium.

You specify the spring constant  $k$ , the natural spring length (offset)  $r_0$ , and the damping constant  $b$ . The natural length is the length of the spring with no forces acting on it and physically must be nonnegative:  $r_0 \geq 0$ . A stable spring requires  $k > 0$ . A damping representing dissipation and respecting the second

law of thermodynamics requires  $b \geq 0$ . You can use a negative  $b$  to represent energy pumping.

## The Spring and Damper Force in Singular Cases

In certain cases, the force formula breaks down, and SimMechanics uses special rules to determine the force.

- If  $r_0$  and  $v \neq 0$ , and  $r = 0$  at some instant, both terms in the force become singular. The spring force is reprojected along the velocity vector. That is,  $v/|v|$  replaces  $r/|r|$  in both terms, once in the first term and twice in the second. If the state  $r = 0$  does not persist for more than an instant, this replacement has no effect on the motion.
- If  $r_0 \neq 0$ , and both  $r$  and  $v = 0$  at some instant, the force direction is undefined. The simulation stops with an error.

To avoid singularities in the initial state of motion, be sure to set the bodies' initial conditions of position and velocity to physically sensible values.

## Dialog Box and Parameters

**Block Parameters: Body Spring & Damper**

Description

Models a damped linear oscillator between two Bodies, equivalent to a translational spring and damper. The force  $F$  between the bodies is projected along the axis connecting the Body coordinate systems and is a function of the relative displacement  $r$  and velocity  $v$  of these Body coordinate systems, given by  $F = -k(r-r_0) - b\dot{v}$ . The parameters  $r_0$ ,  $k$ , and  $b$  represent the spring's natural length, the spring constant, and the damper constant, respectively.

Parameters

Spring constant ( $k$ ):

Damper constant ( $b$ ):

Spring natural length ( $r_0$ ):

Units

Position:

Velocity:

Force:

OK Cancel Help Apply

# Body Spring & Damper

---

The dialog box has two active areas, **Parameters** and **Units**.

## Parameters

### Spring constant ( $k$ )

Enter the linear spring force constant  $k$ . The default is 0.

The units for  $k$  are derived implicitly from your choice of position and force units.

### Damper constant ( $b$ )

Enter the linear damping force constant  $b$ . The default is 0.

The units for  $b$  are derived implicitly from your choice of velocity and force units.

### Spring natural length ( $r_0$ )

Enter the spring's natural length (offset)  $r_0$ . The default is 0.

## Units

### Position

In the pull-down menu, choose units for the relative position vector  $\mathbf{r}$ . The default is m (meters).

### Velocity

In the pull-down menu, choose units for the relative velocity vector  $\mathbf{v}$ . The default is m/s (meters/second).

### Force

In the pull-down menu, choose units for the spring-damper force  $\mathbf{F}$  acting between the bodies. The default is N (Newtons).

## See Also

Body, Body Actuator, Body Sensor, Custom Joint, Ground, Joint Initial Condition Actuator, Joint Spring & Damper, Prismatic

See "Modeling Force Elements" on page 4-55.

**Purpose** Represent a composite joint with three translational and three rotational DoFs

**Library** Joints

**Description** The Bushing block represents a composite joint with three translational degrees of freedom (DoFs) as three prismatic primitives and three rotational DoFs as three revolute primitives. There are no constraints among the primitives. Unlike Six-DoF, Bushing represents the rotational DoFs as three revolutes, rather than as one spherical.



---

**Caution** A joint with three revolute primitives becomes singular if two or three of the rotation axes become parallel (“gimbal lock”). A joint with two or three prismatic primitives becomes singular if two or three of the translation axes become parallel. The simulation stops with errors in these cases.

---

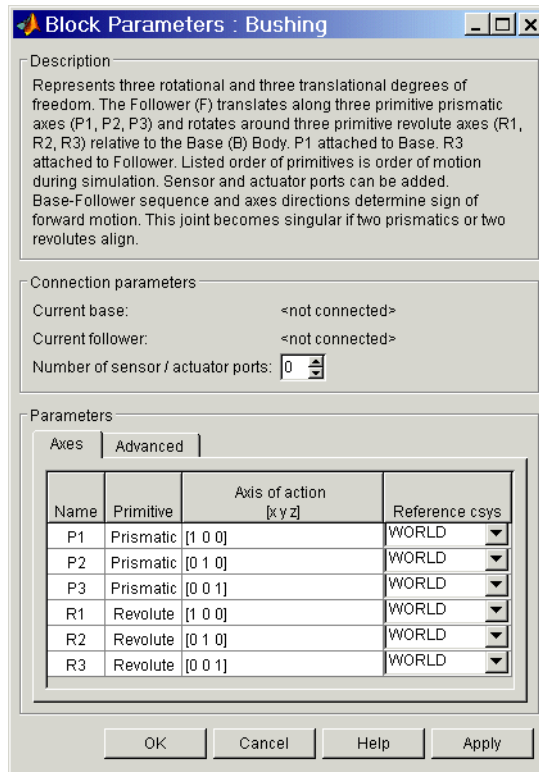
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Bushing block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.

# Bushing

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Bushing block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Bushing base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive P1 in the primitive list in **Parameters**.

## Current follower

When you connect the follower (F) connector port on the Bushing block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Bushing base and follower Body connector ports” figure.

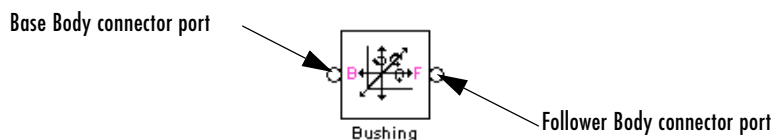
The follower Body is automatically connected to the last joint primitive R3 in the primitive list in **Parameters**.

## Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motions of prismatic and revolute primitives are specified in linear and angular units, respectively.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower moving around the rotational axis following the right-hand rule.



## Bushing base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Bushing has an entry line. These lines specify the direction of the axes of action of the DoFs that the Bushing represents.

### Name - Primitive

The primitive list states the names and types of joint primitives that make up the Bushing block: prismatic primitives P1, P2, P3, and revolute primitives R1, R2, R3.

# Bushing

---

## Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Prismatic: axis of translation
- Revolute: axis of rotation

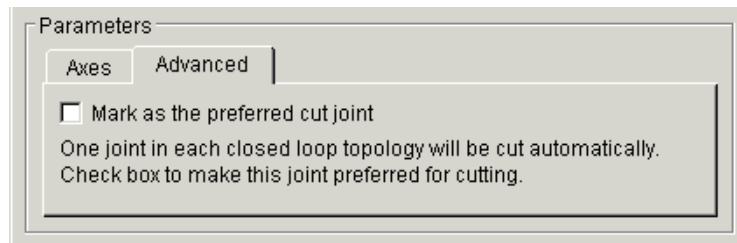
The default vectors are shown in the dialog box above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the revolute axes and no two of the prismatic axes can be parallel.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Bearing, Cylindrical, Gimbal, Prismatic, Revolute, Six-DoF



See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

# Connection Port

**Purpose** Create a Physical Modeling connector port for a subsystem

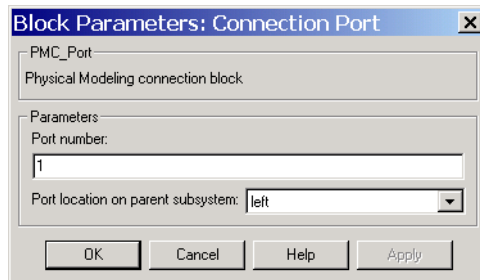
**Library** Utilities

**Description** The Connection Port block, placed inside a subsystem composed of SimMechanics blocks, creates a SimMechanics open round connector port ○ on the boundary of the subsystem. Once connected to a connection line, the Port becomes solid •.



You connect individual SimMechanics blocks and subsystems made of SimMechanics blocks to one another with SimMechanics connection lines, instead of normal Simulink signal lines. These are anchored at the open, round connector ports ○. Subsystems constructed out of SimMechanics blocks automatically have such open round connector ports. You can add additional connector ports by adding Connection Port blocks to your subsystem.

## Dialog Box and Parameters



### Port number

This field labels the subsystem connector port created by this block. Multiple connector ports on the boundary of a single subsystem require different numbers as labels. The default value for the first Port is 1.

### Port location on parent subsystem

Choose here on which side of the parent subsystem boundary the Port is placed. The choices are left or right. The default choice is left.

**See Also** In Simulink, see Creating Subsystems.

**Purpose** Measure constraint force/torque between a pair of constrained bodies

**Library** Sensors & Actuators

**Description** The Constraint & Driver Sensor block measures the force/torque of constraint (reaction force/torque) between a pair of bodies. You connect the block to the Constraint or Driver block connected between the two Bodies. The output signal is the reaction force/torque.

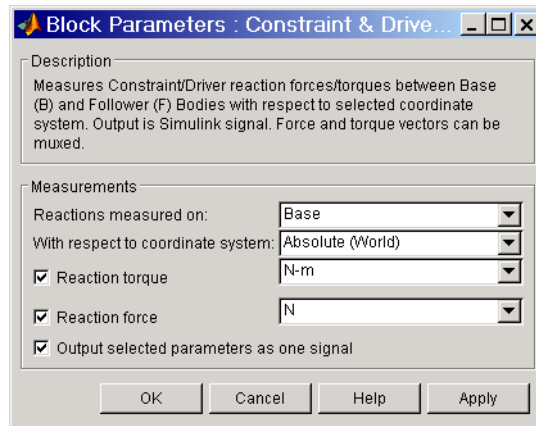


The Constraint & Driver Sensor measures the reaction force/torque in the reference coordinate system (CS) specified in the block dialog box. The Constraint or Driver block connects a base and a follower Body. You choose in the dialog box to measure the reaction force/torque on either the base or the follower Body.

The input is the connector port connected to the Constraint or Driver block you want to sense. The output is a set of Simulink signals or one bundled Simulink signal of the reaction force/torque vector(s).

A body's orientation rotation matrix  $R$  relates vector components measured in the body CS and in the inertial World CS by  $[R] \cdot \mathbf{v}_b = \mathbf{v}_s$ . The column vector  $\mathbf{v}_b$  lists the vector  $\mathbf{v}$ 's three components measured in the body CS. The column vector  $\mathbf{v}_s$  lists the vector  $\mathbf{v}$ 's three components measured in the World CS.

## Dialog Box and Parameters



# Constraint & Driver Sensor

---

The dialog box has one active area, **Measurements**.

## **Measurements**    **Reactions measured on**

In the pull-down menu, choose to measure the reaction force/torque on the base (B) or follower (F) Body. The default is Base.

## **With respect to coordinate system**

In the pull-down menu, choose the CS in which the reaction force/torque or motion is interpreted. The default is Absolute (World).

In the Absolute case, the force vectors have components measured relative to the inertial World CS axes. In the Local case, the same force vector signals are premultiplied by the inverse rotation matrix  $R^{-1} = R^T$  for the Body selected in **Reactions measured on**.

## **Reaction torque**

Select the check box if you want to measure the reaction torque. The default is selected. The torque is a row vector in the Simulink output signal.

In the pull-down menu, choose the units for the reaction torque. The default is N-m (Newton-meters).

## **Reaction force**

Select the check box if you want to measure the reaction force. The default is selected. The force is a row vector in the Simulink output signal.

In the pull-down menu, choose the units for the reaction force. The default is N (Newtons).

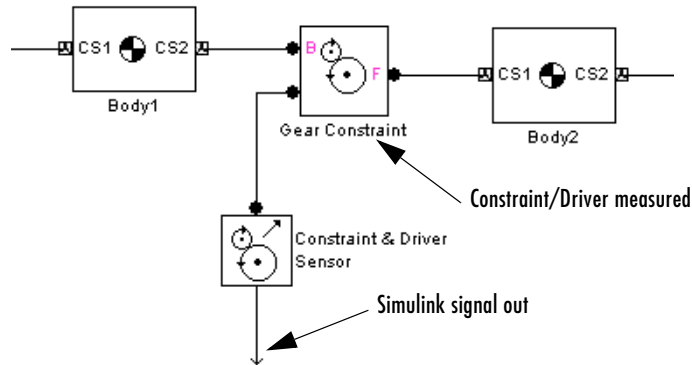
## **Output selected parameters as one signal**

Select this check box to convert the output signals into a single bundled signal. The default is selected. If you unselect it, the Constraint & Driver Sensor block will grow as many Simulink outports as there are active signals selected, in the same order top to bottom, in the dialog box.

If the check box is selected, the Simulink signal out has all the active signals bundled into a single row vector, ordered in the order shown in the dialog box. The type of the signal components depends on which measurements are active (selected).

## Example

Here is a Constraint & Driver Sensor connected to a Gear Constraint, which connects and constraints two Bodies:



You must add a Sensor port (connector port) to the Constraint/Driver block to connect the Constraint & Driver Sensor to it. The base (B)-follower (F) Body sequence on the two sides of the Joint determines the sense of the Constraint & Driver Sensor data.

## See Also

Body Sensor, Driver Actuator, Joint Sensor, Mechanical Branching Bar

See “Representing Body Positions and Orientations” on page 3-2 and “Modeling Sensors” on page 4-60.

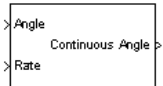
In Simulink, see the Signal Routing Library and the Sinks Library.

# Continuous Angle

**Purpose** Convert discontinuous, bounded angular output from a sensor to continuous, unbounded angular output

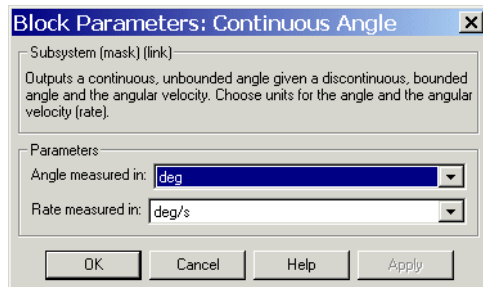
**Library** Utilities

**Description** The Continuous Angle block converts a measured angle signal restricted to the semi-open interval  $(-180^\circ, +180^\circ]$  degrees or  $(-\pi, +\pi]$  radians to a continuous, unbounded angle not restricted to any interval. This block requires the angle and the angular velocity as input signals. The continuous, unbounded angle is the output signal.



The Joint Sensor block outputs the absolute rotational measurement of revolute motion as a bounded angle in the interval  $(-180^\circ, +180^\circ]$  degrees or  $(-\pi, +\pi]$  radians. Motion that crosses the boundaries of this interval causes discontinuities in the measured angle, from  $+180^\circ$  to  $-180^\circ$  or vice versa. Use the Continuous Angle block if you want to convert this restricted angular measurement to an unbounded measurement.

## Dialog Box and Parameters



The dialog box has one active area, **Parameters**.

## Parameters

### Angle measured in

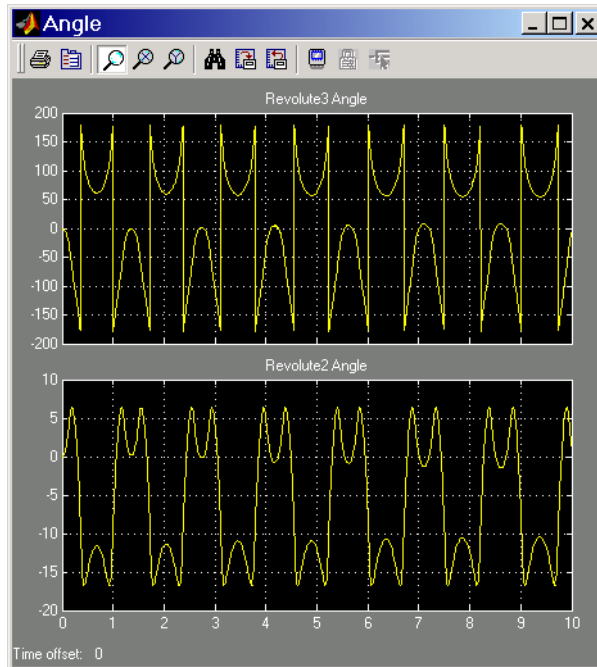
Choose the units for the input angle and the output continuous angle, either deg (degrees) or rad (radians). The default is deg.

### Rate measured in

Choose the units for the input rate (angular velocity), either deg/s (degrees/second) or rad/s (radians/second).

## Example

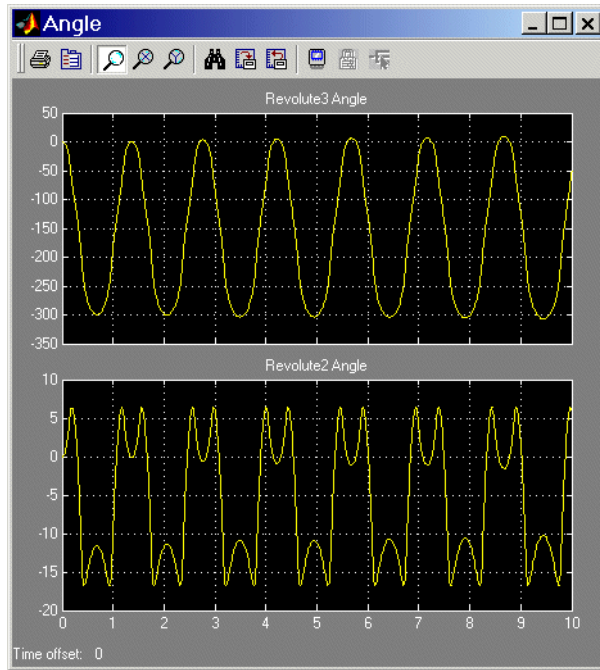
The tutorial “A Four Bar Mechanism” in the “Building and Visualizing Simple Machines” chapter produces this angular motion output for the Revolute3 and Revolute 2 joints:



The Revolute3 angle is restricted to the interval  $(-180^\circ, +180^\circ]$ , so values passing either limit of this interval are mapped to the opposite end of the interval. The Revolute2 angle is not restricted, but instead touches genuine turning points in its motion.

After passing the angles and angular velocities through Continuous Angle blocks, the Revolute3 angular motion appears different:

# Continuous Angle



Revolute3's motion is unchanged, but its angle is now continuous, with no interval restriction. Revolute2's angle is unchanged.

## See Also

Joint Sensor

See "Additional Useful Blocks" on page 8-7.



**Purpose** Represent a customizable composite joint with up to three translational and up to three rotational degrees of freedom

**Library** Joints

## Description



The Custom Joint block is a composite joint that you can customize with a specified combination of primitives (prismatic, revolute, or spherical) representing the most general and unconstrained degrees of freedom (DoFs) in three dimensions:

- Up to three translational DoFs as three prismatic primitives
- Up to three rotational DoFs:
  - As a single spherical primitive
  - As one, two, or three revolute primitives

The sense of rotational DoFs is defined by the right-hand rule. One spherical or three revolutes together form a right-handed system.

You can add, configure, and delete these primitives from the Custom Joint, with a minimum and default of one primitive. The properties of each primitive are the same as the individual Joints of the same names.

---

**Caution** A joint with two or three revolute primitives becomes singular if two or three of the rotation axes become parallel (“gimbal lock”). A joint with two or three prismatic primitives becomes singular if two or three of the translation axes become parallel. The simulation stops with errors in these cases.

---

The Custom Joint block’s primitives are assembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, and the origins of these Body CSs must be spatially collocated points, within assembly tolerances.

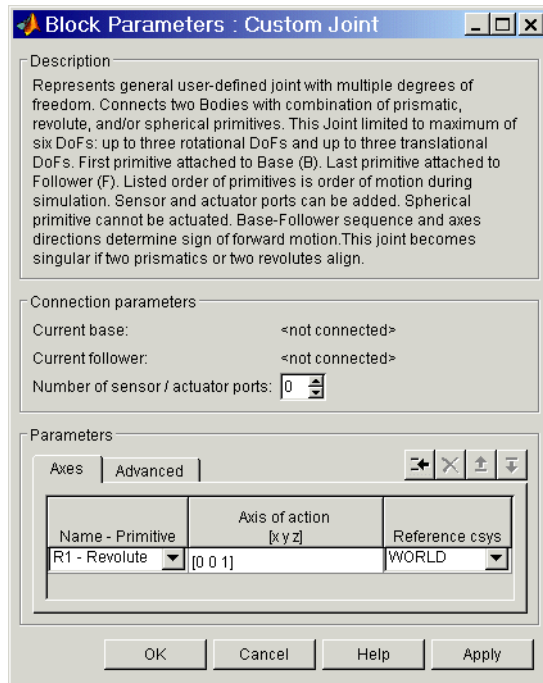
You can connect Actuator and Sensor blocks to a Custom Joint, with each Actuator and Sensor connecting to an individual primitive. You cannot connect an Actuator to a spherical primitive.

# Custom Joint

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

Any Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axes.

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Custom Joint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Custom Joint base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive in the primitive list in **Parameters**.

## Current follower

When you connect the follower (F) connector port on the Custom Joint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Custom Joint base and follower Body connector ports” figure.

The follower Body is automatically connected to the last joint primitive in the primitive list in **Parameters**.

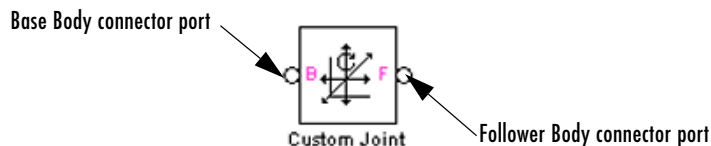
## Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0. A spherical primitive cannot be connected to an Actuator.

The motion of a prismatic primitive is specified in linear units. The motion of a revolute primitive is specified in angular units. The motion of a spherical primitive is three DoFs specified in quaternion form.

The base (B)-follower (F) Body sequence determines the sense of positive motion:

- Positive translation is the follower moving in the direction of the translation axis.
- Positive rotation is the follower rotating in the right-handed sense about the rotation axis.
- Positive rotation is the follower rotating in the right-handed sense as shown by the motion figure in the Spherical block reference page.



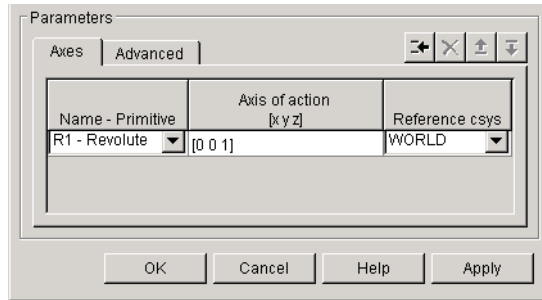
## Custom Joint base and follower Body connector ports

### Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

# Custom Joint

The entries on the **Axes** pane are required. Each DoF primitive in Custom Joint has an entry line. These lines specify the direction of the axes of action of the DoFs that the Custom Joint represents.



## Name - Primitive

In the pull-down menu, select a label and primitive type for this DoF. Up to three prismatic primitives P1, P2, P3 are allowed. The rotational DoFs are represented by up to three revolute primitives: R1, R2, R3. A spherical primitive S can take all three allowed rotational DoFs instead.

The default value is R1 - Revolute.

## Axis of action [x y z]

Enter here as a three-component vector the directional axis defining the allowed motion of this primitive and its corresponding DoF:

- Prismatic: axis of translation
- Revolute: axis of rotation
- Spherical: field is not active

The default vector is [0 0 1]. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the revolute axes and no two of the prismatic axes can be parallel.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute

meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The field is not active for a spherical primitive.

## Managing the Joint Primitives List in a Custom Joint

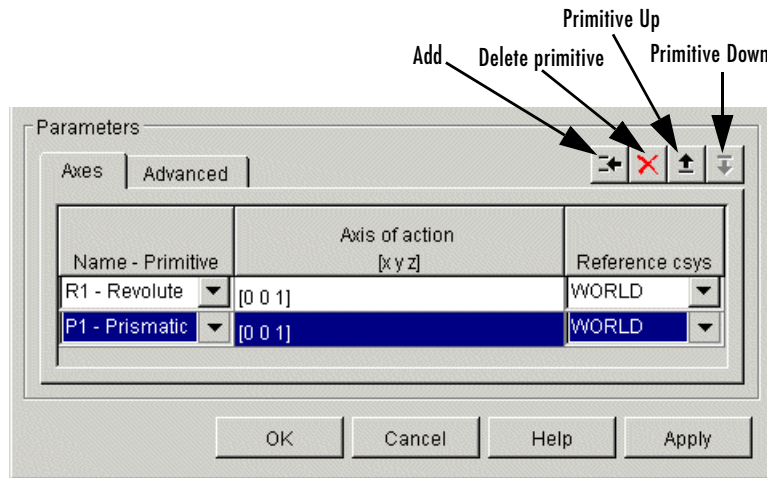
The Custom Joint primitives list controls (see the following “Custom Joint primitives list controls” figure) allow you to add, reorder, and delete joint primitives in a Custom Joint block:

- To add a joint primitive to the primitives list:
  - Highlight an existing primitive name in the list.
  - Click on the **Add** button (see the following “Custom Joint primitives list controls” figure).

A new primitive will appear immediately below the primitive you highlighted.
- To change the position of a joint primitive in the list:
  - Highlight the primitive whose position you want to change.
  - Click on the **Up** or **Down** button (see the following “Custom Joint primitives list controls” figure) until the primitive is where you want it.
- To delete a joint primitive from the list:
  - Highlight the primitive you want to delete.
  - Click on the **Delete** button (see the following “Custom Joint primitives list controls” figure).

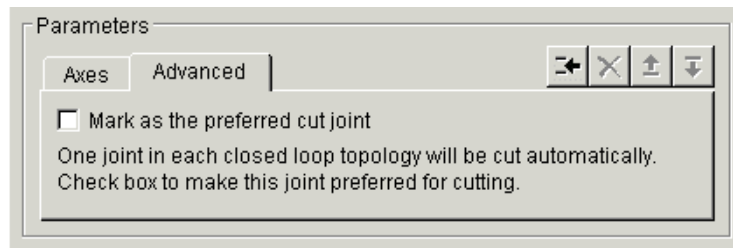
The primitive you highlighted disappears.
- Custom Joint requires at least one primitive, which you cannot delete.

# Custom Joint



## Custom Joint primitives list controls

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Bushing, Gimbal, Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator, Prismatic, Revolute, Six-DoF, Spherical

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

# Cylindrical

---

## Purpose

Represent a composite joint with one translational DoF and one rotational DoF, with parallel translation and rotation axes

## Library

Joints

## Description

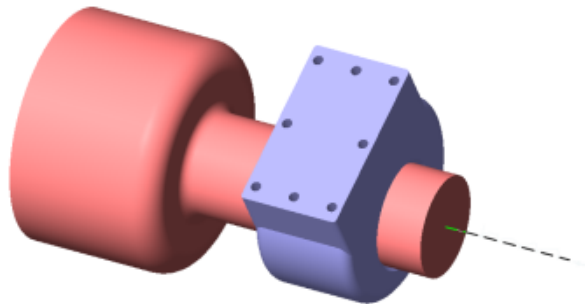


The Cylindrical block represents a composite joint with one translational degree of freedom (DoF) as one prismatic primitive and one rotational DoF as one revolute primitive. The translation and rotation axes must be parallel.

You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Cylindrical block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

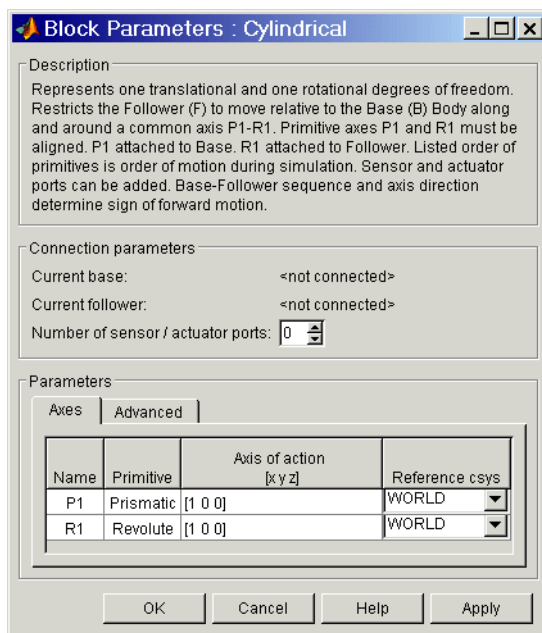
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.





## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Cylindrical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Cylindrical base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive P1 in the primitive list in **Parameters**.

### Current follower

When you connect the follower (F) connector port on the Cylindrical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Cylindrical base and follower Body connector ports” figure.

# Cylindrical

---

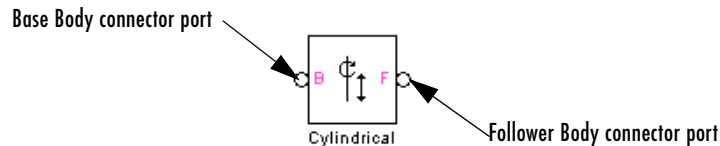
The follower Body is automatically connected to the last joint primitive R1 in the primitive list in **Parameters**.

## Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motions of prismatic and revolute primitives are specified in linear and angular units, respectively.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower moving around the rotational axis following the right-hand rule.



## Cylindrical base and follower Body connector ports

### Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Cylindrical has an entry line. These lines specify the direction of the axes of action of the DoFs that the Cylindrical represents.

#### Name - Primitive

The primitive list states the names and types of joint primitives that make up the Cylindrical block: prismatic revolute P1 and revolute primitive R1.

#### Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Prismatic: axis of translation
- Revolute: axis of rotation

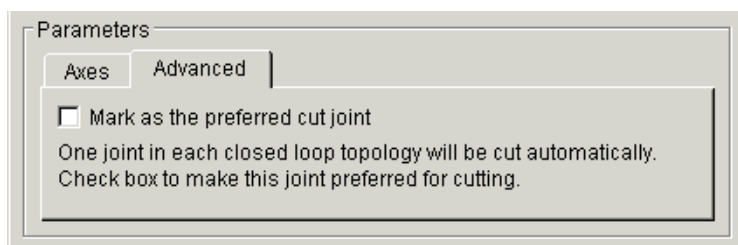
The default vectors are shown in the dialog box above. The axes are directed vectors whose overall sign matters.

The two axes P1 and R1 in Cylindrical must be aligned.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Disassembled Cylindrical, Prismatic, Revolute, Screw

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

# Disassembled Cylindrical

---

**Purpose** Represent a disassembled cylindrical joint, with one translational DoF and one rotational DoF along and about misaligned axes, with no constraints

**Library** Joints/Disassembled Joints

## Description



The Disassembled Cylindrical block represents a composite joint, one translational and one rotational degrees of freedom (DoF) along and about a pair of specified misaligned axes between two bodies. SimMechanics automatically assembles (aligns) the two axes at simulation start.

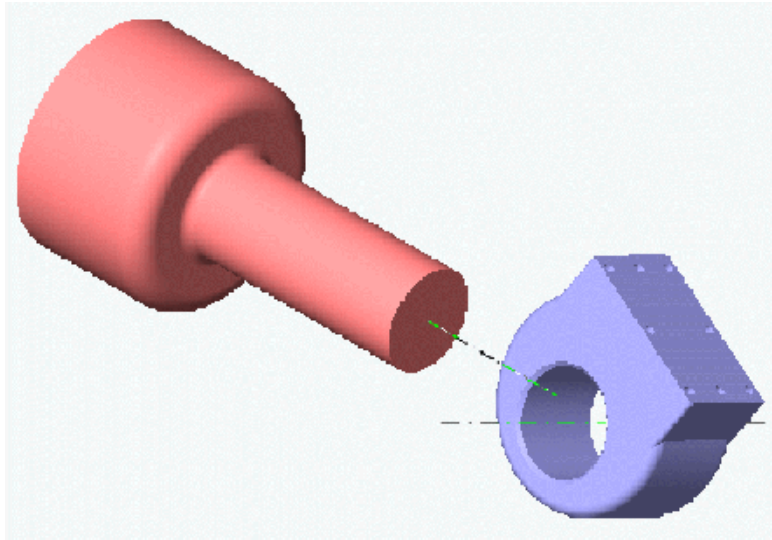
A cylindrical joint is composite, with two DoFs and a single specified axis: a prismatic primitive translating along the axis, and a revolute primitive rotating about the axis. There are no constraints between the two primitives.

This block is disassembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, but the origins of these Body CSs do not need to be spatially collocated points or lie along a joint axis.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies. The disassembled translation-rotation joint axes are associated with the base and follower Bodies, respectively.

# Disassembled Cylindrical

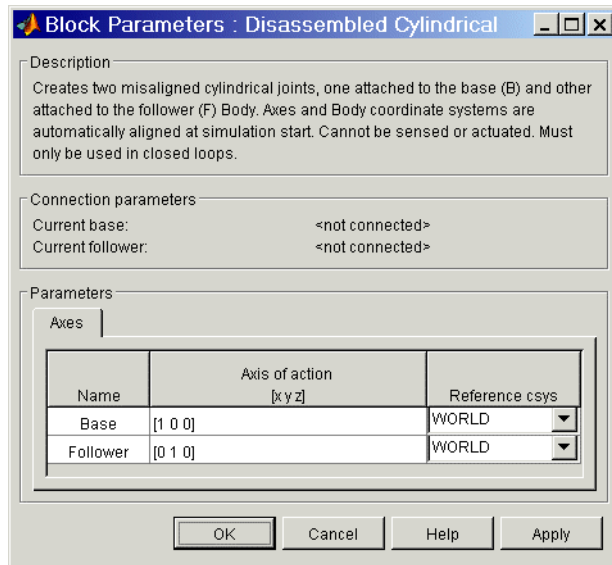
You can only use a Disassembled Joint block to close a loop. One loop must have no more than one disassembled joint. You cannot connect an Actuator or Sensor to a Disassembled Joint.



**Disassembled cylindrical axes of follower (blue) and base (red)**

# Disassembled Cylindrical

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

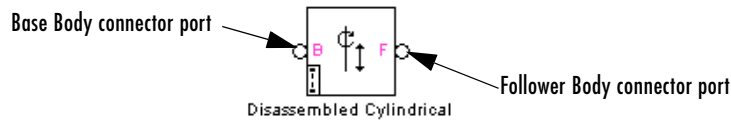
## Connection Parameters

### Current base

When you connect the base (B) connector port on the Disassembled Cylindrical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Disassembled Cylindrical base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Disassembled Cylindrical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Disassembled Cylindrical base and follower Body connector ports” figure.

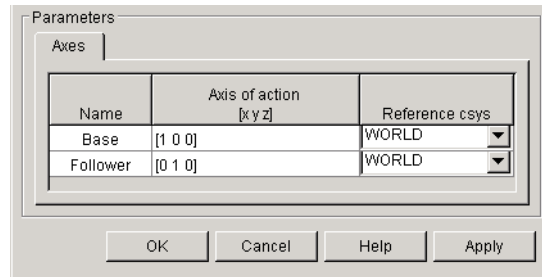


## Disassembled Cylindrical base and follower Body connector ports

### Parameters

There is one **Axes** panel.

The entries on the **Axes** pane are required. They specify the directions of the two misaligned axes of the translational-rotational DoFs that the Disassembled Cylindrical represents.



### Name

This column automatically displays the names of the two misaligned rotation axes attached to base and follower bodies, respectively.

### Axis of action [x y z]

Enter here as two three-component vectors the two misaligned directional axes along and about which the base and follower bodies respectively can translate and rotate. The default vectors are [1 0 0] and [0 1 0], respectively. The axes are directed vectors whose overall signs matter.

### Reference csys

Using the pull-down menu, choose the coordinate systems (World, the base Body CS, or the follower Body CS) whose coordinate axes the two vector axes of translation-rotation are oriented with respect to. The defaults are WORLD.

# Disassembled Cylindrical

---

## **See Also**

Cylindrical, Disassembled Prismatic, Disassembled Revolute, Disassembled Spherical

See “Modeling Joints” on page 4-17 for more on representing DoFs with Disassembled Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting disassembled joints.



**Purpose** Represent a disassembled prismatic joint with one translational degree of freedom along misaligned axes

**Library** Joints/Disassembled Joints

**Description**

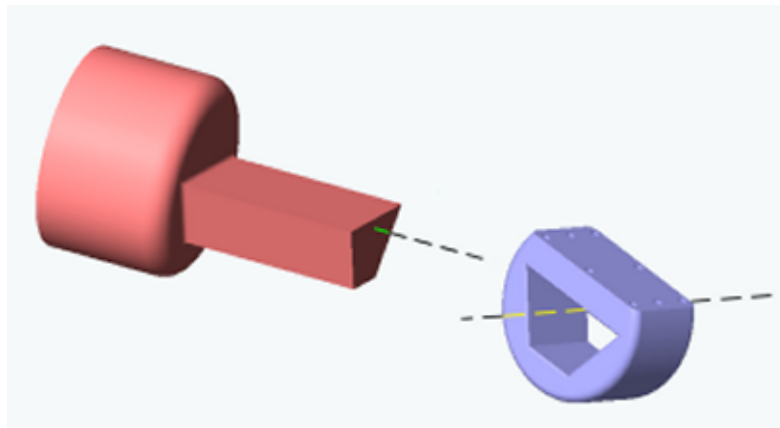


The Disassembled Prismatic block represents a single translational degree of freedom (DoF) along a pair of specified misaligned axes between two bodies. SimMechanics automatically assembles (aligns) the translation axes at simulation start.

This block is disassembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, but the origins of these Body CSs do not have to lie along a joint axis. As with the Prismatic Joint, these Body CS origins do not need to be spatially collocated points either.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies. The disassembled joint axes are associated with the base and follower Bodies, respectively.

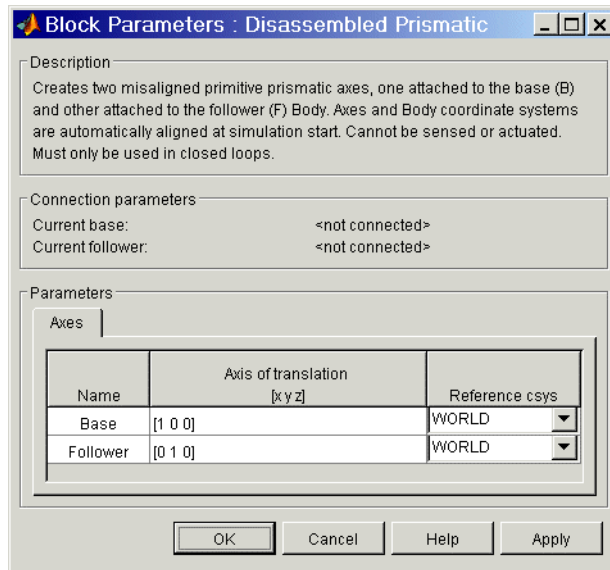
You can only use a disassembled Joint block to close a loop. One loop must have no more than one disassembled joint. You cannot connect an Actuator or Sensor to a Disassembled Joint.



**Disassembled prismatic axes of follower (blue) and base (red)**

# Disassembled Prismatic

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

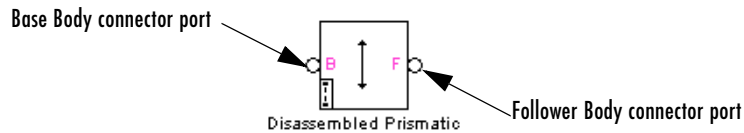
## Connection Parameters

### Current base

When you connect the base (B) connector port on the Disassembled Prismatic block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Disassembled Prismatic base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Disassembled Prismatic block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Disassembled Prismatic base and follower Body connector ports” figure.

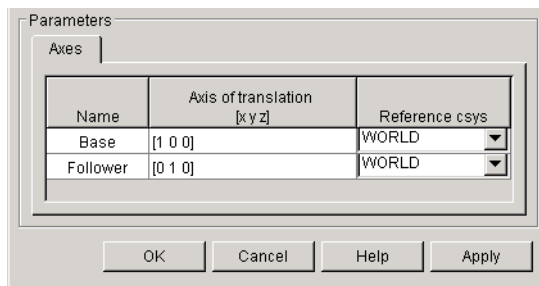


## Disassembled Prismatic base and follower Body connector ports

### Parameters

There is one **Axes** panel.

The entries on the **Axes** pane are required. They specify the directions of the two misaligned axes of the translational DoF that the Disassembled Prismatic represents.



### Name

This column automatically displays the names of the two misaligned translation axes attached to base and follower bodies, respectively.

### Axis of translation [x y z]

Enter here as two three-component vectors the two misaligned directional axes along which the base and follower bodies respectively can translate. The default vectors are [1 0 0] and [0 1 0], respectively. The axes are directed vectors whose overall signs matter.

### Reference csys

Using the pull-down menu, choose the coordinate systems (World, the base Body CS, or the follower Body CS) whose coordinate axes the two vector axes of translation are oriented with respect to. The defaults are WORLD.

# Disassembled Prismatic

---

## **See Also**

Disassembled Cylindrical, Disassembled Revolute, Disassembled Spherical, Prismatic

See “Modeling Joints” on page 4-17 for more on representing DoFs with Disassembled Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closing loops with disassembled joints.

## Purpose

Represent a disassembled revolute joint with one rotational degree of freedom about misaligned axes

## Library

Joints/Disassembled Joints

## Description

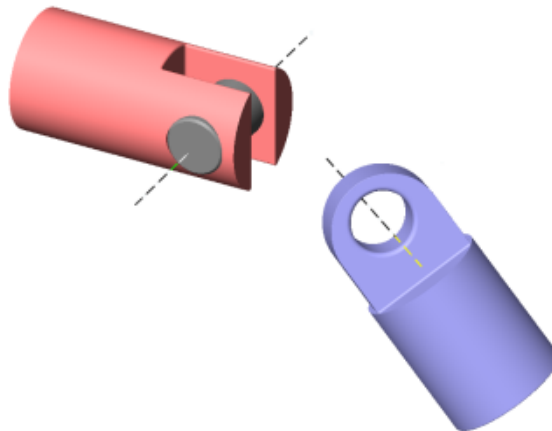


The Disassembled Revolute block represents a single rotational degree of freedom (DoF) along a pair of specified misaligned axes between two bodies. SimMechanics automatically assembles (aligns) the rotation axes at simulation start.

This block is disassembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, but the origins of these Body CSs do not need to be spatially collocated points or lie along a joint axis.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies. The disassembled joint axes are associated with the base and follower Bodies, respectively.

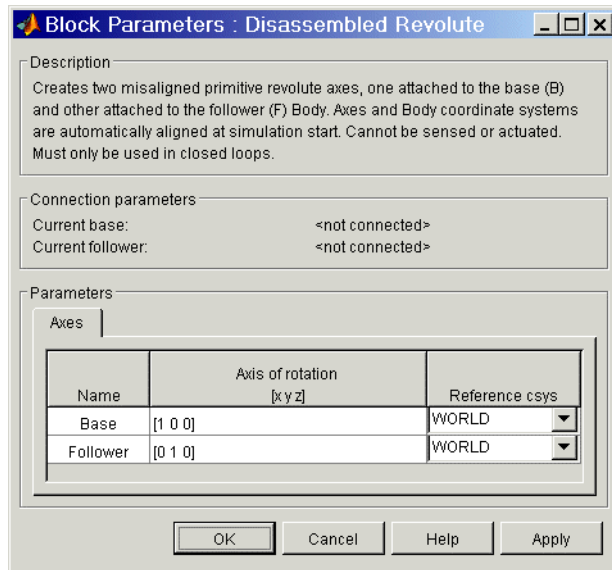
You can only use a disassembled Joint block to close a loop. One loop must have no more than one disassembled joint. You cannot connect an Actuator or Sensor to a Disassembled Joint.



**Disassembled revolute axes of follower (blue) and base (red)**

# Disassembled Revolute

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

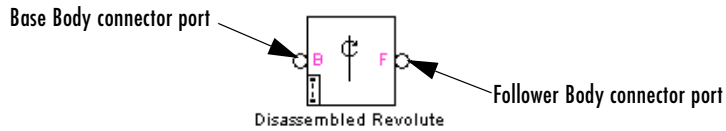
## Connection Parameters

### Current base

When you connect the base (B) connector port on the Disassembled Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Disassembled Revolute base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Disassembled Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Disassembled Revolute base and follower Body connector ports” figure.

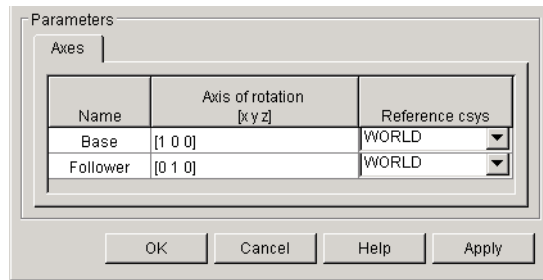


## Disassembled Revolute base and follower Body connector ports

### Parameters

There is one **Axes** panel.

The entries on the **Axes** pane are required. They specify the directions of the two misaligned axes of the rotational DoF that the Disassembled Revolute represents.



### Name

This column automatically displays the names of the two misaligned rotation axes attached to base and follower bodies, respectively.

### Axis of rotation [x y z]

Enter here as two three-component vectors the two misaligned directional axes about which the base and follower bodies respectively can rotate. The default vectors are [1 0 0] and [0 1 0], respectively. The axes are directed vectors whose overall signs matter.

### Reference csys

Using the pull-down menu, choose the coordinate systems (World, the base Body CS, or the follower Body CS) whose coordinate axes the two vector axes of rotation are oriented with respect to. The defaults are WORLD.

# Disassembled Revolute

---

## **See Also**

Disassembled Cylindrical, Disassembled Prismatic, Disassembled Spherical, Revolute

See “Modeling Joints” on page 4-17 for more on representing DoFs with Disassembled Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closing loops with disassembled joints.



**Purpose** Represent a disassembled spherical joint with three rotational degrees of freedom about dislocated pivots

**Library** Joints/Disassembled Joints

## Description



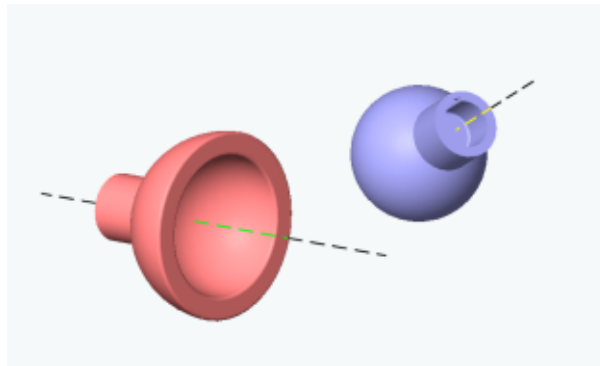
The Disassembled Spherical block represents three rotational degrees of freedom (DoF) about a pair of specified dislocated pivots at the two bodies, separated “ball-in-socket” joints. SimMechanics automatically assembles (collocates) the spherical pivots at simulation start.

Two rotational DoFs specify a directional axis, and a third rotational DoF specifies rotation about that directional axis. (See the motion figure in the Spherical block reference page.) The sense of each rotational DoF is defined by the right-hand rule. Unlike the Gimbal block, the Disassembled Spherical block cannot become singular.

This block is disassembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, but the origins of these Body CSs (the dislocated pivots) do not need to be spatially collocated points.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies. The disassembled joint pivots are associated with the base and follower Bodies, respectively.

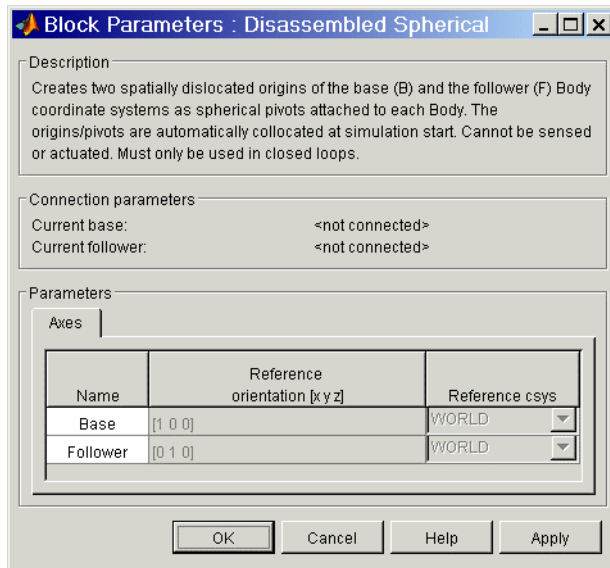
You can only use a disassembled Joint block to close a loop. One loop must have no more than one disassembled joint. You cannot connect an Actuator or Sensor to a Disassembled Joint.



**Disassembled spherical pivots of follower (blue) and base (red)**

# Disassembled Spherical

## Dialog Box and Parameters



The dialog box has one active area, **Connection parameters**. The other area, **Parameters**, is inactive.

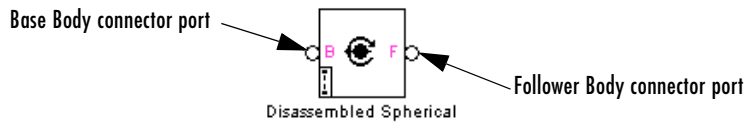
## Connection Parameters

### Current base

When you connect the base (B) connector port on the Disassembled Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Disassembled Spherical base and follower Body connector ports” figure.

### Current follower

When you connect the base (F) connector port on the Disassembled Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Disassembled Spherical base and follower Body connector ports” figure.

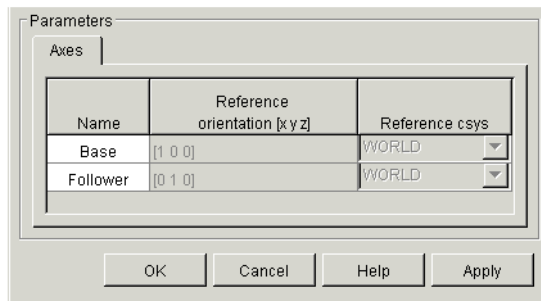


## Disassembled Spherical base and follower Body connector ports

### Parameters

There is one **Axes** panel.

The entries on the **Axes** pane are automatic. They specify the dislocated pivot points of the spherical DoF that the Disassembled Spherical represents.



### Name

This column automatically displays the names of the two dislocated pivot points attached to base and follower bodies, respectively.

### Reference orientation [x y z]

This field is not active.

### Reference csys

This field is not active.

### See Also

Disassembled Cylindrical, Disassembled Prismatic, Disassembled Revolute, Gimbal, Spherical

See “Modeling Joints” on page 4-17 for more on representing DoFs with Disassembled Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closing loops with disassembled joints.

# Distance Driver

---

**Purpose** Specify the distance between two Body CS origins as a function of time

**Library** Constraints & Drivers

## Description



The Distance Driver block drives the distance between the origins of two Body coordinate system (CS) as a function of time that you specify. This function must always remain nonnegative during the simulation.

Let  $\mathbf{r}_1$ ,  $\mathbf{r}_2$  be the vector positions of the origins of CS1 on one Body and CS2 on the other Body, respectively. These vectors can be measured in any CS. The Distance Driver specifies the scalar distance  $|\mathbf{r}_1 - \mathbf{r}_2|$  between these points as a function of time:

$$|\mathbf{r}_1 - \mathbf{r}_2| = f(t).$$

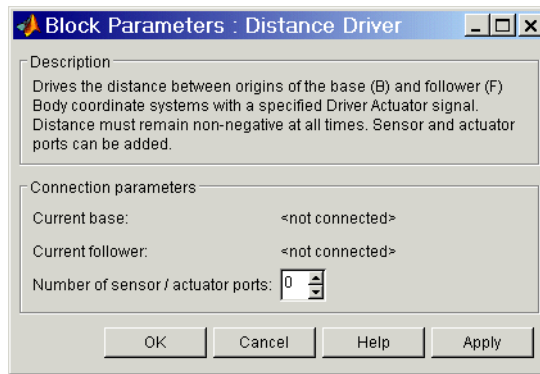
You connect the Distance Driver to a Driver Actuator block.

The Simulink input signal into the Driver Actuator specifies the time-dependent driving function  $f(t)$  and its first two derivatives, as well as their units. If you do not actuate Distance Driver, this block acts as a time-independent constraint that freezes the distance between the two Body CSs at its initial value during the simulation.

Drivers restrict relative degrees of freedom (DoFs) between a pair of bodies as specified functions of time. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Driver blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Drivers are ordered as base and follower, fixing the direction of relative motion.

You can also connect a Constraint & Driver Sensor to any Driver and measure the reaction forces/torques between the driven bodies.

## Dialog Box and Parameters



The dialog box has one active area, **Connection parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Distance Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Distance Driver base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Distance Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Distance Driver base and follower Body connector ports” figure.

### Number of sensor/actuator ports

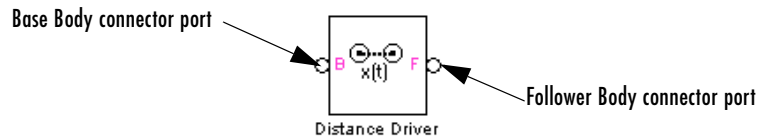
Using this spinner menu, you can set the number of extra connector ports needed for connecting Driver Actuator and Constraint & Driver Sensor blocks to this Driver. The default is 0.

To activate the Driver, connect a Driver Actuator.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.

# Distance Driver

---



## Distance Driver base and follower Body connector ports

### See Also

Constraint & Driver Sensor, Driver Actuator, Linear Driver, Weld

See “Modeling Constraints and Drivers” on page 4-34 for more on restricting DoFs with Drivers.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on using drivers in closed loops.

See “Constraining and Driving Motion” on page 8-6.

**Purpose**

Apply relative motion between a pair of constrained bodies through a driver

**Library**

Sensors & Actuators

**Description**



The Driver Actuator block actuates a Driver block connected between a pair of bodies. You connect the block to the Driver block connected between the Bodies. The Driver block represents a time-dependent (rheonomic) constraint on a relative degree of freedom (DoF) between the two bodies. A Driver requires a time-dependent function to specify the relative position, velocity, and acceleration of the connected Bodies. The output of the Driver Actuator is this time-dependent function  $f(t)$  and its first two derivatives.

You specify these three functions as a bundled Simulink input signal. This signal can be any Simulink signal, including a signal feedback from a Sensor block, satisfying these conditions:

- The signal must consist of three bundled components.
  - *Exception:* The Velocity Driver requires two bundled components.
- The three components must be ordered  $(f(t), df(t)/dt, d^2f(t)/dt^2)$ .
  - *Exception:* The Velocity Driver requires a function and its derivative  $(f(t), df(t)/dt)$ .
- Each successive signal component must be the time derivative of the previous component.
- The specific meaning of  $f(t)$  depends on the connected Driver block being actuated. Select the specific Driver block for details.

Linear Motions	Angular Motions
Distance Driver	Angle Driver
Linear Driver	
Velocity Driver	

The Driver connects a base (B) and a follower (F) Body. The base–follower sequence determines the sense of the actuation signal. The inport is the Simulink input signal. The output is the connector port you connect to the Driver block you want to actuate.

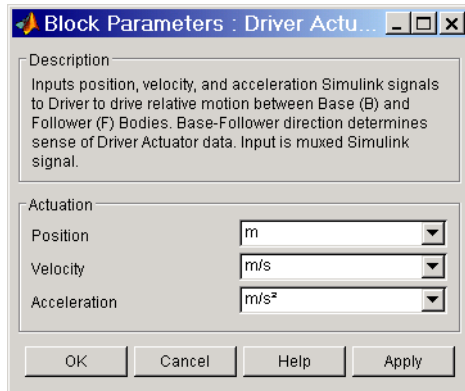
# Driver Actuator

---

**Note** You do not have to connect a Driver Actuator to a Driver block. If you do not actuate a Driver, the Driver block acts as a time-independent constraint that freezes the driven relative DoF between the Bodies at its initial value during the simulation.

---

## Dialog Box and Parameters



The dialog box has one active area, **Actuation**.

### Actuation

#### Position

In the pull-down menu, choose the units of the actuating  $f(t)$  you apply to the relative motion of the bodies.

If the Driver Actuator is connected to a Velocity Driver, this field does not appear.

#### Velocity

In the pull-down menu, choose the units of the actuating  $df(t)/dt$  you apply to the relative motion of the bodies.

If the Driver Actuator is connected to a Velocity Driver, this field refers to  $f(t)$ .



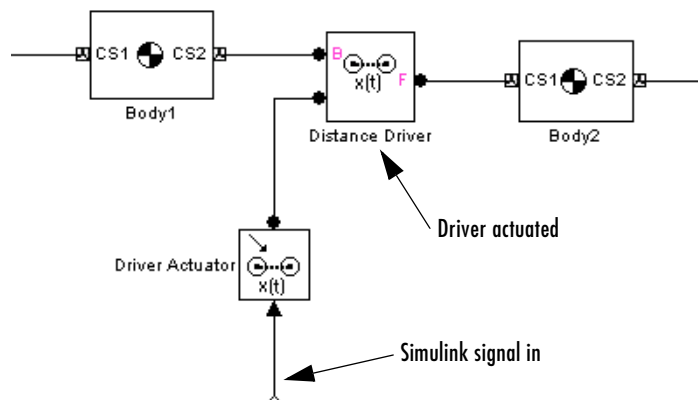
## Acceleration

In the pull-down menu, choose the units of the actuating  $d^2f(t)/dt^2$  you apply to the relative motion of the bodies.

If the Driver Actuator is connected to a Velocity Driver, this field refers to  $df(t)/dt$ .

## Example

Here is a Driver Actuator connected to a Distance Driver, which connects two Bodies:



You must add an Actuator port (connector port) to the Driver block to connect the Driver Actuator to it. The base (B)-follower (F) Body sequence on the two sides of the Driver determines the sense of the Driver Actuator data.

The Driver Actuator drives the relative motion between the two Bodies connected to the Driver. The nature of the connected Driver block determines the exact meaning of the actuation data, including the choice of units.

## See Also

Body Actuator, Constraint & Driver Sensor, Joint Actuator, Mechanical Branching Bar

See “Constraining and Driving Motion” on page 8-6.

In Simulink, see the Signal Routing Library and the Sources Library.

# Gear Constraint

---

**Purpose** Constrain the rotational motion of two bodies to move along tangent pitch circles

**Library** Constraints & Drivers

## Description



The two Bodies connected by a Gear Constraint block are each restricted to turn relative to another along pitch circles centered at each body. The pitch circle centers are the origins of the Body coordinate systems (CSs) at which the Gear Constraint block is connected on either side. The pitch circles are tangent at one contact point.

Let  $\mathbf{r}_1$ ,  $\mathbf{r}_2$  be the radius vectors of the two pitch circles and  $\omega_1$ ,  $\omega_2$  the angular velocity vectors of the two bodies. The Gear Constraint requires that:

$$\omega_1 \times \mathbf{r}_1 = \omega_2 \times \mathbf{r}_2.$$

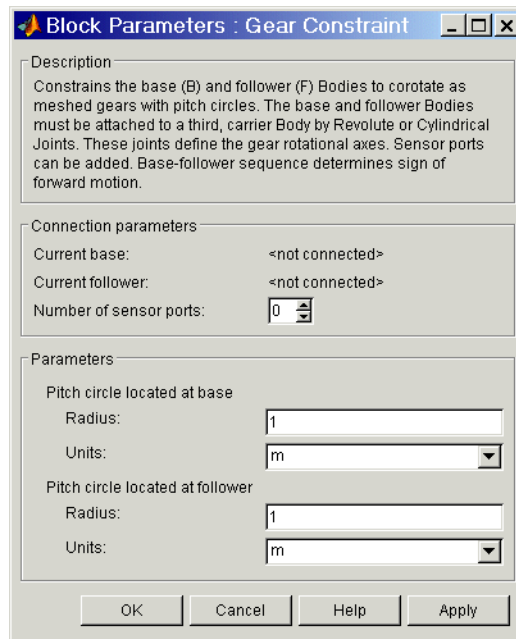
You specify the scalar radii  $r_1$ ,  $r_2$  of the pitch circles.

You must also connect the two Bodies connected by a Gear Constraint to a third, carrier Body by Revolute or Cylindrical Joints. (The third carrier body can be ground, but you must use two Ground blocks in this case, because a Ground has only one Body CS port. Both Grounds represent the same immobile body.) The constrained pair of Bodies rotate relative to one another about distinct rotational axes defined by the angular velocity vectors  $\omega_1$ ,  $\omega_2$ . These axes do not have to be parallel.

Constraints restrict relative degrees of freedom (DoFs) between a pair of bodies. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Constraint blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Constraint are ordered as base and follower, fixing the direction of relative motion.

You can connect a Constraint & Driver Sensor to a Constraint block, but not a Driver Actuator. The Constraint & Driver Sensor measures the reaction forces/torques between the constrained bodies.

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Gear Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Gear Constraint base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Gear Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Gear Constraint base and follower Body connector ports” figure.

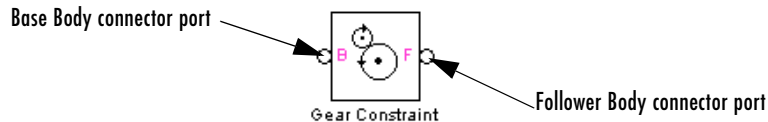
# Gear Constraint

---

## Number of sensor ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Constraint & Driver Sensor blocks to this Constraint. The default is 0.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-handed sense about the rotation axis.



## Gear Constraint base and follower Body connector ports

### Parameters

#### Pitch circle located at base

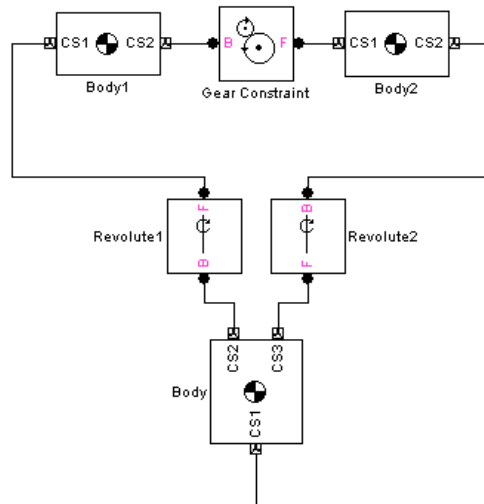
Enter **Radius** and **Units** for the pitch circle centered at base Body CS. The defaults are 1 and m (meters).

#### Pitch circle located at follower

Enter **Radius** and **Units** for the pitch circle centered at follower Body CS. The defaults are 1 and m (meters).

## Example

A simple example of a valid part of a model with a Gear Constraint:



The Body CS origins CS2@Body1 and CS1@Body2 must be separated and oriented in such a way that the gear pitch circles are in contact and tangent at one point.

## See Also

Body, Constraint & Driver Sensor, Cylindrical, Ground, Revolute

See “Modeling Constraints and Drivers” on page 4-34 for more on restricting DoFs with Constraints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on using constraints in closed loops.

See “Constraining and Driving Motion” on page 8-6.

# Gimbal

---

**Purpose** Represent a composite joint with three rotational DoFs

**Library** Joints

**Description** The Gimbal block represents a composite joint with three rotational degrees of freedom (DoFs) as three revolute primitives. There are no constraints among the primitives.



---

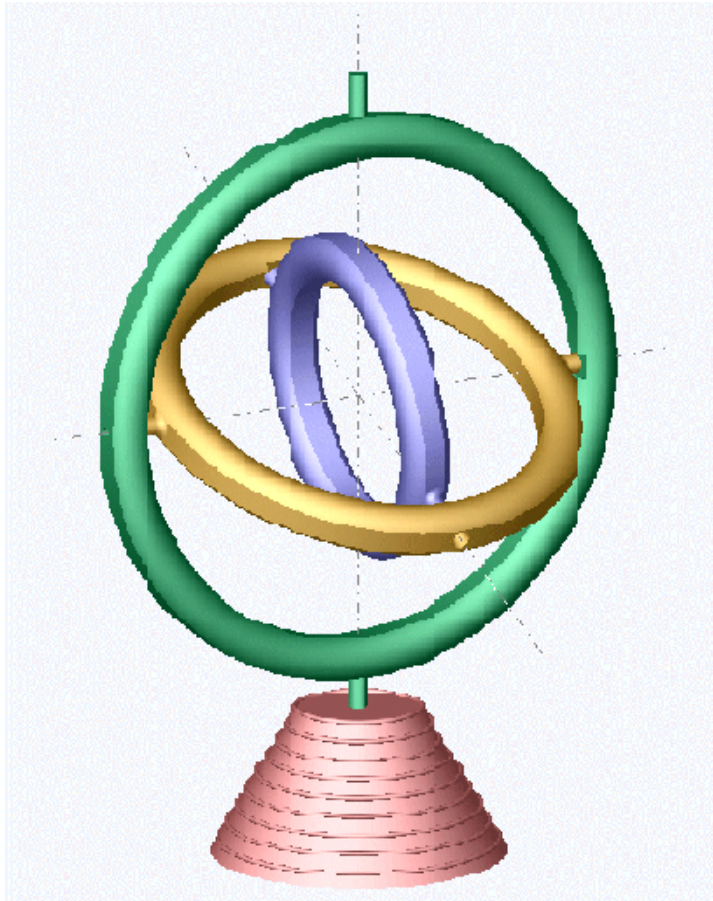
**Caution** A joint with three revolute primitives becomes singular if two or three of the rotation axes become parallel (“gimbal lock”). The simulation stops with an error in this case.

---

You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Gimbal block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

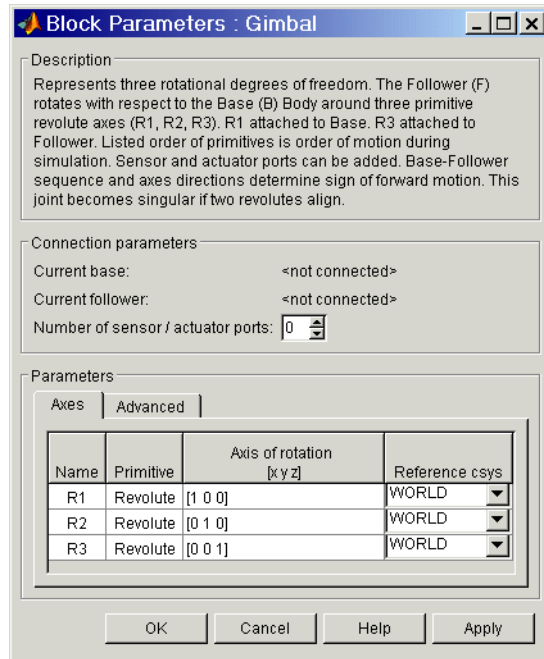
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



# Gimbal

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Gimbal block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Gimbal base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive R1 in the primitive list in **Parameters**.

### Current follower

When you connect the follower (F) connector port on the Gimbal block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Gimbal base and follower Body connector ports” figure.



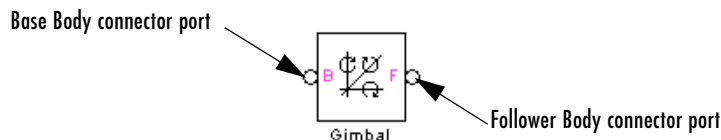
The follower Body is automatically connected to the last joint primitive R3 in the primitive list in **Parameters**.

### Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motion of revolute primitives is specified in angular units.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower moving around the rotational axis following the right-hand rule.



### Gimbal base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Gimbal has an entry line. These lines specify the direction of the axes of action of the DoFs that the Gimbal represents.

### Name - Primitive

The primitive list states the names and types of joint primitives that make up the Gimbal block: revolute primitives R1, R2, R3.

### Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Revolute: axis of rotation

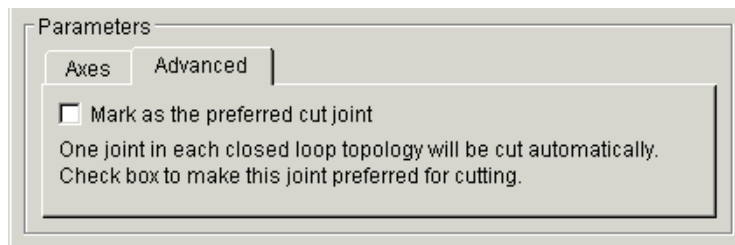
The default vectors are shown in the dialog box above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the revolute axes can be parallel.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Revolute, Spherical

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

**Purpose** Represent an immobile point at rest in World

**Library** Bodies

## Description



A Ground block represents an immobile ground point at rest in the absolute inertial World reference frame (RF). Connecting it to a Joint prevents one side of that Joint from moving.

Ground is a type of Body, but you can connect only one side of a Ground to a Joint block. A Ground block automatically carries a Grounded coordinate system (CS). This Grounded CS is inertial, at rest in the World reference frame, with coordinate axes parallel to the World axes:

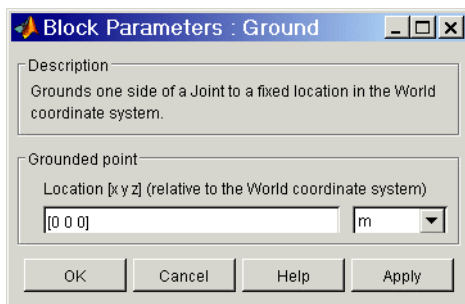
- +x points right
- +y points up (gravity in  $-y$  direction)
- +z points out of the screen, in three dimensions

But a Ground's origin is the ground point, which in general is shifted with respect to the World origin.

Every valid SimMechanics model *must* have at least one Ground block. Multiple Ground blocks represent different fixed points in the global inertial World. In the topology of a machine model, multiple Ground blocks function as a single body.

You cannot connect a Sensor or Actuator to a Ground block, because the ground point cannot be moved.

## Dialog Box and Parameters



# Ground

---

## **Grounded point**

Enter in this field the position of the ground point translated from the origin of the World CS. The position is specified as a translation vector  $(x,y,z)$ , with components projected on to the fixed World CS axes. Set the Ground position units using the pull-down menu to the right. The defaults are [0 0 0] and m (meters).

## **See Also**

Body

See “Creating Body CS Ports” on page 4-16 and “Modeling Joints” on page 4-17 for more on setting up Grounds.

**Purpose** Represent a composite joint with two translational DoFs

**Library** Joints

**Description** The In-Plane block represents a composite joint with two translational degrees of freedom (DoFs) as two prismatic primitives. There are no constraints among the primitives.



---

**Caution** A joint with two prismatic primitives becomes singular if the two translation axes become parallel. The simulation stops with an error in this case.

---

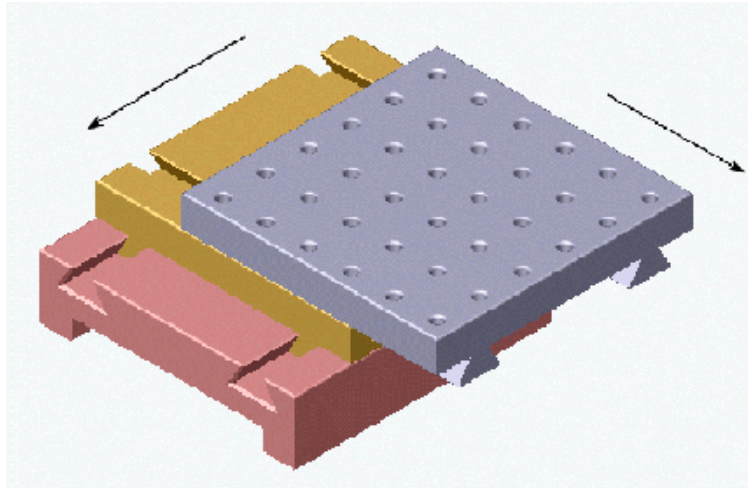
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The In-Plane block is assembled: the origins of these Body CSs must lie along the primitive axes, within assembly tolerances. But the Body CS origins on either side of the Joint do not have to be spatially collocated points.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

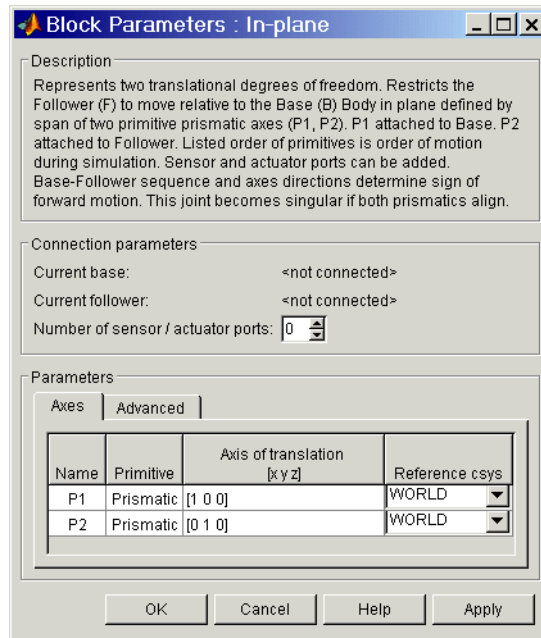
# In-Plane

---

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the In-Plane block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “In-Plane base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive P1 in the primitive list in **Parameters**.

### Current follower

When you connect the follower (F) connector port on the In-Plane block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “In-Plane base and follower Body connector ports” figure.

# In-Plane

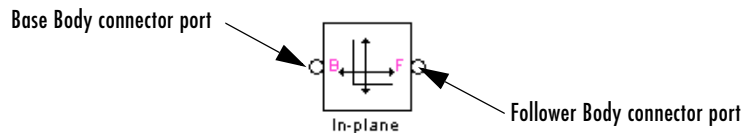
The follower Body is automatically connected to the last joint primitive P2 in the primitive list in **Parameters**.

## Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motion of prismatic primitives is specified in linear units.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.



## In-Plane base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in In-Plane has an entry line. These lines specify the direction of the axes of action of the DoFs that the In-Plane represents.

### Name - Primitive

The primitive list states the names and types of joint primitives that make up the In-Plane block: prismatic primitives P1, P2.

### Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Prismatic: axis of translation

The default vectors are shown in the dialog box above. The axis is a directed vector whose overall sign matters.

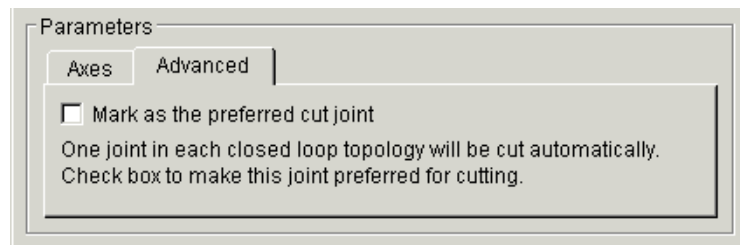
To prevent singularities and simulation errors, the two prismatic axes cannot be parallel.



## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Planar, Prismatic

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

# Joint Actuator

---

**Purpose** Apply force/torque or motion to a joint primitive

**Library** Sensors & Actuators

## Description



A joint between two bodies represents relative degrees of freedom (DoFs) between the bodies. The Joint Actuator block actuates a Joint block connected between two Bodies with one of these signals:

- A generalized force:
  - Force for translational motion along a prismatic joint primitive
  - Torque for rotational motion about a revolute joint primitive
- A motion:
  - Translational motion for a prismatic joint primitive, in terms of linear position, velocity, and acceleration. The velocity signal must be the derivative of the position signal, and the acceleration the derivative of the velocity.
  - Rotational motion for a revolute joint primitive, in terms of angular position, velocity, and acceleration. The angular velocity signal must be the derivative of the angle signal, and the angular acceleration the derivative of the angular velocity.

The generalized force or the motion is a function of time specified by a Simulink input signal. This signal can be any Simulink signal, including a signal feedback from a Sensor block.

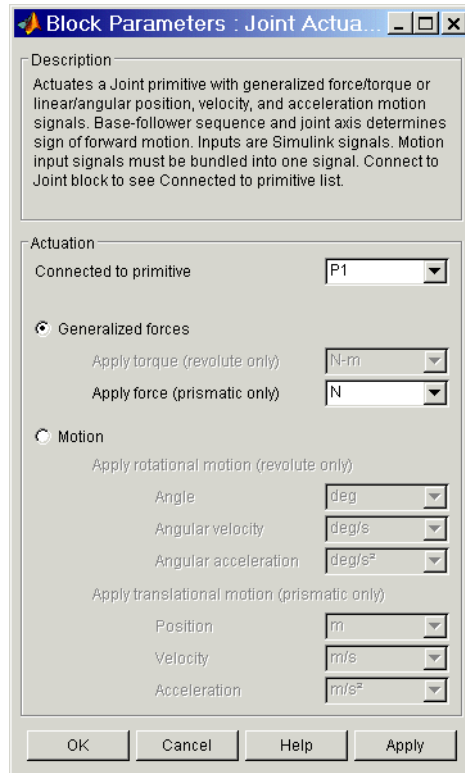
The Joint Actuator applies the actuation signal along/about the joint axis in the reference coordinate system (CS) specified for that joint primitive in the Joint's dialog box. The Joint connects a base and a follower Body. The base–follower sequence determines the sense of the actuation signal.

The inport is the Simulink input signal. The output is the connector port you connect to the Joint block you want to actuate. A Joint Actuator block actuates one joint primitive at a time:

- A primitive Joint (Prismatic or Revolute) has only one primitive within the Joint to actuate.
- A composite Joint has multiple joint primitives within, and you must choose which of those primitives to actuate with the Joint Actuator.

You cannot connect a Joint Actuator to a Spherical or spherical primitive.

## Dialog Box and Parameters



The dialog box has one active area, **Actuation**.

## Actuation

### Connected to primitive

In the pull-down menu, choose the joint primitive within the Joint that you want to actuate with the Joint Actuator. A primitive Joint block has only one joint primitive.

You cannot connect a Joint Actuator to a spherical primitive.

If the Joint Actuator is not connected to a Joint block, this menu shows only Unknown.

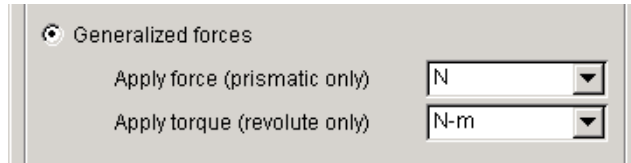
# Joint Actuator

---

Using the radio buttons, you can toggle between the two types of actuation, **Generalized forces** and **Motion**.

## Generalized Forces

This option interprets the actuation signal as a force or a torque between the Bodies connected by the Joint block you are actuating.



### Apply torque (revolute only)

In the pull-down menu, choose units for the actuation torque. The default is N-m (Newton-meters). This choice is valid only if you connect this Actuator block to a revolute primitive.

The Simulink input is a 1-component signal.

### Apply force (prismatic only)

In the pull-down menu, choose units for the actuation force. The default is N (Newtons). This choice is valid only if you connect this Actuator block to a prismatic primitive.

The Simulink input is a 1-component signal.

## Motion

This option interprets the actuation signal as a motion of the joint primitive to which you connect the Joint Actuator.

The screenshot shows a dialog box titled "Motion" with a radio button selected. It is divided into two sections: "Apply translational motion (prismatic only)" and "Apply rotational motion (revolute only)". Each section contains three rows of labels and pull-down menus. The translational section has "Position" (m), "Velocity" (m/s), and "Acceleration" (m/s<sup>2</sup>). The rotational section has "Angle" (deg), "Angular velocity" (deg/s), and "Angular acceleration" (deg/s<sup>2</sup>).

Section	Parameter	Default Unit
Apply translational motion (prismatic only)	Position	m
	Velocity	m/s
	Acceleration	m/s <sup>2</sup>
Apply rotational motion (revolute only)	Angle	deg
	Angular velocity	deg/s
	Angular acceleration	deg/s <sup>2</sup>

### Apply translational motion (prismatic only)

In the pull-down menus, choose units for the linear actuation **Position**, **Velocity**, and **Acceleration**. The defaults are m (meters), m/s (meters/second), and m/s<sup>2</sup> (meters/second<sup>2</sup>), respectively.

The Simulink input is a bundled 3-component signal with components in the order shown in the dialog box.

### Apply rotational motion (revolute only)

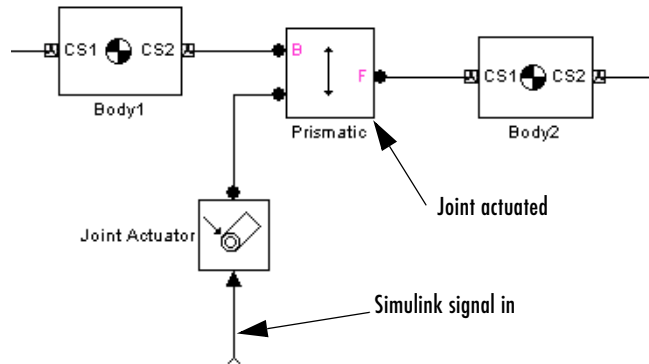
In the pull-down menus, choose units for the angular actuation **Angle**, **Angular Velocity**, and **Angular Acceleration**. The defaults are deg (degrees), deg/s (degrees/second), and deg/s<sup>2</sup> (degrees/second<sup>2</sup>), respectively.

The Simulink input is a bundled 3-component signal with components in the order shown in the dialog box.

# Joint Actuator

## Example

Here is a Joint Actuator connected to a Prismatic that connects two Bodies:



You must add an Actuator port (connector port) to the Joint block to connect the Joint Actuator to it. The base (B)-follower (F) Body sequence on the two sides of the Joint determines the sense of the Joint Actuator data.

## See Also

Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator, Mechanical Branching Bar, Prismatic, Revolute

See “Creating Joints” on page 8-4.

In Simulink, see the Signal Routing Library and the Sources Library.

# Joint Initial Condition Actuator

---

**Purpose** Apply initial positions and velocities to the primitives of a Joint before starting simulation

**Library** Sensors & Actuators

**Description** The Joint Initial Condition Actuator block supplies the prismatic and revolute joint primitives of a Joint block with initial value data. The initial values are the positions and velocities of the joint primitives and fully specify the initial state of motion (initial kinematic state) of those primitives.



When you build your machine, the geometric configuration implicitly specifies the initial positions/angles of bodies relative to one another and to World. The Ground, Body, and Joint layout only specifies initial coordinates (degrees of freedom), not their corresponding velocities, however. Starting a simulation in this state implicitly sets all initial velocities to zero. You can set the full initial kinematic state, both positions and velocities, of joint primitives by using Joint Initial Condition Actuator blocks.

You can set initial positions and velocities for two primitive types:

- Translational initial conditions for a prismatic primitive, in terms of linear position and velocity
- Rotational initial conditions for a revolute primitive, in terms of angular position and velocity

This block can actuate one, some, or all of the prismatic and revolute primitives of a Joint.

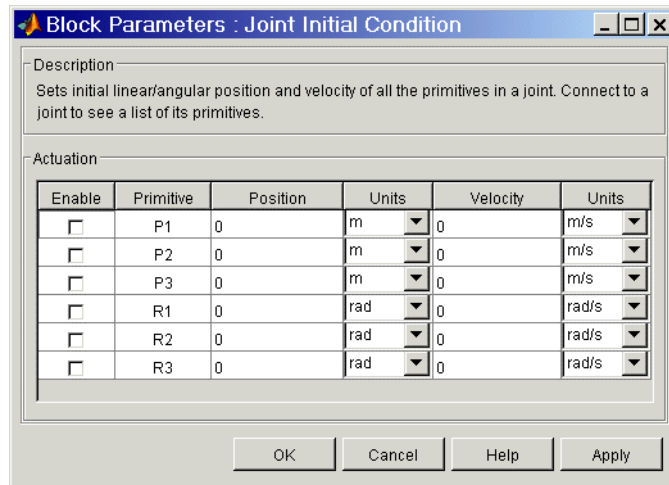
The Joint Initial Condition Actuator applies the initial state along/about the joint axis in the reference coordinate system (CS) specified for that joint primitive in the Joint's dialog box. The Joint connects a base and a follower Body. The base-follower sequence determines the sense of the actuation signal.

The output is the connector port you connect to the Joint block whose initial conditions you want to set. You set the initial linear and/or angular positions and velocities in the block's dialog box, so there is no input signal.

You cannot actuate a Spherical or spherical primitive with a Joint Initial Condition Actuator.

# Joint Initial Condition Actuator

## Dialog Box and Parameters



The dialog box has one active area, **Actuation**.

## Actuation

The menu choices are available for every primitive in the Joint to which the Joint Initial Condition Actuator is connected. If you connect the Actuator with its dialog open, the primitive list is automatically updated to reflect the connected Joint's primitives.

### Enable

Select this check box if you want to actuate the primitive with initial conditions. The default is unselected.

### Primitive

Displays the name of the primitive within the Joint. Not an active field.

### Position

Enter a value for the initial position of the primitive, either prismatic or revolute. The default is 0.

### Units

In the pull-down menu, select units for the initial position. The defaults are m (meters) for prismatic primitives and rad (radians) for revolute primitives.



# Joint Initial Condition Actuator

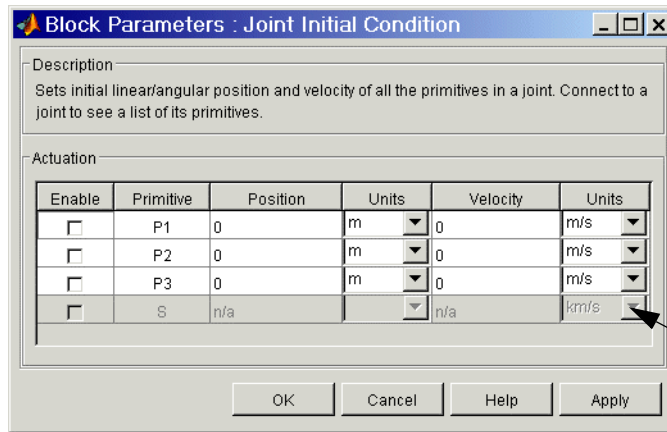
## Velocity

Enter a value for the initial velocity of the primitive, either prismatic or revolute. The default is 0.

## Units

In the pull-down menu, select units for the initial velocity. The defaults are m/s (meters/second) for prismatic primitives and rad/s (radians/second) for revolute primitives.

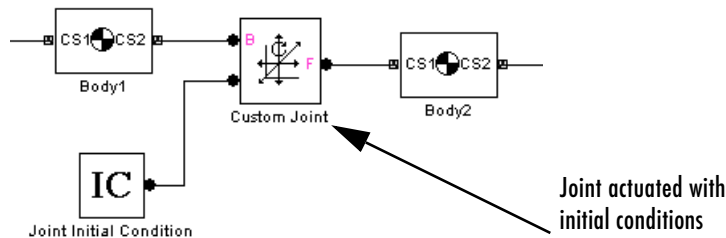
You cannot connect a Joint Initial Condition Actuator to a spherical primitive. If the connected Joint has a spherical primitive, the entry for that primitive is grayed out:



Grayed-out spherical primitive entry

## Example

Here is a Joint Initial Condition Actuator connected to a Custom Joint, which connects two Bodies:



# Joint Initial Condition Actuator

---

You must add an Actuator port (connector port) to the Joint block to connect the Joint Initial Condition Actuator to it. The base (B)-follower (F) Body sequence on the two sides of the Joint determines the sense of the Joint Initial Condition Actuator data.

## **See Also**

Joint Actuator, Joint Sensor, Joint Stiction Actuator, Mechanical Branching Bar, Prismatic, Revolute

See “Using JICA Blocks” on page 4-50 for setting general initial conditions (positions and velocities) of DoFs in a machine.

See “Creating Joints” on page 8-4.

In Simulink, see the Signal Routing Library and the Sources Library.

**Purpose** Measure the motion and force/torque of a joint primitive

**Library** Sensors & Actuators

**Description** The Joint Sensor block measures the position, velocity, and/or acceleration of a joint primitive in a Joint block.



The Joint Sensor measures the motion along/about the joint axis (or about the pivot point for a spherical primitive) in the reference coordinate system (CS) specified for that joint primitive in the Joint's dialog box. The Joint connects a base and a follower Body. The base–follower sequence determines the sense of the motion.

Depending on the joint primitive being sensed, you measure one of these motion types:

- Translational motion for a prismatic joint primitive, in terms of linear position, velocity, and/or acceleration
- Rotational motion for a revolute joint primitive, in terms of angular position, velocity, and/or acceleration
- Spherical motion for a spherical joint primitive, in terms of a quaternion, quaternion derivative, and/or quaternion second derivative

The input is the connector port connected to the Joint being sensed. The output is a set of Simulink signals or one bundled Simulink signal of the position, velocity, and/or acceleration of the joint primitive.

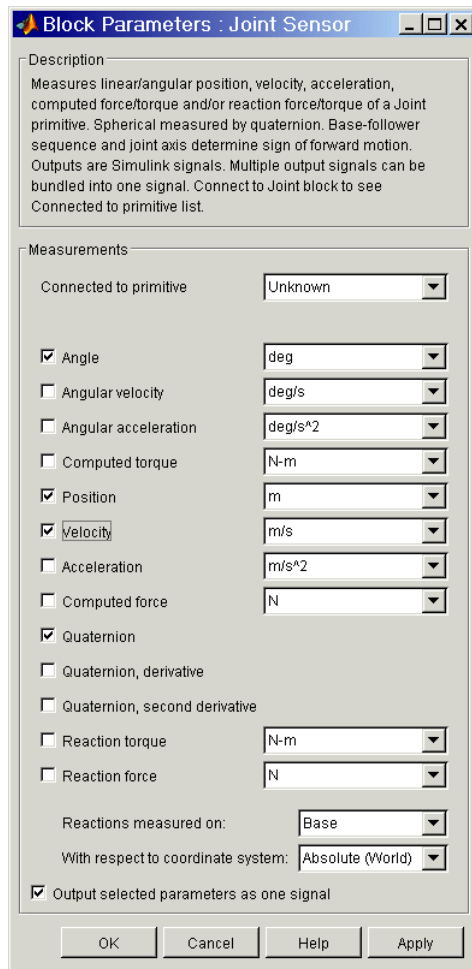
A Joint Sensor block measures one joint primitive at a time:

- A primitive Joint (Prismatic or Revolute) has only one primitive within the Joint to sense.
- A composite Joint has multiple joint primitives within, and you must choose which primitive to sense with the Joint Sensor.

A body's orientation rotation matrix  $R$  relates vector components measured in the body CS and in the inertial World CS by  $[R] \cdot \mathbf{v}_b = \mathbf{v}_g$ . The column vector  $\mathbf{v}_b$  lists the vector  $\mathbf{v}$ 's three components measured in the body CS. The column vector  $\mathbf{v}_g$  lists the vector  $\mathbf{v}$ 's three components measured in the World CS.

# Joint Sensor

## Dialog Box and Parameters



The dialog box has one active area, **Measurements**.

### Measurements Connected to primitive

In the pull-down menu, choose the joint primitive within the Joint that you want to measure with the Joint Sensor. A primitive Joint block has only one joint primitive.

If the Joint Sensor is not connected to a Joint block, this menu shows only Unknown.

The **Measurements** pane you see in the Joint Sensor dialog depends on the type of joint primitive to which you connect the Joint Sensor.

## Prismatic joint primitive (P)

Select the check box(es) for each of the possible measurements you want to make: **Position**, **Velocity**, **Acceleration**, **Computed Force**, **Reaction Torque**, and **Reaction Force**.

The *computed force* is that force along the prismatic axis which reproduces the follower motion with respect to base.

The *reaction force* and/or *torque* is (are) 3-component vector(s) of the force and/or torque that the joint primitive transfers to the base or follower Body.

In the pull-down menus, choose the units for each of the measurements you want. The defaults are m (meters), m/s (meters/second),  $m/s^2$  (meters/second<sup>2</sup>), N (Newtons), and N-m (Newton-meters), respectively, for **Position**, **Velocity**, **Acceleration**, **Force**, and **Torque**.

The screenshot shows a dialog box titled "Measurements" with the following settings:

- Connected to primitive: P1
- Position: m
- Velocity: m/s
- Acceleration: m/s<sup>2</sup>
- Computed Force: N
- Reaction Torque: N-m
- Reaction Force: N
- Reactions measured on: Base
- With respect to coordinate system: Absolute (World)
- Output selected parameters as one signal

Buttons at the bottom: OK, Cancel, Help, Apply.

# Joint Sensor

---

The bundled Simulink output signal for a prismatic primitive has these measurements ordered in a row vector. Unselected components are removed from the vector signal:

Position	Velocity	Acceleration	Computed Force	Reaction Torque (3-vector)	Reaction Force (3-vector)
----------	----------	--------------	----------------	----------------------------	---------------------------

## Revolute joint primitive (R)

Select the check box(es) for each of the possible measurements you want to make: **Angle**, **Angular velocity**, **Angular acceleration**, **Computed Torque**, **Reaction Torque**, and **Reaction Force**.

The *computed torque* is that torque about the revolute axis which reproduces the follower motion with respect to base.

The *reaction force* and/or *torque* is (are) 3-component vector(s) of the force and/or torque that the joint primitive transfers to the base or follower Body.

In the pull-down menus, choose the units for each of the measurements you want. The defaults are m (meters), m/s (meters/second),  $m/s^2$  (meters/second<sup>2</sup>), N (Newtons), and N-m (Newton-meters), respectively, for **Position**, **Velocity**, **Acceleration**, **Force**, and **Torque**.

Measurements

Connected to primitive: R1

Angle: deg

Angular velocity: deg/s

Angular acceleration: deg/s<sup>2</sup>

Computed Torque: N-m

Reaction Torque: N-m

Reaction Force: N

Reactions measured on: Base

With respect to coordinate system: Absolute (World)

Output selected parameters as one signal

OK Cancel Help Apply

**Note** The absolute angle of revolute motion is mapped on to the interval  $(-180^\circ, +180^\circ]$  degrees or  $(-\pi, +\pi]$  radians.

The bundled Simulink output signal for a revolute primitive has these measurements ordered in a row vector. Unselected components are removed from the vector signal:

Angle	Angular Velocity	Angular Acceleration	Computed Torque	Reaction Force (3-vector)	Reaction Torque (3-vector)
-------	------------------	----------------------	-----------------	---------------------------	----------------------------

## Spherical joint primitive (S)

Select the check box(es) for each of the possible measurements you want to make: **Quaternion**, **Quaternion, derivative**, and **Quaternion, second derivative**, **Reaction Torque**, and **Reaction Force**.

# Joint Sensor

The *reaction force* and/or *torque* is (are) 3-component vector(s) of the force and/or torque that the joint primitive transfers to the base or follower Body.

The quaternion is dimensionless. The time unit for the derivatives is seconds. Each quaternion measurement is a 4-component row vector.

Measurements

Connected to primitive: S

Quaternion  
 Quaternion, derivative  
 Quaternion, second derivative  
 Reaction Torque: N-m  
 Reaction Force: N  
Reactions measured on: Base  
With respect to coordinate system: Absolute (World)  
 Output selected parameters as one signal

OK Cancel Help Apply

The bundled Simulink output signal for a spherical primitive has these quaternion measurements ordered into a larger row vector. Unselected components are removed from the vector signal:

Quaternion (4-vector)	Quaternion, derivative (4-vector)	Quaternion, second derivative (4-vector)	Reaction Torque (3-vector)	Reaction Force (3-vector)
--------------------------	---	--	-------------------------------	------------------------------

## Reactions measured on

Choose the Body on which the reaction force and/or torque vector(s) is (are) measured, Base or Follower. The default is Base.

## With respect to coordinate system

In the pull-down menu, choose the coordinate system in which the reaction torque and/or force vector(s) is (are) measured: either the Local (Body CS) to which the Sensor is connected or the default Absolute (World).



In the Absolute case, the force and torque vectors have components measured relative to the inertial World CS axes. In the Local case, the same force and torque signals are premultiplied by the inverse orientation rotation matrix  $R^{-1} = R^T$  for the Body selected in **Reactions measured on**.

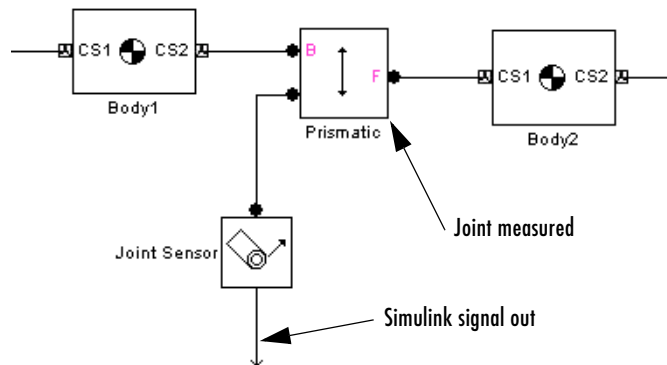
## Output selected parameters as one signal

Select this check box to convert all the output signals into a single bundled signal. The default is selected. If you unselect it, the Joint Sensor block will grow as many Simulink outports as there are active signals selected, in the same order top to bottom, in the dialog box.

If the check box is selected, the Simulink signal out has all the active signals ordered into a single row vector. The order and type of the signal components depend on the joint primitive, as listed in the Simulink signal tables above.

## Example

Here is a Joint Sensor connected to a Prismatic that connects two Bodies:



You must add an Sensor port (connector port) to the Joint block to connect the Joint Sensor to it. The base (B)-follower (F) Body sequence on the two sides of the Joint determines the sense of the Joint Sensor data.

## See Also

Body Sensor, Constraint & Driver Sensor, Joint Actuator, Joint Initial Condition Actuator, Joint Stiction Actuator, Mechanical Branching Bar, Prismatic, Revolute, Spherical

## Joint Sensor

---

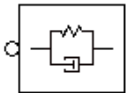
See “Representing Body Positions and Orientations” on page 3-2 and “Modeling Sensors” on page 4-60.

In Simulink, see the Signal Routing Library and the Sinks Library.

**Purpose** Models a damped linear oscillator force or torque on a prismatic or revolute joint between two bodies

**Library** Force Elements

## Description



The Joint Spring & Damper block models a damped linear oscillator force acting along a prismatic primitive or a damped linear oscillator torque acting about a revolute primitive. The joint primitives are connected between two bodies, and the force or torque acts between these bodies. The sign of the force or torque is set by the base (B)-to-follower (F) sequence of the bodies. These models represent a damped linear translational and/or torsional spring in the prismatic and revolute cases, respectively.

You connect this block to a Joint at one of the Joint's sensor/actuator ports. (If the Joint lacks a sensor/actuator port, open its dialog and create one.) The Joint represents any mixture of translational and rotational degrees of freedom (DoFs). With the Joint Spring & Damper block, you can then apply any combination of damped linear oscillator forces on any prismatic and damped linear torsion torques on any revolutes.

### Joint Spring and Damper Theory

Connect two Bodies with a Joint having some combination of prismatic and revolute primitives.

If  $x$  represents the displacement along a prismatic axis, and  $v = dx/dt$  is the prismatic DoF's linear speed, then the damped spring force acting along this prismatic and between the Bodies connected by this Joint is

$$F = -k(x - x_0) - bv$$

The model parameters are the spring constant  $k$ , the natural spring length (offset)  $x_0$ , and the damping constant  $b$ . The natural length is the spring's length with no forces acting on it and should be nonnegative:  $x_0 \geq 0$ . A stable spring requires  $k > 0$ . A damping representing dissipation and respecting the second law of thermodynamics requires  $b \geq 0$ . You can use a negative  $b$  to represent energy pumping.

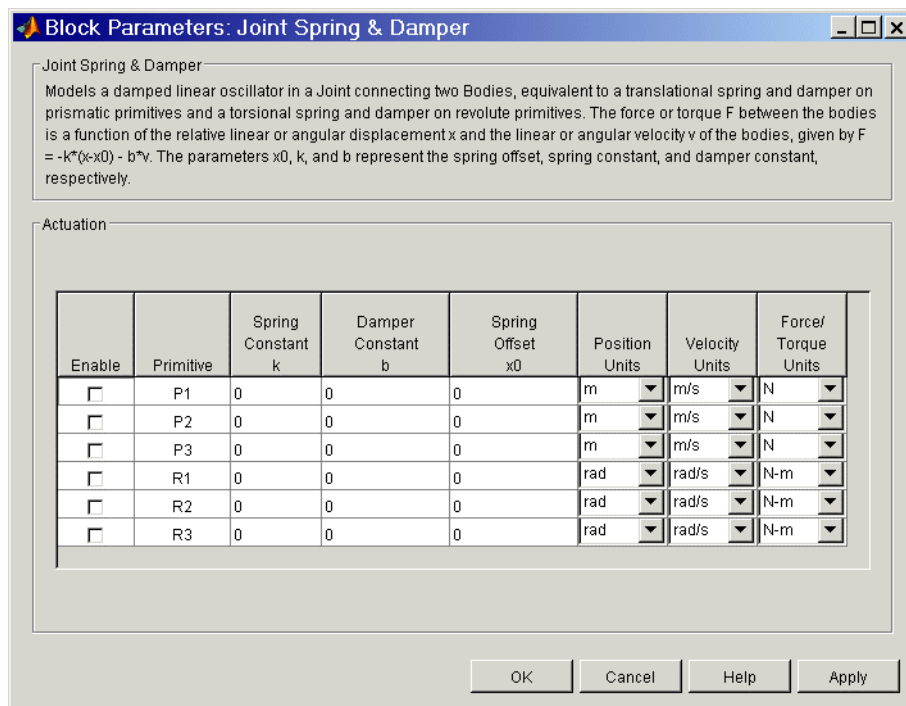
If  $\theta$  represents the displacement about a revolute axis, and  $\omega = d\theta/dt$  is the revolute DoF's angular speed, then the damped torsion torque acting about this revolute and between the Bodies connected by this Joint is

# Joint Spring & Damper

$$N = -k(\theta - \theta_0) - b\omega$$

The model parameters are the torsion constant  $k$ , the natural torsion angle (offset)  $\theta_0$ , and the damping constant  $b$ . The natural angle is the torsion balance's direction with no torques acting on it and can have any sign. A stable torsion requires  $k > 0$ . A damping representing dissipation and respecting the second law of thermodynamics requires  $b \geq 0$ . You can use a negative  $b$  to represent energy pumping.

## Dialog Box and Parameters



The dialog box has one active area, **Actuation**.

## Actuation

The menu lists all the active primitives in the Joint to which the Joint Spring & Damper block is connected. If you connect the Joint Spring & Damper with its dialog open, the primitive list is automatically updated to reflect the connected Joint's primitives.

## Enable

To enable force or torque actuation on any particular primitive in the Joint, select the **Enable** check box next to that primitive's name in the Primitive column. You cannot actuate spherical primitives.

## Primitive

Lists the active primitives in the Joint to which the block is connected. P represents a prismatic primitive, R a revolute primitive, and S a spherical primitive.

## Spring Constant $k$

Enter the spring or torsion constant  $k$ , for a prismatic or revolute primitive, respectively. The default is 0.

The units for  $k$  are derived implicitly from your choice of position and force/torque units.

## Damper Constant $b$

Enter the spring or torsion damping constant  $b$ , for a prismatic or revolute primitive, respectively. The default is 0.

The units for  $b$  are derived implicitly from your choice of velocity and force/torque units.

## Spring Offset $x_0$

Enter the natural spring length  $x_0$  or the natural torsion angle  $\theta_0$ , for a prismatic or revolute primitive, respectively. The default is 0.

## Position Units

In the pull-down menu, select linear or angular units for prismatic or revolute primitives, respectively. The default is m (meters) or rad (radians).

## Velocity Units

In the pull-down menu, select linear or angular velocity units for prismatic or revolute primitives, respectively. The default is m/s (meters/second) or rad/s (radians/second).

## Force/Torque Units

In the pull-down menu, select force or torque units for prismatic or revolute primitives, respectively. The default is N (Newtons) or N·m (Newton-meters).

# Joint Spring & Damper

---

## **See Also**

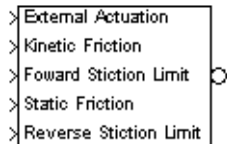
Body, Body Spring & Damper, Custom Joint, Joint Actuator, Joint Sensor, Prismatic, Revolute

See “Modeling Force Elements” on page 4-55.

**Purpose** Apply classical friction to a joint primitive

**Library** Sensors & Actuators

## Description



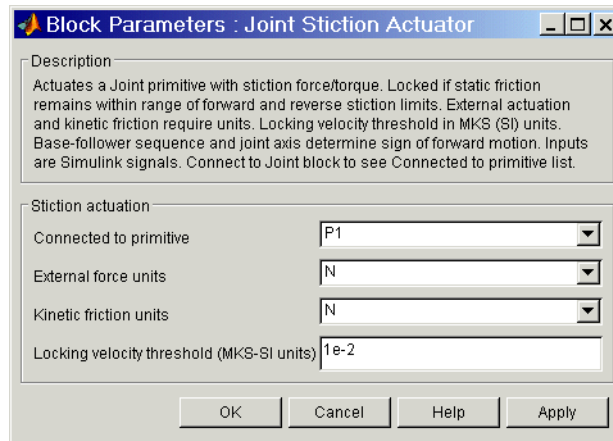
The Joint Stiction Actuator block applies stiction (classical friction) to a prismatic or revolute joint primitive. The stiction is regulated by an idealized model whose parameters you specify. (See “Stiction Theory” following.) The Joint Stiction Actuator applies stiction to the joint primitive as a relative force/torque between the joint’s connected Bodies. The bodies can experience additional forces independent of the joint.

The inports are Simulink signals. The output is a connector port. You cannot connect a Joint Stiction Actuator to a Spherical block or spherical primitive. Restrictions on simultaneous actuators and sensors:

- You cannot actuate a joint primitive simultaneously with a Joint Stiction Actuator and a Joint Actuator. But with the Joint Stiction Actuator inport External Actuation, you can apply to the joint primitive an external (nonfrictional) force/torque actuation signal equivalent to applying a Joint Actuator.
- You can simultaneously actuate a joint primitive with a Joint Stiction Actuator and a Joint Initial Condition Actuator.
- You can also simultaneously actuate a joint primitive with a Joint Stiction Actuator and measure the force/torque along/around the joint primitive with a Joint Sensor.

# Joint Stiction Actuator

## Dialog Box and Parameters



The dialog box has one active area, **Stiction actuation**.

### Connected to primitive

In the pull-down menu, choose the joint primitive within the Joint which you want to actuate with the Joint Stiction Actuator. A primitive Joint block has only one joint primitive.

You cannot connect a Joint Stiction Actuator to a spherical primitive.

If the Joint Stiction Actuator is not connected to a Joint block, this menu shows only Unknown.

### External force units

In the pull-down menu, choose units for the external nonfrictional force/torque  $F_{\text{ext}}$ . The default is N (Newtons) if connected to a prismatic primitive and N-m (Newton-meters) if connected to a revolute primitive.

### Kinetic friction units

In the pull-down menu, choose units for the kinetic friction force/torque  $F_K$ . The default is N (Newtons) if connected to a prismatic primitive and N-m (Newton-meters) if connected to a revolute primitive.

### Locking velocity threshold (MKS-SI units)

Enter the positive relative velocity of the joint primitive below which the joint locks by static friction. Above that velocity, the joint is unlocked. The

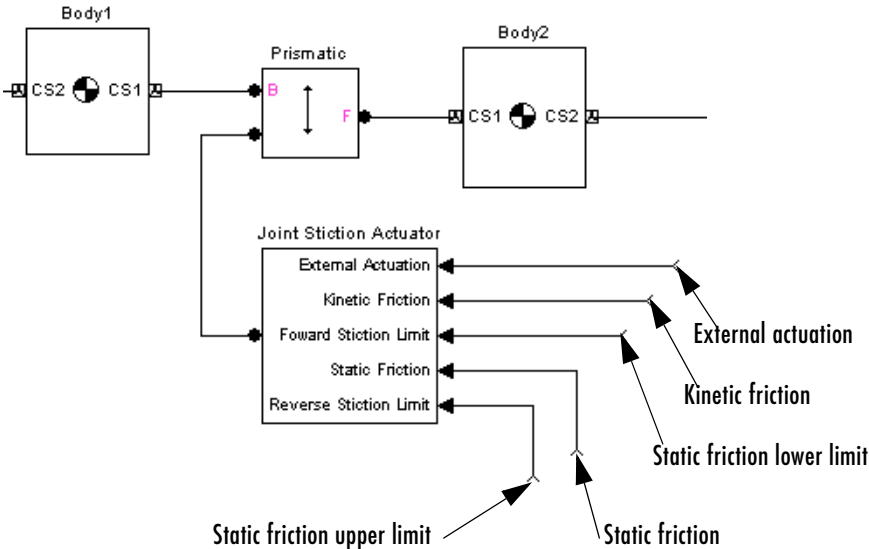


# Joint Stiction Actuator

units must be MKS: for a prismatic primitive, meters/second; for a revolute primitive, radians/second.

## Summary of Joint Stiction Actuator input signals

All the Simulink inports are 1-component signals. Here is an example of a prismatic joint connected between two bodies and actuated with stiction:



## Joint Stiction Actuator Simulink Inport Signals

### Summary of Joint Stiction Actuator Input Signals

Simulink inport	Friction Model Symbol	Description
External actuation	$F_{ext}$	External nonfrictional force/torque
Kinetic Friction	$F_K$	Kinetic friction
Forward Stiction Limit	$F_S^f < 0$	Static friction lower limit

# Joint Stiction Actuator

---

## Summary of Joint Stiction Actuator Input Signals (Continued)

Simulink inport	Friction Model Symbol	Description
Static Friction	$F_{\text{test}}$	Static test friction
Reverse Stiction Limit	$F_S^r > 0$	Static friction upper limit

### Units

You specify units in the dialog box only for the external nonfrictional and kinetic friction forces/torques,  $F_{\text{ext}}$  and  $F_K$ . These two friction signals are used to integrate the physical motion of the joint and have physical significance in the model context. So units are necessary for  $F_{\text{ext}}$  and  $F_K$ .

The other three signals are compared only to one another in the locking condition  $F_S^f < F_{\text{test}} < F_S^r$ . These friction signals are not used to integrate motion and thus do not have units set in the dialog box. But they must have the same implicit units for comparison.

---

**Caution** The threshold velocity  $v_{\text{th}}$  must be set greater than the **Absolute tolerance** in the **Simulation Parameters** dialog box to avoid a meaningless threshold value.

Never set **Absolute tolerance** to auto if stiction actuators are present in a model. A recommended setting is to make  $v_{\text{th}}$  at least 10 times the **Simulation Parameters** absolute tolerance.

See “Choosing Simulation Options” on page 5-2 for a discussion of setting simulation parameters.

---

### Example

The `mech_dpen_sticky` model in the Demos library has two revolute joints actuated with stiction. See “Joint Stiction Actuator Example: Mixed Static and Kinetic Friction” on page 4-48.

### Stiction Theory Kinematics

$v$  and  $a$  are the velocity and acceleration along/around a joint primitive axis. These quantities are relative between the two bodies at the joint ends and

signed  $\pm$  to indicate forward or reverse. The joint directionality is set by the base (B)-follower (F) Body sequence of Bodies attached to the joint primitive being actuated.

## Continuous Motion

A joint subject to stiction, if unlocked, moves in continuous motion. During this motion, you can apply two forces/torques at the joint primitive:

- a kinetic friction force/torque  $F_K$ :
  - $F_K < 0$  retards forward motion
  - $F_K > 0$  retards reverse motion
- an external, non-frictional force/torque  $F_{ext}$

## Discrete Modes: Locked, Wait, Unlocked

Besides its continuous motion state, a joint actuated by stiction has multiple discrete modes. The Joint Stiction Actuator switches a joint primitive between locked and unlocked modes via event detection. In one mode, the joint locks rigidly; in the other, it moves with the kinetic friction and external, nonfrictional forces/torques  $F_K$  and  $F_{ext}$  applied.

You specify the switching event by a two-condition threshold, constructed from four user-specified inputs. (See the following “Discrete Joint Stiction Modes and Transition Thresholds” figure).

- Joint unlocking threshold velocity  $v_{th} > 0$  via the block dialog box.
- *Static* friction limits  $F_S^f < 0$  and  $F_S^r > 0$  for forward and reverse motion, and a *static test friction*  $F_{test}$ ; all three specified via Simulink signals. The static test friction  $F_{test}$  and forward/reverse limits  $F_S^f$  and  $F_S^r$  can be functions of machine state and/or time.

The static test and kinetic frictions  $F_{test}$  and  $F_K$  and the threshold velocity  $v_{th}$  can be discontinuous, but should be physically sensible.

## Locked Mode

In this mode,  $v$  and  $a$  of the joint are zero. The static computed force/torque  $F_S$  at the joint is internally computed to maintain this mode:  $F_{ext} + F_S + F_F - F_B = 0$ . The forces/torques  $F_B$ ,  $F_F$  are the forces/torques on the base and follower Bodies apart from those forces/torques acting at the joint.

# Joint Stiction Actuator

---

The mode remains locked as long as  $F_S^f < F_{\text{test}} < F_S^r$ .

In most realistic friction models, you would set  $F_{\text{test}}$  equal to the computed  $F_S$ .

## Wait Mode

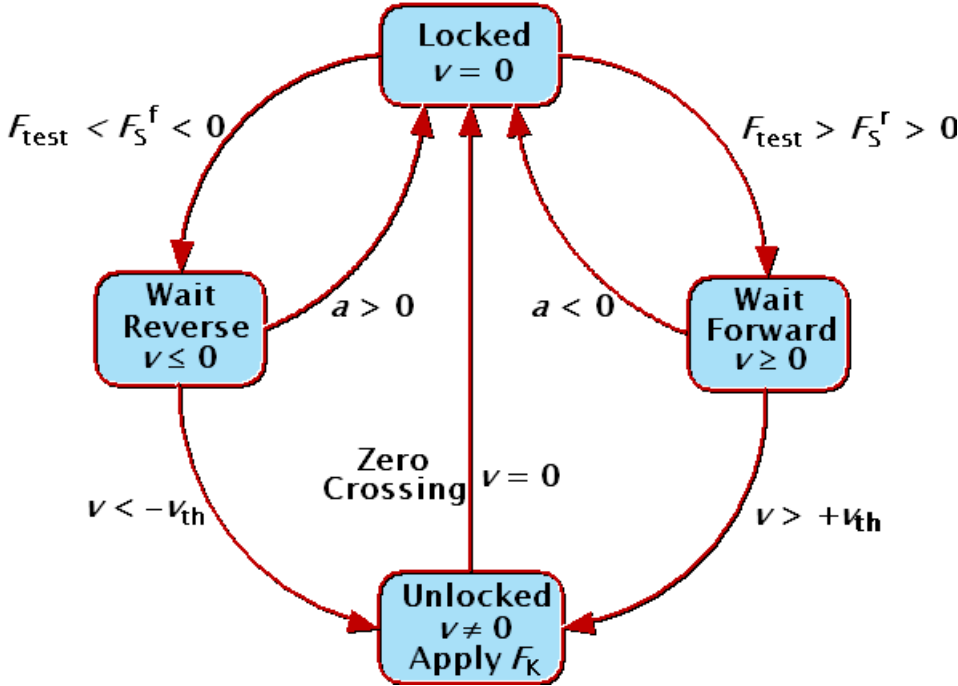
If the static test friction  $F_{\text{test}}$  leaves the static friction range  $[F_S^f, F_S^r]$ , the joint has passed the first threshold condition for unlocking, and the simulation enters Wait Mode.

A search begins for a mode of this joint consistent with the modes of all other joints. During the search, the net force/torque at the joint primitive  $F = F_{\text{ext}} + F_K$  is computed, where  $F_K$  is the kinetic friction. In forward (reverse) motion, if  $a < 0$  ( $a > 0$ ), the search returns to the Locked Mode. Once consistent modes for all joints are found, the simulation restarts and integrates  $a$  to obtain  $v$ . When  $|v|$  exceeds  $v_{\text{th}}$ , the second threshold condition, the joint unlocks.

This mode prevents infinite cycling between Locked and Unlocked Modes, although it can noticeably slow down the simulation. The mode search uses an algebraic loop, which displays warnings at the MATLAB command line.

## Unlocked Mode

In the Unlocked Mode, the joint primitive moves, actuated by the sum of the external, non-frictional force/torque  $F_{\text{ext}}$  and the kinetic friction  $F_K$ . The joint returns to the Locked Mode if Simulink detects  $v$  crossing zero.



### Discrete Joint Stiction Modes and Transition Thresholds

**See Also**

Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Mechanical Branching Bar, Prismatic, Revolute

See “Creating Joints” on page 8-4.

In Simulink, see the Signal Routing Library and the Sources Library, and Zero Crossing Detection.

# Linear Driver

---

**Purpose** Specify a component of the vector difference of two Body CS origins as a function of time

**Library** Constraints & Drivers

**Description** The Linear Driver block specifies a component of the vector difference of Body coordinate system (CS) origins as a function of time.



Let  $\mathbf{r}_1$ ,  $\mathbf{r}_2$  be the vector positions of the origins of CS1 on one Body, CS2 on the other Body, and  $\mathbf{R} = \mathbf{r}_1 - \mathbf{r}_2$ . The Linear Driver block specifies one of the vector components of  $\mathbf{R} = (X, Y, Z)$ , projected on to the World CS axes, as a function of time:

$$X, Y, \text{ or } Z = f(t).$$

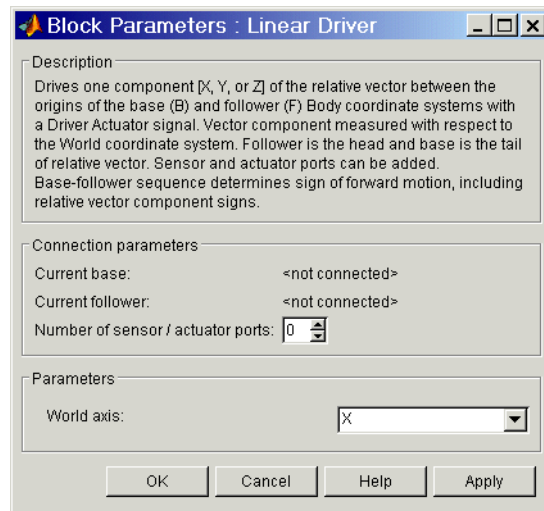
You connect a Driver Actuator block to the Linear Driver.

The Simulink input signal into the Driver Actuator specifies the time-dependent driving function  $f(t)$  and its first two derivatives, as well as their units. If you do not actuate Linear Driver, this block acts as a time-independent constraint that freezes the vector component between the two Body CS origins at its initial value during the simulation.

Drivers restrict relative degrees of freedom (DoFs) between a pair of bodies as specified functions of time. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Driver blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Drivers are ordered as base and follower, fixing the direction of relative motion.

You can also connect a Constraint & Driver Sensor to any Driver and measure the reaction forces/torques between the driven bodies.

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Linear Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Linear Driver base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Linear Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Linear Driver base and follower Body connector ports” figure.

### Number of sensor/actuator ports

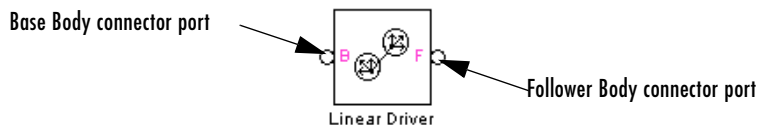
Using this spinner menu, you can set the number of extra connector ports needed for connecting Driver Actuator and Constraint & Driver Sensor blocks to this Driver. The default is 0.

To activate the Driver, connect a Driver Actuator.

# Linear Driver

---

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.



## Linear Driver base and follower Body connector ports

### Parameters

#### World axis

In the pull-down menu, choose the component of the vector difference  $\mathbf{R}$  between the Body CS origins that you want to drive as a function of time. The components are measured with respect to the World CS axes. The choices are X, Y, or Z. The default is X.

### See Also

Constraint & Driver Sensor, Distance Driver, Driver Actuator

See “Modeling Constraints and Drivers” on page 4-34 for more on restricting DoFs with Drivers.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on using drivers in closed loops.

See “Constraining and Driving Motion” on page 8-6.



**Purpose** Map multiple sensor/actuator lines to one sensor/actuator port on a Joint, Constraint, or Driver, or to one Body coordinate system port on a Body

**Library** Utilities

## Description



The Mechanical Branching Bar bundles multiple actuator and sensor connection lines into one line, allowing you to connect multiple actuators and/or sensors to a single connector port on a Joint, Constraint, or Driver, or to a single Body coordinate system (CS) port on a Body. You can choose any number of sensor/actuator ports on the Mechanical Branching Bar.

- In the case of a Body, a single Body CS port represents a single Body CS. If the needed Body CS port does not exist, open the Body dialog and create one. You can connect the selected Body CS to multiple Body Actuators and Sensors through the Mechanical Branching Bar.
- In the case of a Joint, you need a single sensor/actuator port on the Joint. If the needed port does not exist, open the Joint's dialog and create one. You can connect this sensor/actuator port to multiple Actuators and Sensors through the Mechanical Branching Bar.

Using the Mechanical Branching Bar, you can connect a Joint block to any combination of Joint Sensors, Joint Actuators, Joint Initial Condition Actuators, and Joint Stiction Actuators. The Actuator and Sensor dialogs display the Joint's primitives as if they were directly connected to the Joint.

- The procedure for Constraints and Drivers is the same as it is for Joints, except that you need to choose to measure reaction forces/torques or to actuate motions.

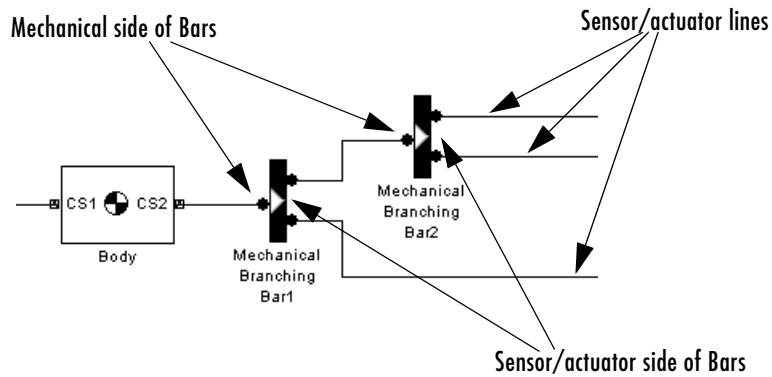
### Cascading Mechanical Branching Bars and Avoiding Closed Loops

You can connect multiple Mechanical Branching Bar blocks in series, creating a cascade. Connect the mechanical side of the first Branching Bar to a Joint, Constraint, Driver, or Body. Then connect its sensor/actuator side to the mechanical side of the second Branching Bar, and so on.

The only restriction on cascading Mechanical Branching Bars is that you must avoid connecting them into closed loops.

The following diagram shows a cascade, starting at a Body.

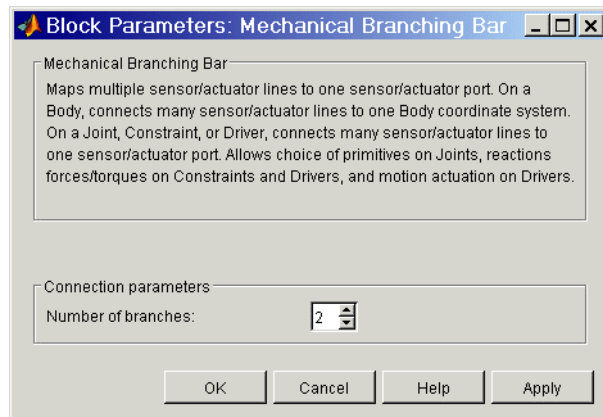
# Mechanical Branching Bar



**Caution** To avoid simulation errors, you should not create a cascade of Mechanical Branching Bars that closes on itself in a loop.

You should not connect the mechanical side of one Mechanical Branching Bar to the mechanical side of another Mechanical Branching Bar. You should also not connect the sensor/actuator side of one Mechanical Branching Bar to the sensor/actuator side of another Mechanical Branching Bar.

## Dialog Box and Parameters



The dialog box has one active area, **Connection parameters**.

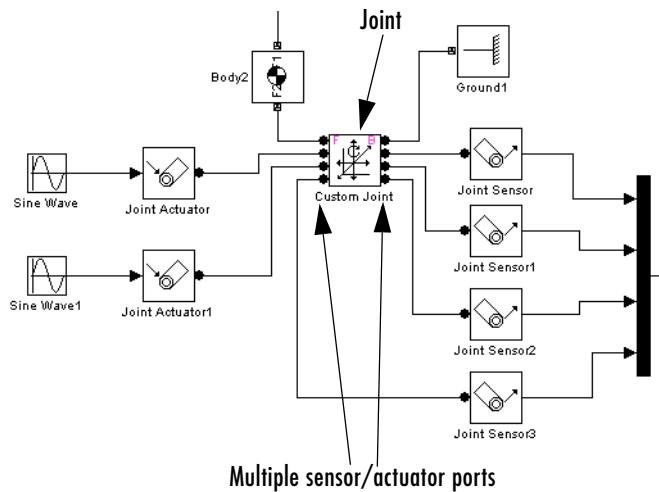
## Connection Parameters

### Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Actuator and Sensor blocks to the Mechanical Branching Bar. The default is 2.

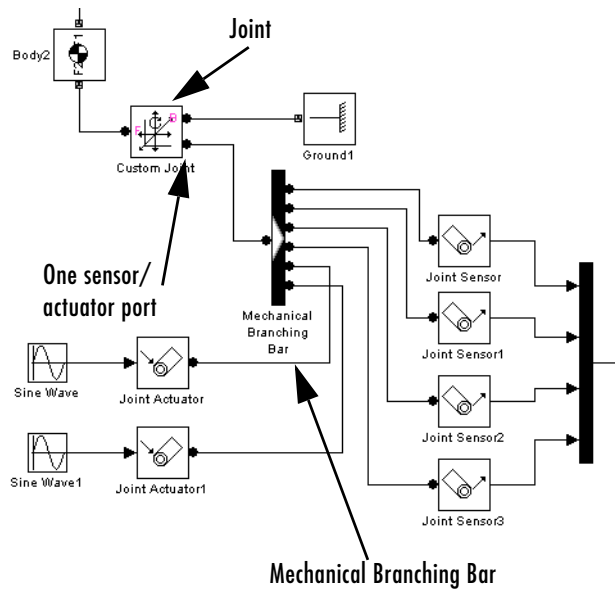
## Example

Without the Mechanical Branching Bar, you must connect multiple Sensors and Actuators to a Joint by creating a sensor/actuator port on the Joint for each Sensor and each Actuator:



With the Mechanical Branching Bar block, you can combine all the sensor and actuator ports for a single Joint into one sensor/actuator port:

# Mechanical Branching Bar



## See Also

Body, Body Actuator, Body Sensor, Constraint & Driver Sensor, Driver Actuator, Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator

**Purpose** Constrain the body axis vectors of two bodies to be parallel

**Library** Constraints & Drivers

## Description



The two Bodies connected by a Parallel Constraint are restricted in their relative rotational motion. The Parallel Constraint is connected on either side to a Body CS, one on each Body. A vector  $\mathbf{a}_B$  defined in one Body CS on the base body remains parallel to a second vector  $\mathbf{a}_F$  defined in another Body CS on the follower body.

The Parallel Constraint block requires that:

$$|\mathbf{a}_B \cdot \mathbf{a}_F| / (|\mathbf{a}_B| |\mathbf{a}_F|) = 1.$$

You specify the initial direction to which both vectors must remain parallel.

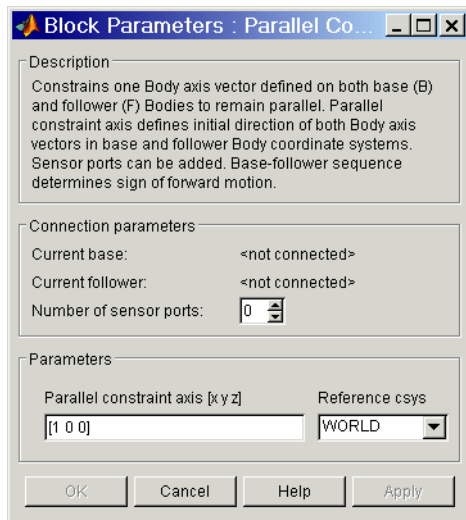
Constraints restrict relative degrees of freedom (DoFs) between a pair of bodies. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Constraint blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Constraint are ordered as base and follower, fixing the direction of relative motion.

Parallel Constraint is assembled: the Body CS origin on the base body must be initially collocated with the Body CS origin on the follower body, to within assembly tolerance.

You can connect a Constraint & Driver Sensor to any Constraint block, but not a Driver Actuator. The Constraint & Driver Sensor measures the reaction forces/torques between the constrained bodies.

# Parallel Constraint

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Parallel Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Parallel Constraint base and follower Body connector ports” figure.

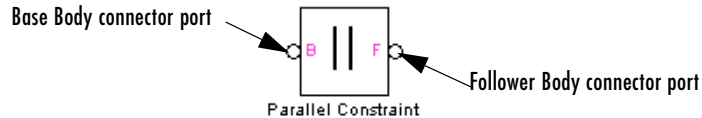
### Current follower

When you connect the follower (F) connector port on the Parallel Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Parallel Constraint base and follower Body connector ports” figure.

### Number of sensor ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Constraint & Driver Sensor blocks to this Constraint. The default is 0.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-handed sense about the rotation axis.



## Parallel Constraint base and follower Body connector ports

### Parameters

#### Parallel constraint axis [x y z]

Enter the axis vector defining the initial direction of the two body axis vectors  $\mathbf{a}_b$ ,  $\mathbf{a}_f$ . These body axis vectors are restricted to always remain parallel to this initial axis. The default is [1 0 0].

#### Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the initial **Parallel constraint axis** is oriented with respect to. This CS also determines the absolute meaning of reaction forces/torques at this Constraint. The default is WORLD.

### See Also

Angle Driver, Constraint & Driver Sensor, Velocity Driver

See “Modeling Constraints and Drivers” on page 4-34 for more on restricting DoFs with Constraints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on using constraints in closed loops.

See “Constraining and Driving Motion” on page 8-6.

# Planar

---

## Purpose

Represent a composite joint with two translational DoFs and one rotational DoF, with the rotational axis orthogonal to the plane of the translational axes

## Library

Joints

## Description



The Planar block represents a composite joint with two translational degrees of freedom (DoFs) as two prismatic primitives and one rotational DoFs as one revolute primitives. The rotation axis must be orthogonal to the plane defined by the two translation axes.

---

**Caution** A joint with two prismatic primitives becomes singular if the two translation axes become parallel. The simulation stops with an error in this case.

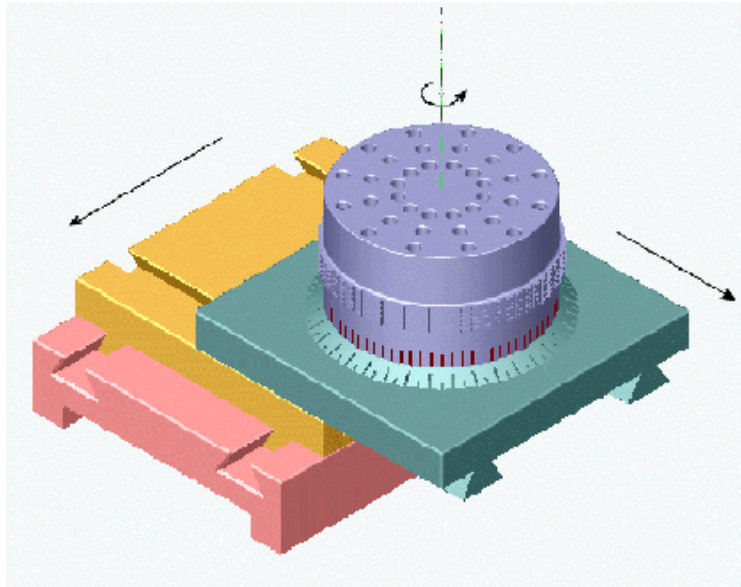
---

You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Planar block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

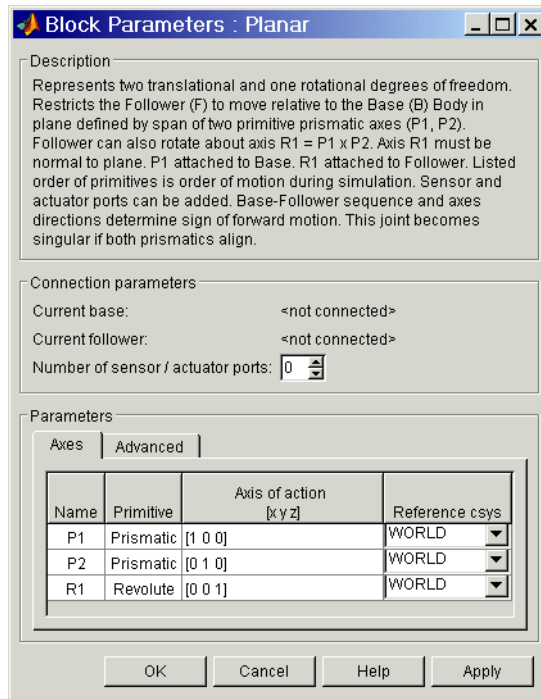


A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



# Planar

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Planar block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Planar base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive P1 in the primitive list in **Parameters**.

### Current follower

When you connect the follower (F) connector port on the Planar block to a Body CS Port on a Body, this parameter is automatically reset to the name

of this Body CS. See the following “Planar base and follower Body connector ports” figure.

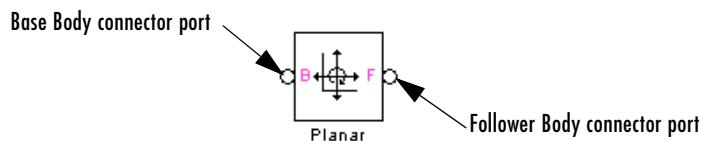
The follower Body is automatically connected to the last joint primitive R1 in the primitive list in **Parameters**.

### Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motions of prismatic and revolute primitives are specified in linear and angular units, respectively.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower moving around the rotational axis following the right-hand rule.



### Planar base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Planar has an entry line. These lines specify the direction of the axes of action of the DoFs that the Planar represents.

### Name - Primitive

The primitive list states the names and types of joint primitives that make up the Planar block: prismatic primitives P1, P2 and revolute primitives R1.

### Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Prismatic: axis of translation

- Revolute: axis of rotation

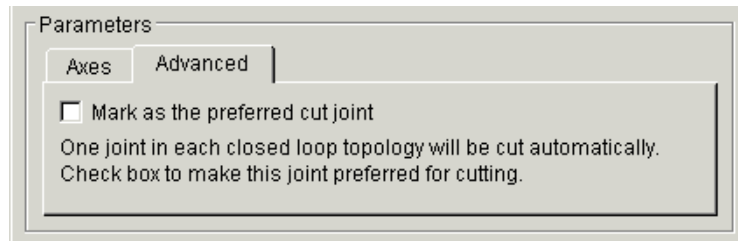
The default vectors are shown in the dialog box above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, the two prismatic axes cannot be parallel.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

In-Plane, Prismatic, Revolute

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

**Purpose** Constrain the motion of a point on one body to be along a curve on another body

**Library** Constraints & Drivers

**Description**



The two Bodies connected by a Point-Curve Constraint can only move relative to one another if a point on one body moves along a curve on the other body. The point on one body is the origin of the Body coordinate systems (CS) to which one side of the Point-Curve Constraint is connected. The corresponding curve starting point on the other body is the origin of the Body CS to which the other side of the Point-Curve Constraint is connected.

---

**Specifying the Curve** You specify the curve function on the second body as a *spline* with break points and end conditions. The spline is a piecewise cubic polynomial, with the pieces joined at user-specified *breakpoints*:

$$(x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_N, y_N, z_N)$$

and boundary conditions applied at the spline's *endpoints*,  $(x_0, y_0, z_0)$  and  $(x_{N+1}, y_{N+1}, z_{N+1})$ . The spline curve and its first two derivatives are continuous at each breakpoint.

---

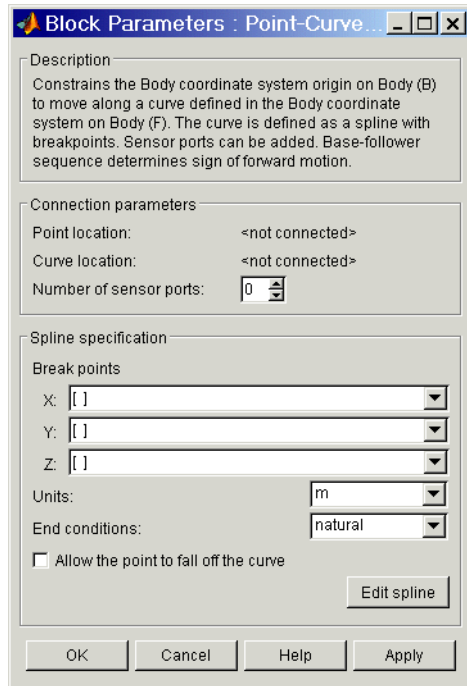
Constraints restrict relative degrees of freedom (DoFs) between a pair of bodies. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Constraint blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Constraint are ordered as base and follower, fixing the direction of relative motion.

For the Point-Curve Constraint, the base is the Body carrying the Point, and the follower is the Body carrying the Curve. The Point-Curve Constraint is assembled: the Body CS origin on the base (Point) body must initially be collocated with the Body CS origin on the follower (Curve) body, to within assembly tolerance.

You can connect a Constraint & Driver Sensor to any Constraint block, but not a Driver Actuator. The Constraint & Driver Sensor measures the reaction forces/torques between the constrained bodies.

# Point-Curve Constraint

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Spline specification**. It stores the defining information of a single spline for the constraint.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Point-Curve Constraint block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Point-Curve Constraint base and follower Body connector ports” figure.

This Body CS origin is the point of the Point-Curve Constraint.

### Current follower

When you connect the follower (F) connector port on the Point-Curve Constraint block to a Body CS Port on a Body, this parameter is

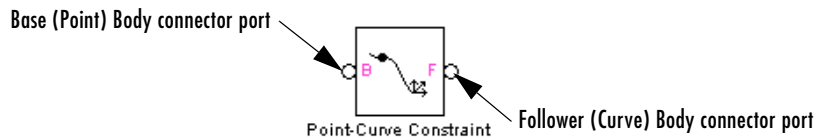
automatically reset to the name of this Body CS. See the following “Point-Curve Constraint base and follower Body connector ports” figure.

This Body CS origin is the starting point of the curve of the Point-Curve Constraint.

## Number of sensor ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Constraint & Driver Sensor blocks to this Constraint. The default is 0.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.



## Point-Curve Constraint base and follower Body connector ports

## Specifying the Spline

The Point-Curve Constraint dialog box gives you two ways to specify the spline curve. The first way is entering in this dialog box the coordinates of breakpoints and endpoints on the follower and is valid for defining curves in up to three dimensions.

The second way is graphically displaying and editing the spline in the **Edit spline** editor (see following), valid only for two-dimensional curves on the follower.

## Break points

List here the  $x$ -components, the  $y$ -components, and the  $z$ -components, respectively, of the breakpoints and endpoints that define the spline:

**X:** enter  $(x_0, x_1, \dots, x_{N+1})$  as a vector.

**Y:** enter  $(y_0, y_1, \dots, y_{N+1})$  as a vector.

**Z:** enter  $(z_0, z_1, \dots, z_{N+1})$  as a vector.

# Point-Curve Constraint

All three fields require nonnull entries. The number of components in each vector should be the same. *Exception and shortcut:* if all the **Z** components are the same, just enter one number in the **Z** vector. The **Break points** list replicates this number to expand out a full vector.

If there are no **X** and/or **Y** components, you must still enter [0 0] in that/those field(s). If there are no **Z** components, you must still enter at least [0] in the **Z** field (using the replication/expansion shortcut).

The pull-down menu for each spatial dimension lists the history of those previous breakpoints created by the graphical spline editor (see following) within a single dialog box session. Closing the dialog box destroys this history, and only the current breakpoint list is retained.

## Units

In the pull-down menu, choose the linear units for distances on the constrained bodies. The default is m (meters).

## End conditions

In the pull-down menu, choose the type of end (boundary) condition on the spline curve. The possible conditions are:

End Condition	Definition	Minimum Number of Points
natural	Match each endslope to the slope of the cubic that fits the first four points at that end	Two points
not-a-knot	Only the curve and its first derivative are continuous at first and last interior points	Four points
periodic	Match the first and second derivatives of the two endpoints	Two points, three recommended

The default is natural. The periodic end condition closes the spline.

## Allow the point to fall off the curve

If the check box is selected, the base point continues with unconstrained motion if it reaches an endpoint and leaves the spline on the follower. The



direction of motion at the instant the base point leaves the constraint is tangent to the spline.

If the check box is unselected, and the base point attempts to leave the spline on the follower, the simulation stops with an error. The default is unselected.

## **Edit spline**

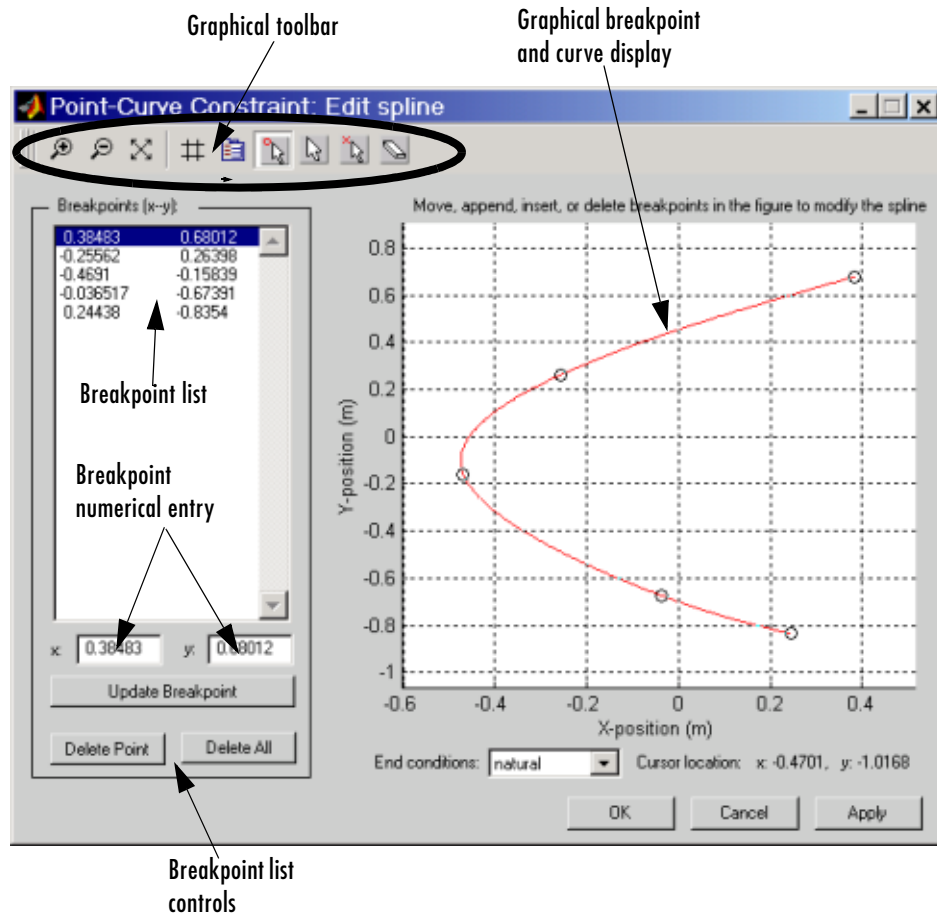
Click [here](#) to open the optional **Edit spline** dialog box.

The **Edit spline** dialog box provides alternative numerical entry and graphical editing methods for defining the constraint spline. But it can define only two-dimensional curves in the  $x$ - $y$  coordinate directions on the follower Body. The **Edit spline** editor ignores any  $z$ -components in existing breakpoints.

# Point-Curve Constraint

## Edit Spline

The numerical entry area lies on the left side of the **Edit spline** dialog box, the graphical editing area on the right side.

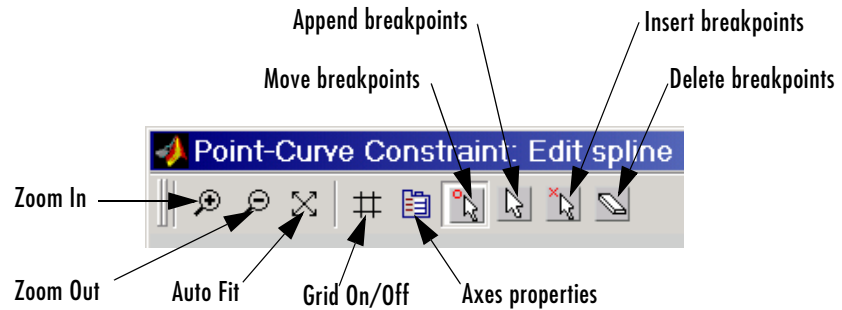


## Point-Curve Constraint Spline Editor

### Graphical Editing of Spline Points

- 1 To place a breakpoint in the graphical display, place your cursor at the position where you want the breakpoint. The **Cursor location** display in the lower right indicates your current cursor coordinates in the display.

- 2 Then click at the desired point. A circle appears where you clicked, and simultaneously, the breakpoint is listed in the **Breakpoints (x-y)** list. Continuing to add breakpoints generates the spline (red curve).
- 3 Use the Graphical toolbar controls to edit the spline graphically in the display:



- Remove points by clicking on the **Delete breakpoints** icon. Your cursor turns into an eraser symbol. With it, select and click the breakpoints you want to delete.
- Insert new (interior) breakpoints by clicking on the **Insert breakpoints** icon. Your cursor acquires a small circle. Click on the positions, near the existing curve, where you want the new breakpoints. The editor modifies the spline to fit the new breakpoints.
- Add new endpoints and extend the curve by clicking on the **Append breakpoints** icon. Your cursor acquires a small circle. Click on the positions, near the existing endpoints, to where you want to extend the curve. The editor modifies the spline to fit the new endpoints.
- Move existing endpoints by clicking on the **Move breakpoints** icon. Click and drag the breakpoints you want to move, then drop them where you want them.

The editor modifies both the spline red curve in the graphical display and the **Breakpoints (x-y)** list as you make these changes.

Additional graphical toolbar controls:

- **Zoom In/Zoom Out** and **Auto Fit**: Standard Handle Graphics zooming and auto resizing of graphics display.
- **Grid On/Off**: Turn the graphical display x-y grid on or off.

# Point-Curve Constraint

---

- **Axes properties:** Edit properties of graphical display.

## Numerical Editing of Spline Points

Use the numerical entry controls, instead of the graphical editing tools, to edit breakpoints by text entry.

### Breakpoints (x-y)

You can also add, delete, and edit the breakpoints via this Breakpoints list:

Select an existing breakpoint by highlighting it with your cursor.

Add a breakpoint by moving the highlighted selection to the empty line below the last breakpoint with your cursor control.

In **x:** and **y:** enter the  $x$ - and  $y$ -coordinates of the currently selected breakpoint.

### Update breakpoint

After editing an existing breakpoint or entering a new one, update the breakpoint list by clicking here. The new or changed breakpoint appears in the graphical display as a circle.

### Delete point

Click here to delete the currently selected breakpoint.

### Delete all

Click here to delete all the breakpoints in the Breakpoint list.

### End conditions

In the pull-down menu, choose the type of end (boundary) condition on the spline curve. The possible conditions are natural, not-a-knot, and periodic. The default is natural.

## Closing the Edit spline Dialog Box

Clicking on **Apply** or **OK** updates the breakpoints stored in the main Point-Curve Constraint dialog box.

Previous breakpoint lists are stored in the history pull-down menus of the main Point-Curve Constraint dialog box's **Break points** list. This history is

destroyed if you close the main dialog box, and only the current Breakpoint list is retained.

## **See Also**

Constraint & Driver Sensor

See “Modeling Constraints and Drivers” on page 4-34 for more on restricting DoFs with Constraints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on using constraints in closed loops.

See “Constraining and Driving Motion” on page 8-6.

For more about representing curves as splines, see the Spline Toolbox User’s Guide tutorial.

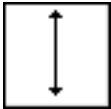
# Prismatic

---

**Purpose** Represent a prismatic joint with one translational degree of freedom

**Library** Joints

**Description**

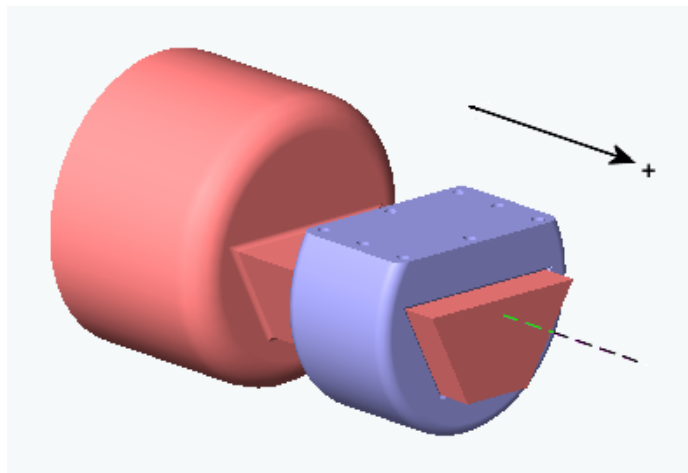


The Prismatic block represents a single translational degree of freedom (DoF) along a specified axis between two bodies. A prismatic joint is one of SimMechanics primitive joints, along with revolute and spherical.

The Prismatic block is assembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, and the origins of these Body CSs must lie along the prismatic axis, to within assembly tolerances. These Body CS origins do not need to be collocated in space.

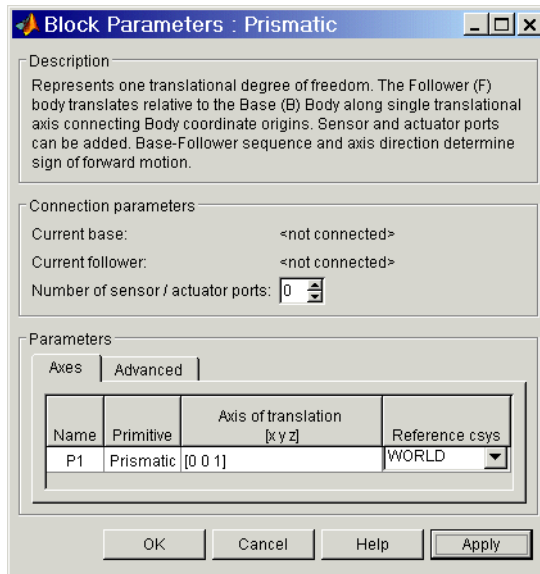
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.



**Prismatic motion of follower (blue) relative to base (red)**

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Prismatic block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Prismatic base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Prismatic block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Prismatic base and follower Body connector ports” figure.

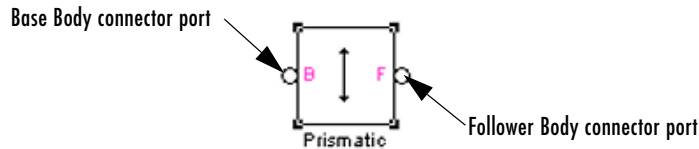
### Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

# Prismatic

The motion of a Prismatic is specified in linear units.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis.

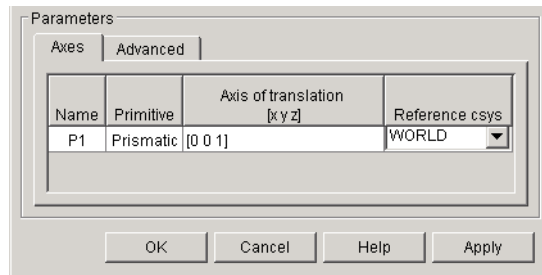


## Prismatic base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. They specify the direction of the translational DoF that the Prismatic represents.



## Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Prismatic, there is only one primitive joint, a prismatic, labeled P1.

## Primitive

This column automatically displays the type of each primitive joint contained in the Joint block. For Prismatic, there is only one primitive type, labeled Prismatic.



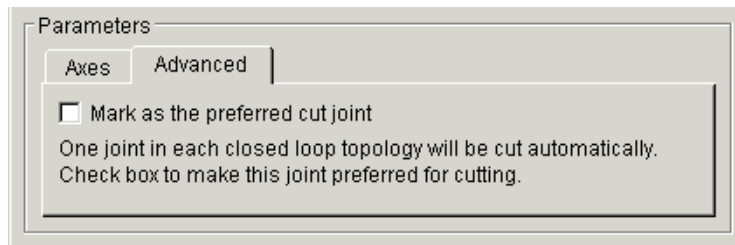
**Axis of translation [x y z]**

Enter here as a three-component vector the directional axis along which this translational DoF can move. The default vector is [0 0 1]. The axis is a directed vector whose overall sign matters.

**Reference csys**

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of translation is oriented with respect to. This CS also determines the absolute meaning of force and motion along the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.

**Mark as the preferred cut joint**

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

**See Also**

Disassembled Prismatic, Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator, Revolute, Spherical

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

# Revolute

---

## Purpose

Represent an assembled revolute joint with one rotational degree of freedom

## Library

Joints

## Description

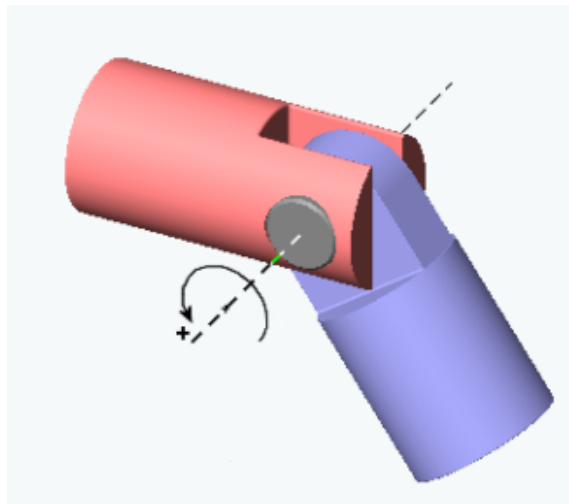


The Revolute block represents a single rotational degree of freedom (DoF) about a specified axis between two bodies. The rotational sense is defined by the right-hand rule. A revolute joint is one of SimMechanics primitive joints, along with prismatic and spherical.

The Revolute block is assembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, and the origins of these Body CSs must be spatially collocated points, to within assembly tolerances.

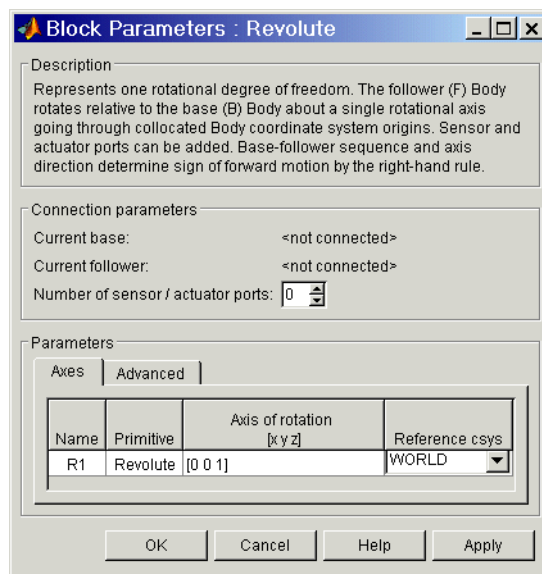
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.



**Revolute motion of follower (blue) relative to base (red)**

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Revolute base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Revolute base and follower Body connector ports” figure.

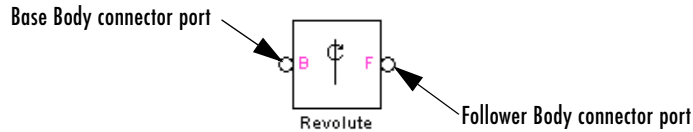
### Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

# Revolute

The motion of a Revolute is specified in angular units.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-handed sense about the rotation axis.

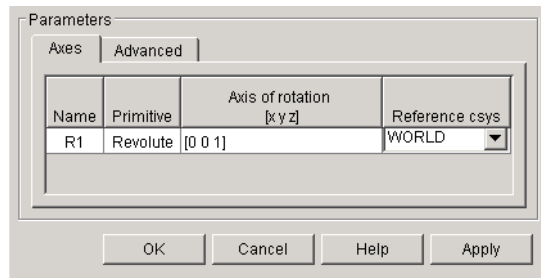


## Revolute base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. They specify the direction of the rotation axis of the DoF that the Revolute represents.



## Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Revolute, there is only one primitive joint, a revolute, labeled R1.

## Primitive

This column automatically displays the type of each primitive joint contained in the Joint block. For Revolute, there is only one primitive type, labeled Revolute.

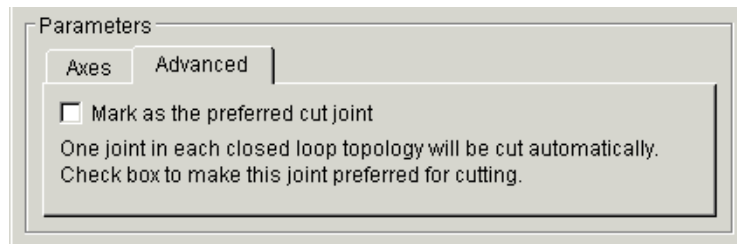
## Axis of rotation [x y z]

Enter here as a three-component vector the directional axis about which this rotational DoF can move. The default vector is [0 0 1]. The axis is a directed vector whose overall sign matters.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of rotation is oriented with respect to. This CS also determines the absolute meaning of torque and motion about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Disassembled Revolute, Joint Actuator, Joint Initial Condition Actuator, Joint Sensor, Joint Stiction Actuator, Prismatic, Spherical

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

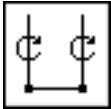
# Revolute-Revolute

---

**Purpose** Represent a composite joint composed of two revolute primitives spatially separated by a massless connector of constant length

**Library** Joints/Massless Connectors

**Description**



The Revolute-Revolute block represents a composite joint composed of two revolute joint primitives. The Body coordinate systems (CSs) on either side of the Joint are each connected to a revolute primitive. The primitives are separated spatially by a vector of constant length but variable direction connecting the two Body CS origins. Both revolute primitives are assembled.

---

**Caution** This joint becomes singular if the two revolute primitive axes align with the vector separating the primitives. The simulation stops with an error in this case.

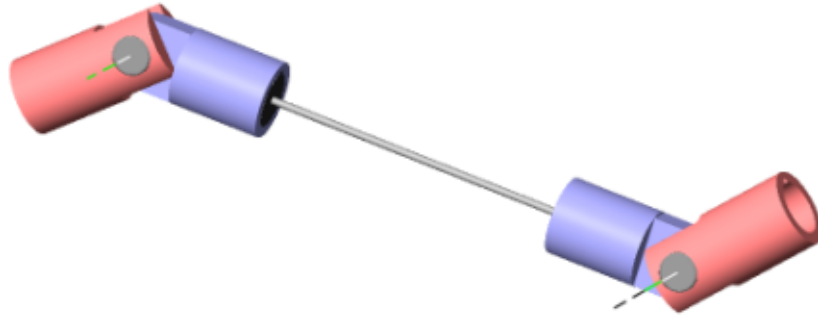
---

You specify the two revolute axes of these two joint primitives in the dialog box. The distance separation between the two axes is computed automatically from the Body CS origins to which the Joint is connected. This distance separation (the magnitude of the vector between the Body CS origins) remains fixed at its initial value during the simulation. This initial value must be nonzero.

You cannot connect an Actuator or Sensor to a Massless Connector.

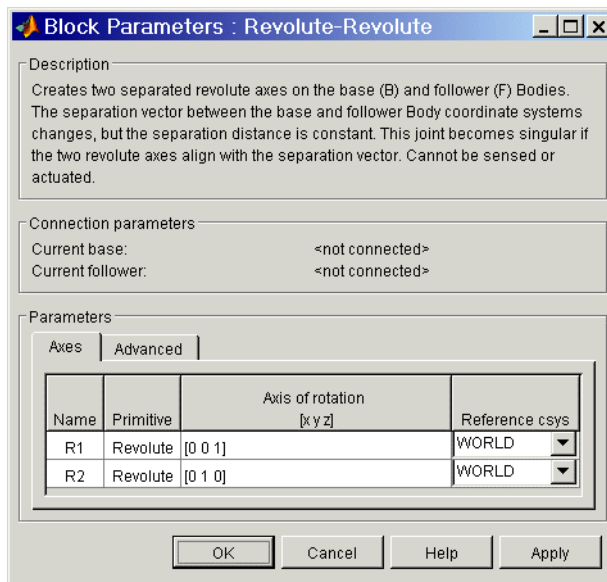
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the directions of the joint axes.



## Massless Connector between Revolute and Revolute Joints

### Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

# Revolute-Revolute

## Connection Parameters

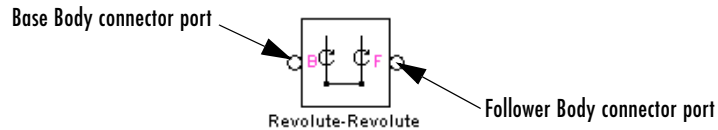
### Current base

When you connect the base (B) connector port on the Revolute-Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Revolute-Revolute base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Revolute-Revolute block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Revolute-Revolute base and follower Body connector ports” figure.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the base or follower rotating in the right-handed sense about its respective rotation axis.

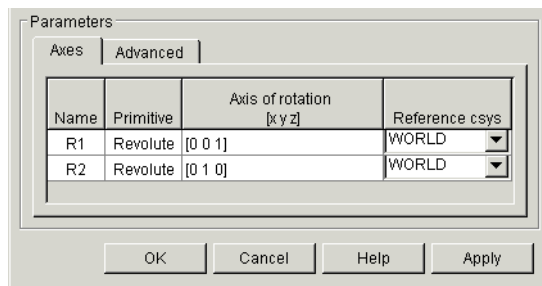


## Revolute-Revolute base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. They specify the direction of the rotation axes of these DoF's that the Revolute-Revolute represents.





## Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Revolute-Revolute, there are two revolute primitives, labeled R1 and R2, connecting to base and follower, respectively.

## Primitive

This column automatically displays the type of each primitive joint contained in the Joint block. For Revolute-Revolute, there is only one primitive type, labeled Revolute.

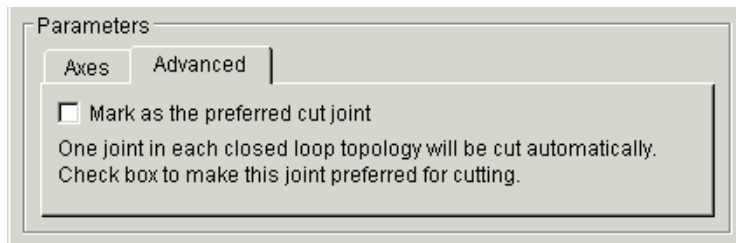
## Axis of rotation [x y z]

Enter here as a three-component vector the directional axis about which these rotational DoFs can move. The default vectors are [0 0 1] and [0 1 0]. The axes are directed vectors whose overall signs matter.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axes of rotation are oriented with respect to. These CSs also determine the absolute meaning of torque and motion about the primitive axes. The defaults are WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

# Revolute-Revolute

---

## See Also

Revolute

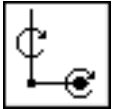
See “Modeling Joints” on page 4-17 for more on representing DoFs with Massless Connectors.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

**Purpose** Represent a composite joint composed of a revolute and a spherical primitive spatially separated by a massless connector of constant length

**Library** Joints/Massless Connectors

**Description**



The Revolute-Spherical block represents a composite joint composed of a revolute and a spherical joint primitive. The base Body coordinate system (CS) on one side of the Joint is connected to the revolute primitive, and the follower Body CS is connected to the spherical primitive. The primitives are separated spatially by a vector of constant length but variable direction connecting the two Body CS origins. Both primitives are assembled.

---

**Caution** This joint becomes singular if the revolute primitive axis aligns with the vector separating the primitives. The simulation stops with an error in this case.

---

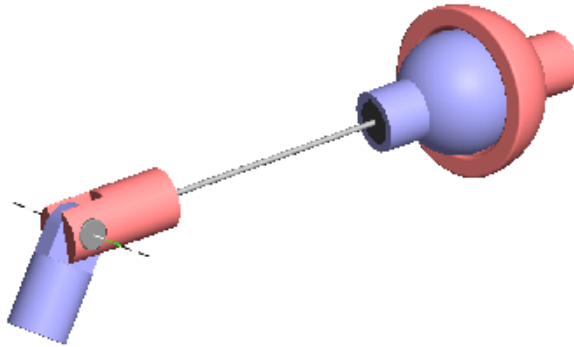
You specify the revolute axis of the revolute joint primitives in the dialog box. The distance separation between the two axes is computed automatically from the Body CS origins to which the Joint is connected. This distance separation (the magnitude of the vector between the Body CS origins) remains fixed at its initial value during the simulation. This initial value must be nonzero.

You cannot connect an Actuator or Sensor to a Massless Connector.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

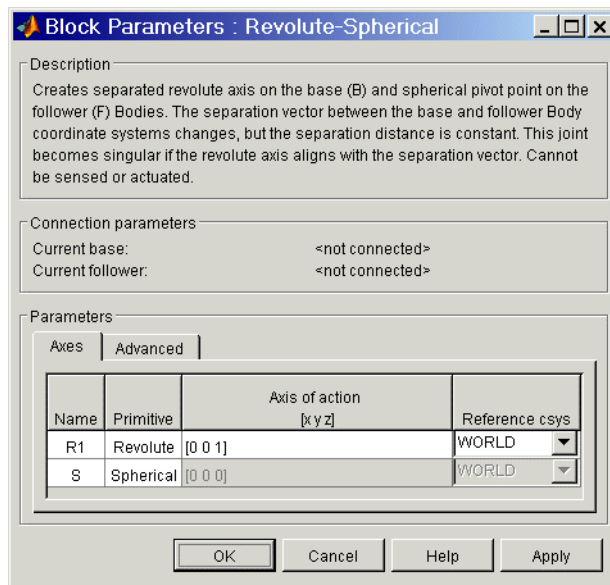
A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.

# Revolute-Spherical



## Massless Connector between Revolute and Spherical Joints

### Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

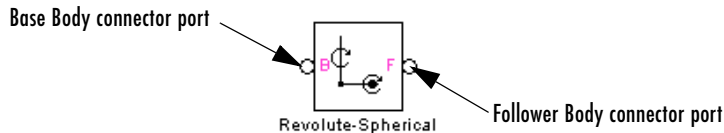
### Current base

When you connect the base (B) connector port on the Revolute-Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Revolute-Spherical base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Revolute-Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Revolute-Spherical base and follower Body connector ports” figure.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the base rotating in the right-handed sense about its rotation axis or the follower pivoting as shown for the Spherical Joint.

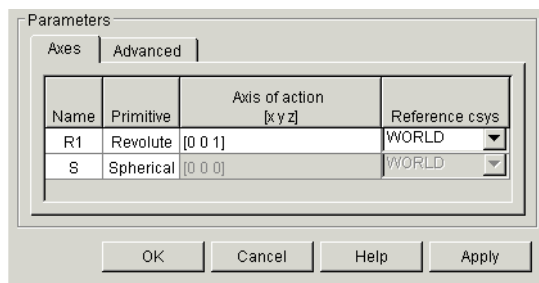


### Revolute-Spherical base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. They specify the direction of the rotation axis of one of the DoFs that Revolute-Spherical represents.



# Revolute-Spherical

---

## Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Revolute-Spherical, there are one revolute and one spherical primitive, labeled R1 and S, connecting to base and follower, respectively.

## Primitive

This column automatically displays the type of each primitive joint contained in the Joint block. For Revolute-Spherical, there are two primitive types, labeled Revolute and Spherical.

## Axis of action [x y z]

Enter here as a three-component vector the directional axis about which the rotational DoF can move. The default vector is [0 0 1]. The axis is a directed vector whose overall sign matters.

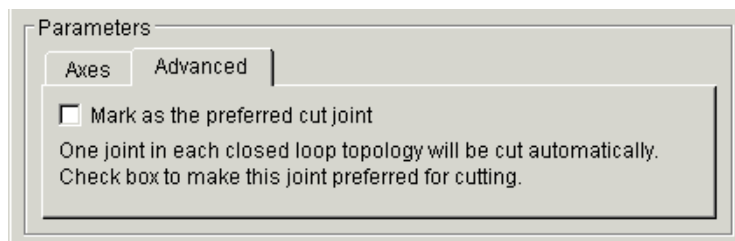
This field is not active for the Spherical primitive.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of rotation is oriented with respect to. This CS also determines the absolute meaning of torque and motion about the primitive axis. The default is WORLD.

This field is not active for the Spherical primitive.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## **Mark as the preferred cut joint**

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## **See Also**

Revolute, Spherical

See “Modeling Joints” on page 4-17 for more on representing DoFs with Massless Connectors.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

# RotationMatrix2VR

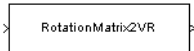
## Purpose

Convert a 3-by-3 rotation matrix to an equivalent VRML form of rotation axis and angle

## Library

Utilities

## Description

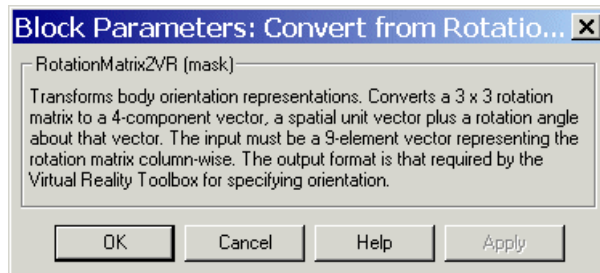


A rotation with respect to an initial orientation has many equivalent representations. A common and important one is the 3-by-3 orthogonal rotation matrix  $R$ , where  $R^{-1} = R^T$  and  $R^T R = R R^T = I$ , the 3-by-3 identity matrix. Another important representation is the combination of rotation axis (a unit vector  $\mathbf{n}$ ) and angle of rotation  $\theta$  about that axis. The sign of rotation follows the right-hand-rule.

The RotationMatrix2VR block converts the 3-by-3 rotation matrix representation of orientation to its equivalent representation as a rotation axis and angle about that axis, the form used in Virtual Reality Modeling Language (VRML) and the Virtual Reality Toolbox for orienting bodies. The input and output signals are bundled Simulink signals.

The most common use of rotations is to represent the orientation of a body with respect to some coordinate system (CS) axes.

## Dialog Box and Parameters



The dialog box has no active areas.



## Representations of rotation signals

The rotation matrix  $R$  has the form:

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

The input signal to the RotationMatrix2VR block is the  $R$  matrix components passed column-wise and bundled into a single 9-component Simulink signal: [R11 R21 R31 R12 ].

The output signal is the equivalent rotation represented as the axis of rotation, a unit vector  $\mathbf{n} = (n_x, n_y, n_z)$ , with

$$\mathbf{n}^* \mathbf{n} = n_x^2 + n_y^2 + n_z^2 = 1,$$

and the angle of rotation  $\theta$  about that axis. The sign of the rotation follows the right-hand rule. The output signal is bundled into a single 4-component Simulink signal:

$$[n_x \ n_y \ n_z \ \theta].$$

## See Also

Body

See “Representing Body Positions and Orientations” on page 3-2 for more details on body coordinate system rotations.

See entries on axis-angle rotation, Euler angles, quaternion, and rotation matrix in “Glossary” for summaries of body orientation representations.

For more on virtual reality and VRML, see the Virtual Reality Toolbox User’s Guide.

# Screw

---

## Purpose

Represent a composite joint with one translational DoF and one rotational DoF, with parallel translation and rotation axes and a linear pitch constraint between translational and rotational motion

## Library

Joints

## Description

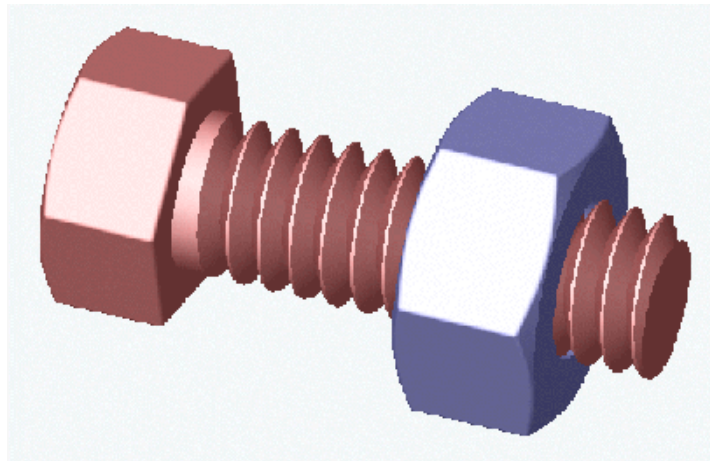


The Screw block represents a composite joint with one translational degree of freedom (DoF) as one prismatic primitive and one rotational DoF as one revolute primitive. The translation and rotation axes are parallel. The translational and rotational DoFs are constrained by a pitch constraint to have proportional motion.

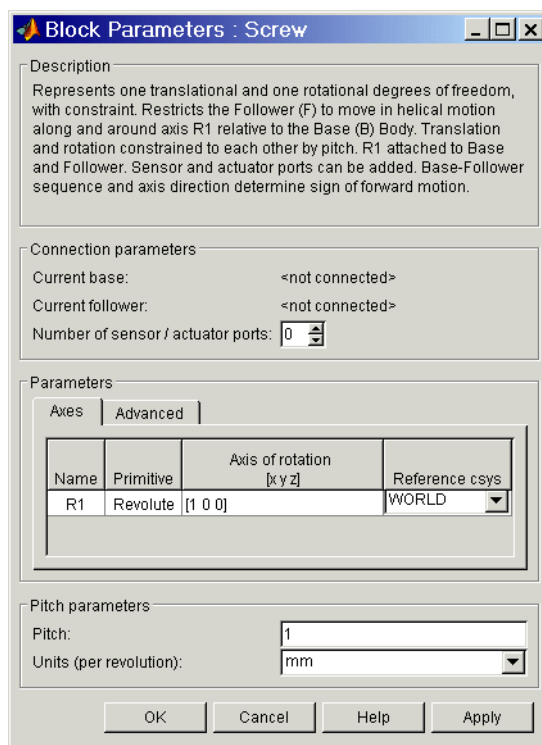
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Screw block is assembled: the origins of these Body CSs must lie along the primitive axes. But the Body CS origins on either side of the Joint do not need to be spatially collocated points.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.



## Dialog Box and Parameters



The dialog box has three active areas, **Connection parameters**, **Parameters**, and **Pitch parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Screw block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Screw base and follower Body connector ports” figure.

The base Body is automatically connected to the joint primitive R1 in the primitive list in **Parameters**.

# Screw

## Current follower

When you connect the follower (F) connector port on the Screw block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Screw base and follower Body connector ports” figure.

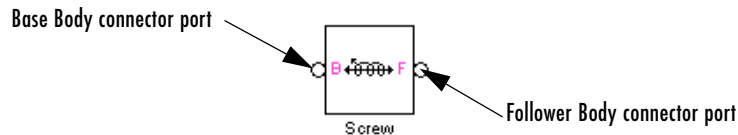
The follower Body is automatically connected to the joint primitive R1 in the primitive list in **Parameters**.

## Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motion of revolute primitives is specified in angular units.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower moving around the rotational axis following the right-hand rule.



## Screw base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Screw has an entry line. These lines specify the direction of the axes of action of the DoFs that the Screw represents.

### Name - Primitive

The primitive list states the name and type of the joint primitive that makes up the Screw block: revolute primitive R1.

### Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of this primitive and its corresponding DoF:

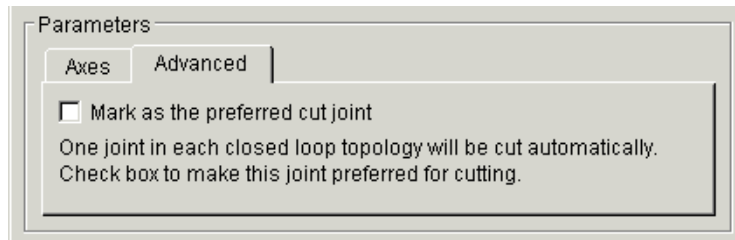
- Revolute: axis of rotation

The default vectors are shown in the dialog box above. The axis is a directed vector whose overall sign matters.

### Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



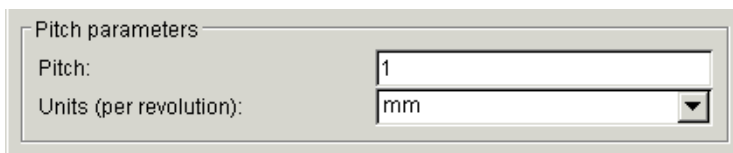
### Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## Pitch Parameters

The **Pitch parameters** control how far the screw translates for each revolution.



### Pitch

Linear distance the screw travels along the screw axis for each complete revolution (turn) of  $2\pi$  radians ( $360^\circ$ ). The default is 1.

# Screw

---

## **Units (per revolution)**

In the pull-down menu, choose the units for the pitch linear distance. The default is mm (millimeters).

## **See Also**

Cylindrical, Prismatic, Revolute

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

**Purpose** Represent a composite joint with three translational and three rotational DoFs

**Library** Joints

**Description**



The Six-DoF block represents a composite joint with three translational degrees of freedom (DoFs) as three prismatic primitives and three rotational DoFs as one spherical primitives. There are no constraints among the primitives. Unlike Bushing, Six-DoF represents the rotational DoFs as one spherical, rather than as three revolutes.

---

**Caution** A joint with three prismatic primitives becomes singular if two or three of the translation axes become parallel. The simulation stops with as error in this case.

---

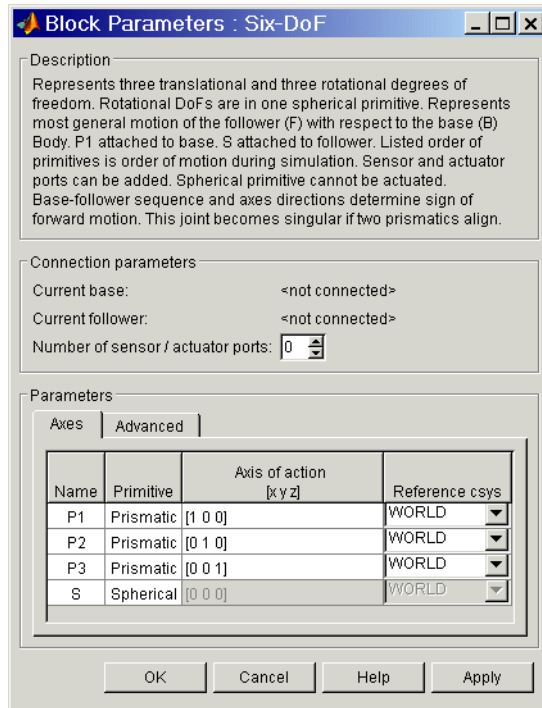
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Six-DoF block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.

# Six-DoF

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Six-DoF block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Six-DoF base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive P1 in the primitive list in **Parameters**.

### Current follower

When you connect the follower (F) connector port on the Six-DoF block to a Body CS Port on a Body, this parameter is automatically reset to the name



of this Body CS. See the following “Six-DoF base and follower Body connector ports” figure.

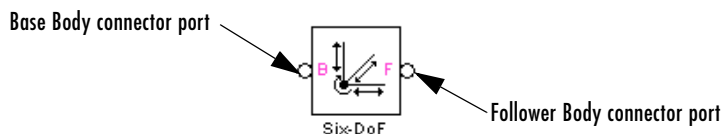
The follower Body is automatically connected to the last joint primitive S in the primitive list in **Parameters**.

### Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motion of prismatic primitives is specified in linear units. The motion of spherical primitives is specified by a dimensionless quaternion.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive spherical motion is the follower rotating in the right-handed sense as shown in the Spherical block figure.



### Six-DoF base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Six-DoF has an entry line. These lines specify the direction of the axes of action of the DoFs that the Six-DoF represents.

### Name - Primitive

The primitive list states the names and types of joint primitives that make up the Six-DoF block: prismatic primitives P1, P2, P3, and spherical primitive S.

### Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Prismatic: axis of translation

- Spherical: field is not active

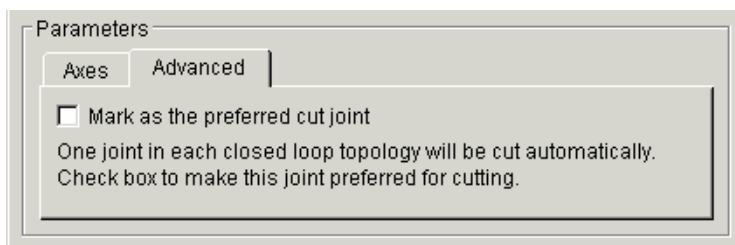
The default vectors are shown in the dialog box above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, no two of the prismatic axes can be parallel.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Bushing, Gimbal, Prismatic, Spherical

See "Modeling Joints" on page 4-17 for more on representing DoFs with Joints.

See "Checking Schematic Topology" on page 4-64 and "How SimMechanics Works" on page 5-14 for more on closed loops and cutting.

**Purpose** Represent an assembled spherical joint with three rotational degrees of freedom

**Library** Joints

**Description**



The Spherical block represents three rotational degrees of freedom (DoFs) at a single pivot point, a “ball-in-socket” joint. Two rotational DoFs specify a directional axis, and a third rotational DoF specifies rotation about that directional axis. The sense of each rotational DoF is defined by the right-hand rule, and the three rotations together form a right-handed system. A spherical joint is one of the SimMechanics primitive joints, along with prismatic and revolute.

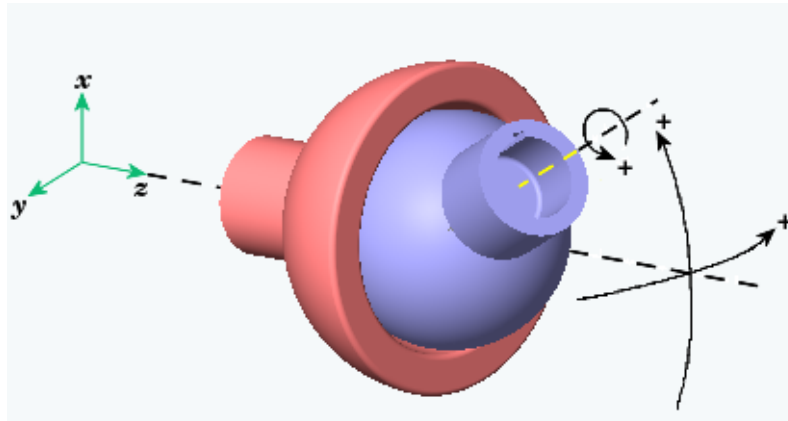
The Spherical block is assembled: you must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point, and the origins of these Body CSs must be spatially collocated points, within assembly tolerances.

You cannot connect an Actuator to a Spherical. Unlike the Gimbal block, the Spherical block cannot become singular.

You can connect all Joint blocks to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

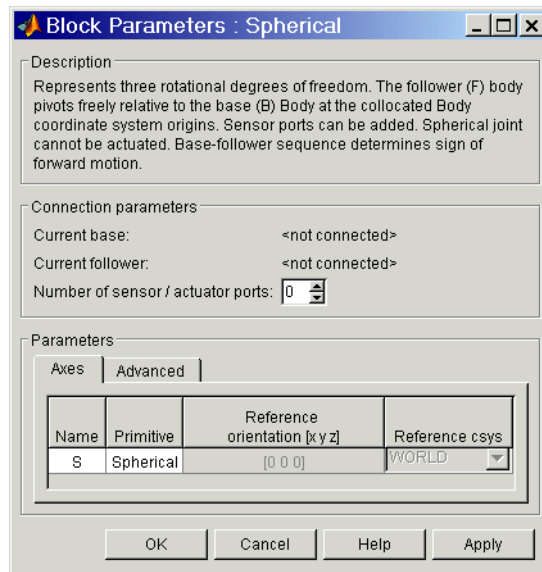
Any Joint block represents only the abstract relative motion of two bodies, not the bodies themselves.

# Spherical



**Spherical motion of follower (blue) relative to base (red)**

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Spherical base and follower Body connector ports” figure.

### Current follower

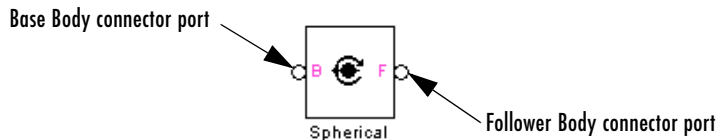
When you connect the follower (F) connector port on the Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Spherical base and follower Body connector ports” figure.

### Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Sensor blocks to this Joint. The default is 0. A Spherical cannot be connected to a Joint Actuator.

The motion of a Spherical is three DoFs specified in quaternion form.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower rotating in the right-handed sense as shown in the figure above.



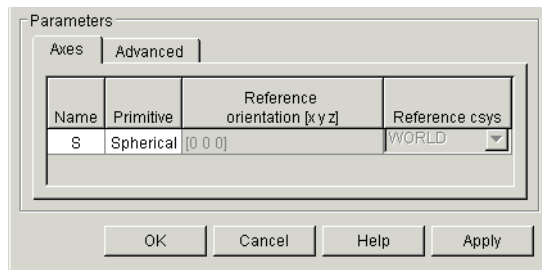
### Spherical base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are automatic. They specify the orientation of the spherical DoF that the Spherical represents.

# Spherical



## Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Spherical, there is only one primitive joint, a spherical, labeled S.

## Primitive

This column automatically displays the type of each primitive joint contained in the Joint block. For Spherical, there is only one primitive type, labeled Spherical.

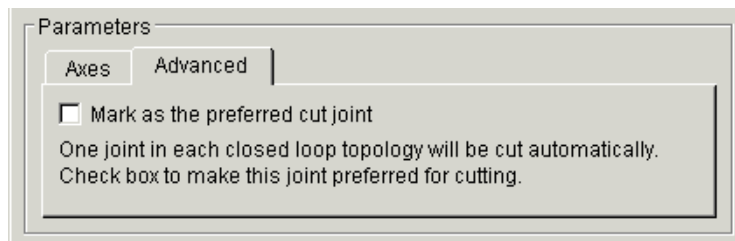
## Reference orientation [x y z]

This field is not active.

## Reference csys

This field is not active.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## **Mark as the preferred cut joint**

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## **See Also**

Disassembled Spherical, Gimbal, Joint Sensor, Prismatic, Revolute

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

# Spherical-Spherical

---

**Purpose** Represent a composite joint composed of two spherical primitives spatially separated by a massless connector of constant length

**Library** Joints/Massless Connectors

## Description



The Spherical-Spherical block represents a composite joint composed of two spherical joint primitives. The Body coordinate systems (CSs) on either side of the Joint are connected to the spherical primitives. The primitives are separated spatially by a vector of constant length but variable direction connecting the two Body CS origins. Both primitives are assembled.

The distance separation between the two axes is computed automatically from the Body CS origins to which the Joint is connected. This distance separation (the magnitude of the vector between the Body CS origins) remains fixed at its initial value during the simulation. This initial value must be nonzero.

You cannot connect an Actuator or Sensor to a Massless Connector.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

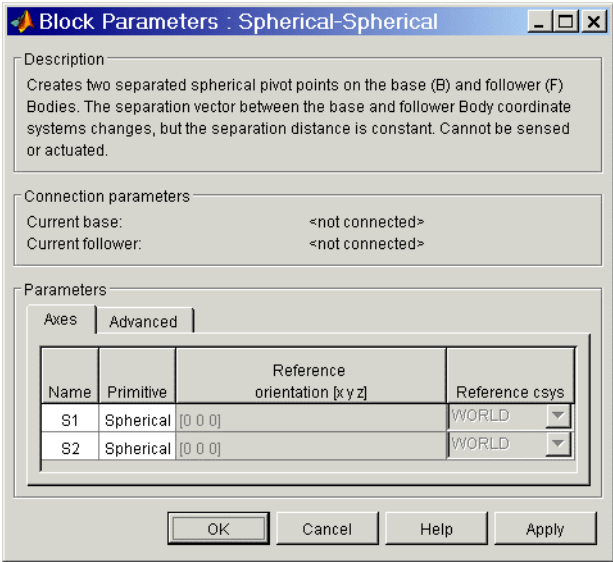
A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the directions of the joint axes.



**Massless Connector between Spherical and Spherical Joints**



## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

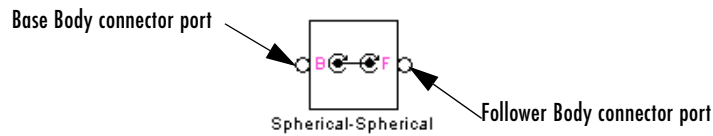
When you connect the base (B) connector port on the Spherical-Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Spherical-Spherical base and follower Body connector ports” figure.

### Current follower

When you connect the follower (F) connector port on the Spherical-Spherical block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Spherical-Spherical base and follower Body connector ports” figure.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the base or follower pivoting as shown by the motion figure in the Spherical block reference page.

# Spherical-Spherical

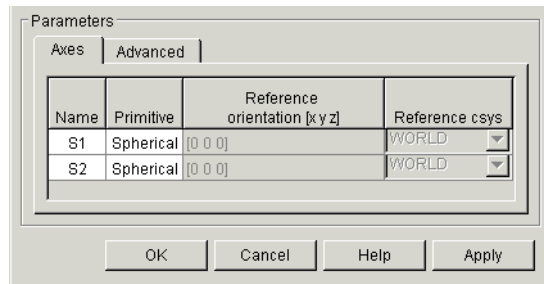


## Spherical-Spherical base and follower Body connector ports

### Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are automatic. They specify the orientation of the spherical DoFs that the Spherical-Spherical represents.



### Name

This column automatically displays the name of each primitive joint contained in the Joint block. For Spherical-Spherical, there are two spherical primitives, labeled S1 and S2, connecting to base and follower, respectively.

### Primitive

This column automatically displays the type of each primitive joint contained in the Joint block. For Spherical-Spherical, there is only one primitive type, labeled Spherical.

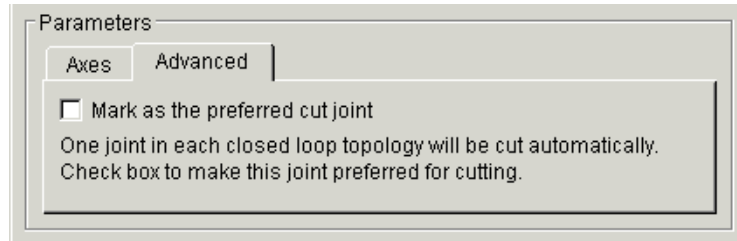
### Reference orientation [x y z]

These fields are not active.

### Reference csys

These fields are not active.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



### Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

### See Also

Spherical

See “Modeling Joints” on page 4-17 for more on representing DoFs with Massless Connectors.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

# Telescoping

---

**Purpose** Represent a composite joint with one translational and three rotational DoFs

**Library** Joints

**Description**

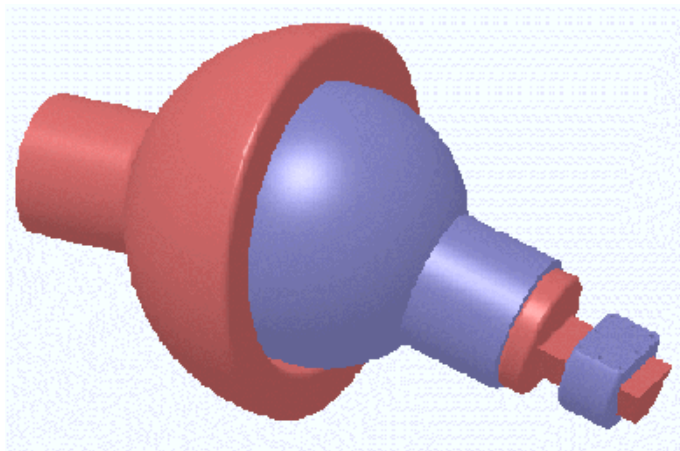


The Telescoping block represents a composite joint with one translational degree of freedom (DoF) as one prismatic primitive and three rotational DoFs as one spherical primitive. There are no constraints among the primitives. Unlike Bearing, Telescoping represents the rotational DoFs as one spherical, rather than as three revolutes.

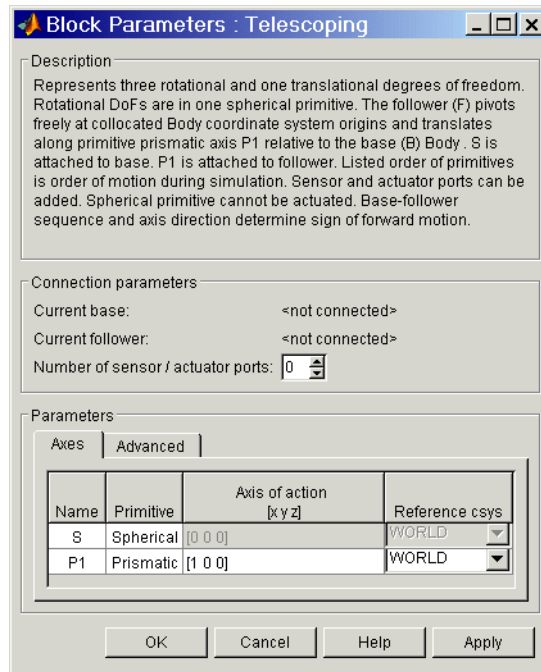
You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Telescoping block is assembled: the origins of these Body CSs must lie along the primitive axes, and the Body CS origins on either side of the Joint must be spatially collocated points, to within assembly tolerances.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify a reference CS to define the direction of the joint axis.



## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Telescoping block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Telescoping base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive S in the primitive list in **Parameters**.

### Current follower

When you connect the follower (F) connector port on the Telescoping block to a Body CS Port on a Body, this parameter is automatically reset to the

# Telescoping

name of this Body CS. See the following “Telescoping base and follower Body connector ports” figure.

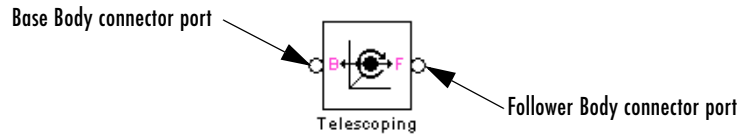
The follower Body is automatically connected to the last joint primitive P1 in the primitive list in **Parameters**.

## Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motion of prismatic primitives is specified in linear units. The motion of spherical primitives is specified by a dimensionless quaternion.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive spherical motion is the follower rotating in the right-handed sense as shown in the Spherical block figure.



## Telescoping base and follower Body connector ports

### Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Telescoping has an entry line. These lines specify the direction of the axes of action of the DoFs that Telescoping represents.

#### Name - Primitive

The primitive list states the names and types of joint primitives that make up the Telescoping block: spherical primitive S and prismatic primitives P1.

#### Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

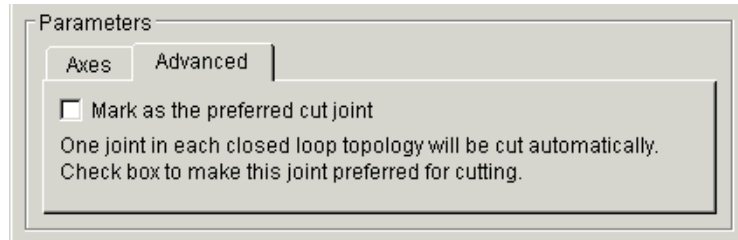
- Prismatic: axis of translation
- Spherical: field is not active

The default vectors are shown in the dialog box above. The axis is a directed vector whose overall sign matters.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Bearing, Prismatic, Six-DoF, Spherical

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.

# Universal

---

**Purpose** Represent a composite joint with two rotational DoFs

**Library** Joints

**Description**



The Universal block represents a composite joint with two rotational degrees of freedom (DoFs) as two revolute primitives. There are no constraints among the primitives.

---

**Caution** A joint with two revolute primitives becomes singular if the two rotation axes become parallel (“gimbal lock”). The simulation stops with an error in this case.

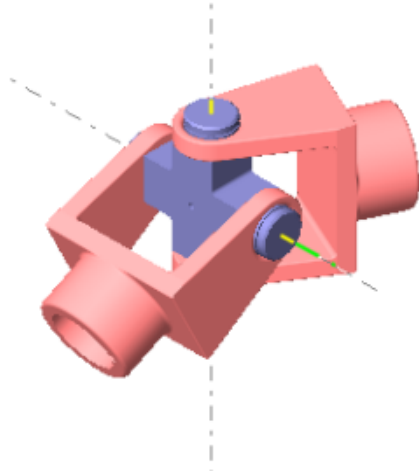
---

You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Universal block is assembled: the origins of these Body CSs must be spatially collocated points, within assembly tolerances.

You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

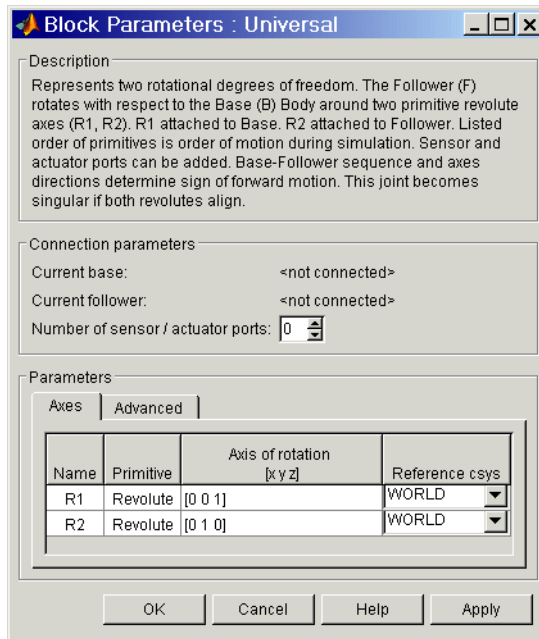


A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves. You must specify reference CSs to define the directions of the joint axes.



# Universal

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

### Connection Parameters

#### Current base

When you connect the base (B) connector port on the Universal block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Universal base and follower Body connector ports” figure.

The base Body is automatically connected to the first joint primitive R1 in the primitive list in **Parameters**.

#### Current follower

When you connect the follower (F) connector port on the Universal block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Universal base and follower Body connector ports” figure.

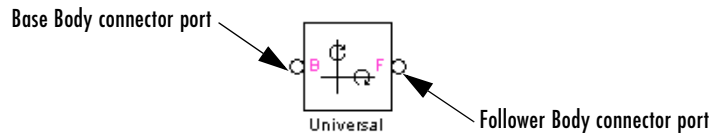
The follower Body is automatically connected to the last joint primitive R2 in the primitive list in **Parameters**.

## Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Actuator and Joint Sensor blocks to this Joint. The default is 0.

The motion of revolute primitives is specified in angular units.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive rotation is the follower moving around the rotational axis following the right-hand rule.



## Universal base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are required. Each DoF primitive in Universal has an entry line. These lines specify the direction of the axes of action of the DoFs that the Universal represents.

### Name - Primitive

The primitive list states the names and types of joint primitives that make up the Universal block: revolute primitives R1, R2.

### Axis of action [x y z]

Enter here as a three-component vector the directional axes defining the allowed motions of these primitives and their corresponding DoFs:

- Revolute: axis of rotation

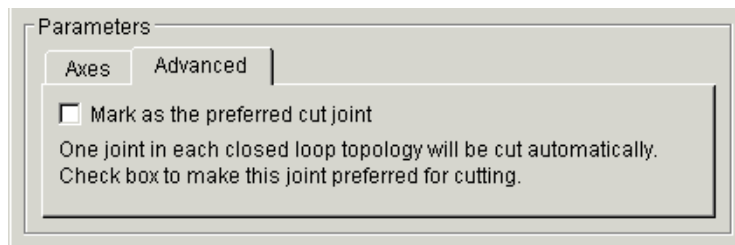
The default vectors are shown in the dialog box above. The axis is a directed vector whose overall sign matters.

To prevent singularities and simulation errors, the two revolute axes cannot be parallel.

## Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



## Mark as the preferred cut joint

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

## See Also

Gimbal, Revolute

See "Modeling Joints" on page 4-17 for more on representing DoFs with Joints.

See "Checking Schematic Topology" on page 4-64 and "How SimMechanics Works" on page 5-14 for more on closed loops and cutting.

# Variable Mass & Inertia Actuator

---

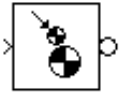
## Purpose

Varies the mass and inertia on a body at a specific body coordinate system as a function of time (*not* including thrust force or torque)

## Library

Sensors & Actuators

## Description



The Variable Mass & Inertia Actuator block allows you to vary the mass  $m$  and/or inertia tensor  $\mathbf{I}$  of the Body to which it is connected. The general form of Newton's second law for linear or angular motion is

$$(\text{mass or inertia}) * \text{acceleration} = \text{external force or torque}$$

This block externally varies the leftmost parameter in this law of motion with a Simulink signal.

---

**Note** The Variable Mass & Inertia Actuator does *not* apply any thrust forces or torques associated with the Body's mass loss or gain. Such thrust effects would occur on the left-hand side of the force or torque law as terms proportional to the time derivatives of the mass or inertia tensor,  $dm/dt$  or  $d\mathbf{I}/dt$ , multiplied by the related thrust velocities. You must separately apply such thrust forces or torques to the Body with Body Actuators.

---

## How the Actuator Varies a Body's Mass and Inertia Tensor

You connect the Variable Mass & Inertia Actuator block to the original, user-supplied Body at a Body coordinate system (CS). You can connect multiple Variable Mass & Inertia Actuators to a single Body, each Actuator at a separate Body CS port. If Body CS ports are lacking, open the Body dialog and create them as needed.

At each Body CS so connected, the Variable Mass & Inertia Actuator creates an invisible body. The attachment is equivalent to connecting another Body with a Weld, except that the other body's mass properties vary with time. This invisible body has a time-varying mass and/or symmetric inertia tensor supplied by the external Simulink signal. The center of gravity coordinate system (CG CS) of the invisible body is identical to the attached Body CS. The inertia tensor of the invisible body is evaluated at this CS, in this coordinate system's axes.

# Variable Mass & Inertia Actuator

---

SimMechanics creates a combined or *composite* body, made of the invisible, time-varying body created by the Actuator and the original, user-supplied Body. The total mass of the composite body is the sum of the visible Body and the invisible body's masses. The CG of this composite body is recomputed at each time step. The inertia tensor of the composite body is formed at each time step by combining the inertia tensors of the visible Body and the invisible body. The combined inertia tensor is then evaluated at the composite body's new CG.

The time-varying mass and inertia tensor of the invisible body must satisfy these requirements:

- The mass and principal inertial moments can be positive, negative, or zero. The only restriction is that the total mass and the principal inertial moments of the composite body be nonnegative.
- The time-varying inertia tensor of the invisible body must be symmetric.

You can mix variable mass and/or variable inertia tensor actuation.

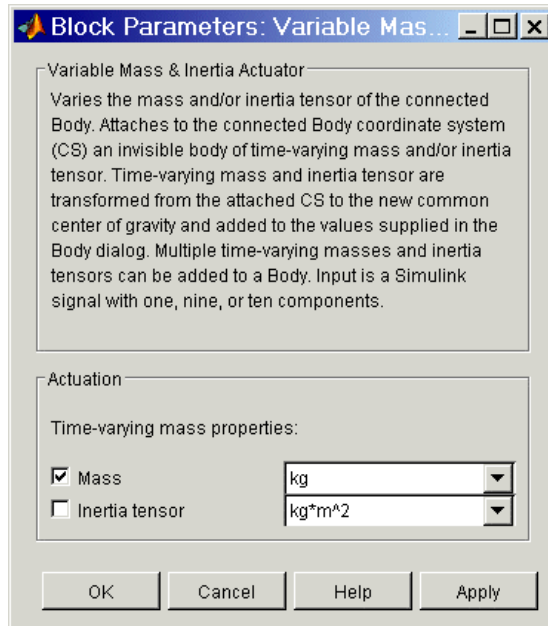
<b>Actuation</b>	<b>Effect on Connected Body</b>
Variable mass alone	Adds a time-varying point mass at the attached Body CS
Variable inertia tensor alone	Adds time-varying inertia tensor at the attached Body CS without changing the composite body's total mass
Variable mass and inertia tensor combined	Adds invisible body with time-varying mass and inertia tensor at the attached Body CS

## What Does Not Vary in the User-Supplied Body

While the invisible, attached body and the invisible composite body have time-varying mass properties, you do not see any visible changes in the original Body that you are actuating. The mass properties in its dialog do not change.

If you are visualizing the varying-mass/inertia actuated Body as an equivalent ellipsoid, the ellipsoid is rendered using the static data in the Body dialog itself. The ellipsoid rendering ignores the effect of any Variable Mass & Inertia Actuators attached to the Body. See "Rendering Body Shapes in SimMechanics" on page 6-5.

## Dialog Box and Parameters



The dialog box has one active area, **Actuation**.

### Actuation

You can apply a variable mass, a variable inertia tensor, or both, to a body.

If you apply both, you need to bundle the variable mass and inertia tensor into a 10-component signal, in the order shown in the dialog box.

### Mass

Select the check box to apply an external time-varying mass from the input Simulink signal. In the pull-down menu to the right, select units for this time-varying mass. The default is kg (kilograms).

### Inertia tensor

Select the check box to apply an external time-varying inertia tensor from the input Simulink signal. In the pull-down menu to the right, select units for this time-varying inertia tensor. The default is  $\text{kg}\cdot\text{m}^2$  (kilogram-meters<sup>2</sup>).

# Variable Mass & Inertia Actuator

---

The Simulink input signal has the following components. For variable mass or inertia tensor actuation alone, omit the missing components.

Time-varying mass (scalar)	Time-varying inertia tensor (9-vector): ( $I_{11}, I_{21}, I_{31}, I_{12}, \dots$ )
----------------------------	--

## References

Corbin, H. C., and P. Stehle, *Classical Mechanics*, Second Edition, New York, Dover Publications, 1994 (original edition, 1960), chapters 2, 5, and 9.

Goldstein, H., *Classical Mechanics*, Second Edition, Reading, Massachusetts, Addison-Wesley, 1980, chapters 4 and 5.

Piscane, V. L., and R. C. Moore, eds., *Fundamentals of Space Systems*, Johns Hopkins University/Applied Physics Laboratory Series, New York, Oxford University Press, 1994, chapters 3, 4, and 5.

## See Also

Body, Body Actuator, Weld

See “Varying a Body’s Mass and Inertia Tensor” on page 4-43 for more on varying the mass and inertia tensor of a body.



**Purpose** Specify a linear combination of the linear and angular velocities of two bodies as a function of time

**Library** Constraints & Drivers

## Description



The Velocity Driver block drives a linear combination of the projected translational and angular velocities of two Bodies. The velocities are projected by inner products on to constant vectors you specify.

Let  $\mathbf{v}_B$ ,  $\mathbf{v}_F$  be the two body velocity vectors and  $\omega_B$ ,  $\omega_F$  be the two body angular velocity vectors. Let  $\mathbf{c}_B$ ,  $\mathbf{c}_F$ ,  $\mathbf{d}_B$ ,  $\mathbf{d}_F$  be constant vectors. The subscripts 'B' and 'F' refer to base and follower bodies. The Velocity Driver block specifies this linear combination:

$$\mathbf{c}_B \cdot \mathbf{v}_B + \mathbf{d}_B \cdot \omega_B - \mathbf{c}_F \cdot \mathbf{v}_F - \mathbf{d}_F \cdot \omega_F = f(t)$$

as a function of time  $f(t)$ . You specify the vectors  $\mathbf{c}_B$ ,  $\mathbf{c}_F$ ,  $\mathbf{d}_B$ ,  $\mathbf{d}_F$ . You also connect a Driver Actuator block to the Velocity Driver.

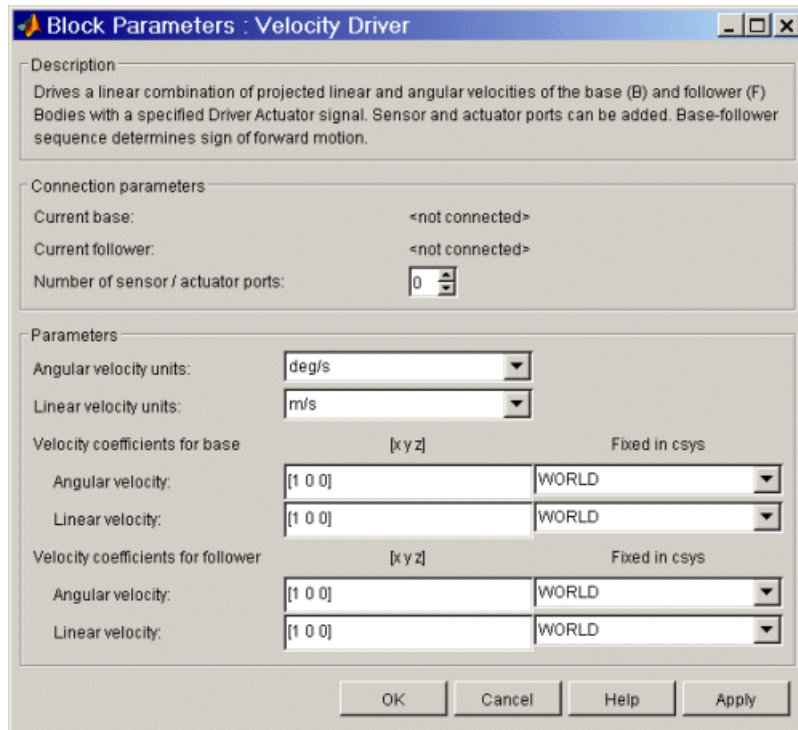
The Simulink input signal into the Driver Actuator specifies the time-dependent driving function  $f(t)$  and its first two derivatives, as well as their units. If you do not actuate Velocity Driver, this block acts as a time-independent constraint that freezes the constraint linear combination at its initial value during the simulation.

Drivers restrict relative degrees of freedom (DoFs) between a pair of bodies as specified functions of time. Locally in a machine, they replace a Joint as the expression of the DoFs. Globally, Driver blocks must occur topologically in closed loops. Like Bodies connected to a Joint, the two Bodies connected to a Drivers are ordered as base and follower, fixing the direction of relative motion.

You can also connect a Constraint & Driver Sensor to any Driver block and measure the reaction forces/torques between the driven bodies.

# Velocity Driver

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

## Connection Parameters

### Current base

When you connect the base (B) connector port on the Velocity Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Velocity Driver base and follower Body connector ports” figure.

### Current follower

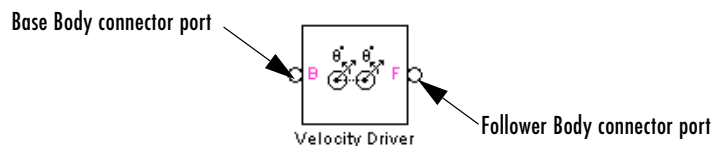
When you connect the follower (F) connector port on the Velocity Driver block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Velocity Driver base and follower Body connector ports” figure.

## Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Driver Actuator and Constraint & Driver Sensor blocks to this Driver. The default is 0.

To activate the Driver, connect a Driver Actuator.

The base (B)-follower (F) Body sequence determines the sense of positive motion. Positive translation is the follower moving in the direction of the translation axis. Positive rotation is the follower rotating in the right-handed sense about the rotation axis.



## Velocity Driver base and follower Body connector ports

### Parameters

#### Angular velocity units

From the pull-down menu, choose the common units for all angular velocities. The default is deg/s (degrees/second).

The vectors  $\mathbf{d}_B$  and  $\mathbf{d}_F$  implicitly carry the units conversion of length/angle. The driving function  $f(t)$  has the linear velocity units that you set in the Driver Actuator block that you connect to Velocity Driver. If the  $f(t)$  units differ from the units set in **Linear velocity units** in this dialog box, the vectors  $\mathbf{d}_B$  and  $\mathbf{d}_F$  implicitly carry the additional units conversion.

#### Linear velocity units

From the pull-down menu, choose the common units for all linear velocities. The default is m/s (meters/second).

The driving function  $f(t)$  has the linear velocity units that you set in the Driver Actuator block that you connect to the Velocity Driver. If the  $f(t)$  units differ from the units set here, the vectors  $\mathbf{c}_B$  and  $\mathbf{c}_F$  implicitly carry the units conversion.

# Velocity Driver

---

## Velocity coefficients for base

Under [**x y z**], enter the **Angular velocity** and **Linear velocity** coefficient vectors for the base Body. These are the components of  $\mathbf{d}_B$  and  $\mathbf{c}_B$ , respectively. The defaults are [1 0 0].

In the pull-down menus, choose the coordinate systems (WORLD or BASE) whose coordinate axes the vectors  $\mathbf{d}_B$  and  $\mathbf{c}_B$  are oriented with respect to. The defaults are WORLD.

The vectors  $\mathbf{d}_B$  and  $\mathbf{c}_B$  carry the implicit units conversion to convert all velocities to the common linear velocity units of  $f(t)$  that you set in the connected Driver Actuator block.

## Velocity coefficients for follower

Under [**x y z**], enter the **Angular velocity** and **Linear velocity** coefficient vectors for the follower Body. These are the components of  $\mathbf{d}_F$  and  $\mathbf{c}_F$ , respectively. The defaults are [1 0 0].

In the pull-down menus, choose the coordinate systems (WORLD or FOLLOWER) whose coordinate axes the vectors  $\mathbf{d}_F$  and  $\mathbf{c}_F$  are oriented with respect to. The defaults are WORLD.

The vectors  $\mathbf{d}_F$  and  $\mathbf{c}_F$  carry the implicit units conversion to convert all velocities to the common linear velocity units of  $f(t)$  that you set in the connected Driver Actuator block.

## See Also

Angle Driver, Constraint & Driver Sensor, Driver Actuator, Parallel Constraint

See “Modeling Constraints and Drivers” on page 4-34 for more on restricting DoFs with Drivers.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on using drivers in closed loops.

See “Constraining and Driving Motion” on page 8-6.

**Purpose** Represent a joint with no DoFs

**Library** Joints

**Description**

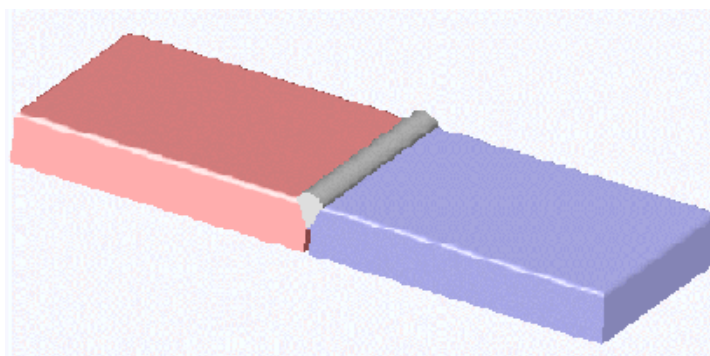


The Weld block represents a joint with no degrees of freedom (DoFs). The two Bodies connected to either side of the Weld block are locked rigidly to one another, with no possible relative motion.

You must connect each side of the Joint block to a Body block at a Body coordinate system (CS) point. The Weld block is assembled: the origins of these Body CSs must lie along the primitive axes, within assembly tolerances. But the Body CS origins on either side of the Joint do not have to be spatially collocated points.

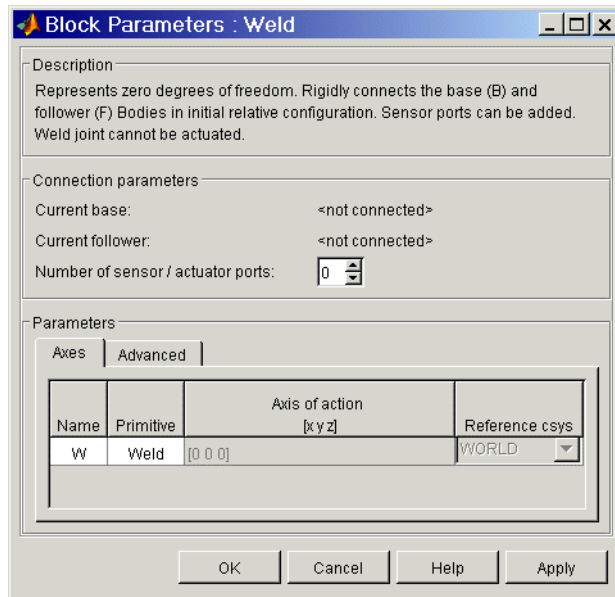
You must connect any Joint block to two and only two Body blocks, and Joints have a default of two connector ports for connecting to base and follower Bodies.

A Joint block represents only the abstract relative motion of two bodies, not the bodies themselves.



# Weld

## Dialog Box and Parameters



The dialog box has two active areas, **Connection parameters** and **Parameters**.

### Connection Parameters

#### Current base

When you connect the base (B) connector port on the Weld block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Weld base and follower Body connector ports” figure.

The base Body is automatically connected to the joint primitive W in the primitive list in **Parameters**.

#### Current follower

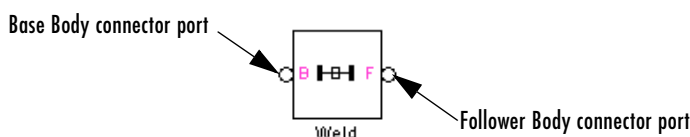
When you connect the follower (F) connector port on the Bushing block to a Body CS Port on a Body, this parameter is automatically reset to the name of this Body CS. See the following “Weld base and follower Body connector ports” figure.

The follower Body is automatically connected to the joint primitive W in the primitive list in **Parameters**.

### Number of sensor/actuator ports

Using this spinner menu, you can set the number of extra connector ports needed for connecting Joint Sensor blocks to this Joint. The default is 0.

You cannot actuate a Weld joint, and a Weld joint undergoes no motion. A Joint Sensor measures zero motion, but in general nonzero reaction forces, at this joint.



### Weld base and follower Body connector ports

## Parameters

Toggle between the **Axes** and **Advanced** panels with the tabs.

The entries on the **Axes** pane are inactive for Weld. This block has no DoF primitives.

### Name - Primitive

The primitive list states the names and types of joint primitives that make up the Weld block: a rigid primitive W representing no motion.

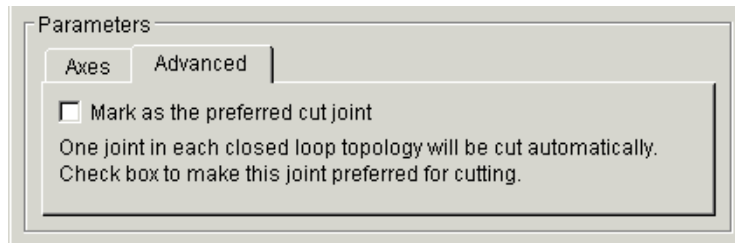
### Axis of action [x y z]

This field is inactive.

### Reference csys

Using the pull-down menu, choose the coordinate system (World, the base Body CS, or the follower Body CS) whose coordinate axes the vector axis of action is oriented with respect to. This CS also determines the absolute meaning of forces/torques and motion along/about the joint axis. The default is WORLD.

The **Advanced** pane is optional. You use it to control the way SimMechanics interprets the topology of your schematic diagram.



### **Mark as the preferred cut joint**

In a closed loop, one and only one joint is cut during the simulation. SimMechanics does the cutting internally and automatically.

If you want this particular joint to be weighted preferentially for cutting during the simulation, select the check box. The default is unselected.

### **See Also**

Distance Driver

See “Modeling Joints” on page 4-17 for more on representing DoFs with Joints.

See “Checking Schematic Topology” on page 4-64 and “How SimMechanics Works” on page 5-14 for more on closed loops and cutting.



# SimMechanics Command Reference

---

This table indicates the tasks performed by the command described in this chapter.

<b>Command</b>	<b>Purpose</b>
<code>mech_stateVectorMgr</code>	Display and set machine state vector

# mech\_stateVectorMgr

---

**Purpose** Display and set the machine state with a state vector manager

**Synopsis** You must call `mech_stateVectorMgr` with one argument, the pathname or handle of any block in the machine whose state you want:

```
MachineState = mech_stateVectorMgr('pathname')  
MachineState = mech_stateVectorMgr('handle')
```

You obtain the pathname and handle with the Simulink `gcb` and `gcbh` commands.

You can also call `mech_stateVectorMgr` with an indirect call to the block pathname or handle. Select one of the SimMechanics blocks in the machine and enter one of these commands:

```
MachineState = mech_stateVectorMgr(gcb)  
MachineState = mech_stateVectorMgr(gcbh)
```

The command `mech_stateVectorMgr` returns an object `MachineState`.

The state manager object includes only the state of a machine made of SimMechanics blocks. Simulink associates the machine to one of the machine's Ground blocks.

**Description** The total number of state components is

$$N = 2 * (\# \text{ of prismatic} + \# \text{ of revolute}) + 8 * (\# \text{ of spherical}) \\ + (\# \text{ of Point-Curve Constraints}),$$

not including any motion-actuated joint primitives. The machine state consists of all the linear/angular positions and velocities of all degrees of freedom (DoFs) in the machine:

- Prismatic and revolute joint primitives each have two state components, a position and a velocity.
- Spherical joint primitives each have eight state components, a quaternion and a quaternion derivative.
- A joint primitive actuated by a Joint Initial Condition Actuator is counted like other joint primitives. But because the JICA externally specifies such a primitive's initial position and velocity, the primitive has an active `FixedAtT_0` flag.

- Point-Curve Constraints each have one predictor state component, the arc parameter velocity of the point along the curve.

## Joint Primitives Not Counted As Degrees of Freedom

- If the joint primitive is motion actuated with a Joint Actuator block, that joint primitive is not counted in the machine state components.
- The weld primitive contributes no state components.

## Input Arguments

There is one input argument. The argument must be a SimMechanics block's full pathname or handle, or an indirect call to the pathname or handle using the commands `gcb` or `gcbh`.

The state manager obtained from the command references the machine to which this block belongs. The full path name starts with the model name and continues through any subsystem hierarchy:

```
pathname = modelname/subsystem1/etc.../blockname
```

You can obtain the pathname or handle of any block by selecting that block in a window and entering `gcb` or `gcbh` at the command line.

You can combine these steps into one step with an indirect pathname or handle call. Select a SimMechanics block in the model window and enter either command:

```
mec_h_stateVectorMgr(gcb)  
mec_h_stateVectorMgr(gcbh)
```

instead.

## Output Arguments

The output of `mec_h_stateVectorMgr` is a `MachineState` object of the `MECH.StateVectorMgr` class.

A machine is a connected set of SimMechanics blocks. Each machine must have at least one Ground block. Simulink chooses one of the Ground blocks as the *machine root*. This root serves as a proxy for the whole machine.

The `MachineState` object has four properties.

# mech\_stateVectorMgr

Property	Variable Type	Content
MachineState.MachineName	string	'modelName/subsystem1/etc .../rootgroundblock'
MachineState.X	1-by- $N$ real array	[ 0 0 ... 0 ]
MachineState.BlockStates	array of $N$ block state managers	Joint primitive and Point-Curve Constraint states
MachineState.StateNames	cell array of $N$ strings	Names of joint primitives and Point-Curve Constraints

Entering the `mech_stateVectorMgr` command or querying the entire object returns a summary of the object contents by property.

- The `MachineState.X` property does not show the actual machine state, but only indicates the number of state components.
- The block state managers of `BlockStates` are structures, arranged in the array by class: `MECH.RPJointStateMgr`, `MECH.SJointStateMgr`, and `MECH.PointCurveStateMgr`. The components of each manager contain:
  - The joint block and prismatic, revolute, and spherical primitive names
  - The position and velocity values and units
  - The `FixedAtT_0` flag indicating if the primitive is actuated with initial conditions.

## Querying the State Manager Object

Once you define a `MachineState` object, you can query the properties individually by entering full property name:

```
MachineState.MachineName  
MachineState.X
```

## Examples

Some examples illustrate the use of the state vector manager.

### Example: One Primitive

Open the model `mech_spen` in the Demos library. Select one of the `SimMechanics` blocks and enter

```
machinestate = mech_stateVectorMgr(gcb)
```

at the command line. The command returns

```
machinestate =  
MECH.StateVectorMgr  
    MachineName: 'mech_spen/Ground'  
             X: [0 0]  
    BlockStates: [1x1 MECH.RPJointStateMgr]  
    StateNames: {2x1 cell}
```

The first line in the object is the class and the last four are the properties. The model `mech_spen` contains one `Joint` block (a `Revolute`), with two states (angle and angular velocity).

Query individual properties. Entering `machinestate.machinename` returns

```
mech_spen/Ground
```

referring to the one `Ground` block in the model. Entering `machinestate.X` returns

```
0    0
```

indicating a two-component state vector ( $N = 2$ , position and velocity).

Entering `machinestate.blockstates` returns

```
MECH.RPJointStateMgr  
    BlockName: 'Revolute'  
    Primitive: 'R1'  
    Position: 0  
    PositionUnits: 'rad'  
    Velocity: 0  
    VelocityUnits: 'rad/s'  
    FixedAtT_0: 'off'
```

There are one `Joint` and no `Point-Curve Constraints`, and the `Joint` is a `Revolute`. So there is only one state manager of class `MECH.RPJointStateMgr`. This property gives detailed `Joint` information: block name, primitive name,

# **mech\_stateVectorMgr**

---

position and velocity values and units, and the absence of initial condition actuators.

Entering `machinestate.statenames` returns

```
'Revolute:R1:Position'  
'Revolute:R1:Velocity'
```

the names of the Joint block, the joint primitive, and the states.

## **Example: Multiple Primitives**

Construct an unnamed model with Ground and Body blocks connected by a Telescoping Joint. Then select one of the blocks and enter `machinestate = mech_stateVectorMgr(gcf)` at the command line. Simulink returns

```
machinestate =  
MECH.StateVectorMgr  
  MachineName: 'untitled/Ground'  
           X: [0 0 0 0 0 0 0 0 0 0]  
  BlockStates: [2x1 MECH.BlockStateMgr]  
  StateNames: {10x1 cell}
```

The unnamed model is still associated with its Ground block. There are two primitives, a spherical and a prismatic, and hence 10 components in the state vector. To see those primitive names, enter `machinestate.statenames` to obtain

```
'Telescoping:S:Quaternion:1'  
'Telescoping:S:Quaternion:2'  
'Telescoping:S:Quaternion:3'  
'Telescoping:S:Quaternion:4'  
'Telescoping:P1:Position'  
'Telescoping:S:Quaternion_dot:1'  
'Telescoping:S:Quaternion_dot:2'  
'Telescoping:S:Quaternion_dot:3'  
'Telescoping:S:Quaternion_dot:4'  
'Telescoping:P1:Velocity'
```

The quaternion and the prismatic position make the first five components, while the quaternion derivative and prismatic velocity make the last five.

**See Also**

Point-Curve Constraint, Prismatic, Revolute, Spherical

See “Counting Degrees of Freedom” on page 4-67.

See “Trimming SimMechanics Models” on page 7-12 and “Linearizing SimMechanics Models” on page 7-23.

In Simulink, see gcb, gcbh, gcs.





# Glossary

---

**actuator**

An *actuator* converts a Simulink signal into SimMechanics force/torque or motion signals.

- You can configure a *body actuator* to apply forces/torques to a body either as an explicit function of time or through feedback forces/torques.
- You can configure a *joint actuator* to apply forces/torques between the bodies connected on either side of the joint.
- You can configure a *driver actuator* to apply relative motion between the bodies connected on either side of the driver.
- SimMechanics also has two specialized actuators, one for setting joint initial conditions and one for applying stiction to a joint.

In SimMechanics, an Actuator block has an open round SimMechanics connector port  $\circ$  for connecting with a Body, Joint, or Driver block and an angle bracket  $>$  Simulink inport for connecting with normal Simulink blocks, such as Source blocks for generating force/torque signals.

See also *body*, *connector port*, *driver*, *initial condition actuator*, *joint*, *primitive joint*, *sensor*, and *stiction actuator*.

**adjoining CS**

The *adjoining CS* of a Body CS is the CS on the neighboring body or ground directly connected to the original Body CS by a Joint, Constraint, or Driver.

See also *body*, *Body CS*, *coordinate system (CS)*, *grounded CS*, and *World*.

**assembled joint**

Restricts the Body coordinate systems (CSs) on the two bodies at either end of the joint.

- For an *assembled prismatic joint*, the two Body CS origins must lie along the prismatic axis. The two Bodies translate relatively along the same axis.  
For an assembled joint with multiple prismatic primitives, the two Body CS origins must lie in the plane or space defined by the directions of the prismatic axes.
- For an *assembled revolute joint*, the two Body CS origins must be collocated. The two Bodies rotate relatively about the same axis.  
For an assembled joint with multiple revolute primitives, the two Body CS origins must be collocated.
- For an *assembled spherical joint*, the two Body CS origins must be collocated at the spherical primitive's pivot point. The two Bodies pivot relatively about this common origin.

You specify an *assembly tolerance* for assembled joints, the maximum dislocation distance allowed between all pairs of assembled Body CS origins and the maximum angle of misalignment between all pairs of assembled Body motion axes. If the distance dislocations and/or axis misalignments in an assembled joint grow larger than the assembly tolerance, the simulation stops with an error.

See also *assembly tolerance*, *Body CS*, *collocation*, *disassembled joint*, *joint*, and *primitive joint*.

### **assembly tolerance**

Determines how closely an assembled joint must be collocated and aligned. An assembled joint is connected on either side to Body coordinate systems (CSs) on two Bodies and restricts the relative configurations and motions of those Body CSs.

The assembly tolerances set the *maximum dislocation* of Body CS origins and *maximum misalignment* of motion axes allowed in assembled joints during the simulation.

- For *assembled prismatic primitives*, each pair of Body CS origins must lie in the subspace defined by the prismatic axis(es). Each pair of Bodies translates along this (these) common axis(es).
- For *assembled revolute primitives*, each pair of Body CS origins must be collocated and their respective rotational axes aligned. Each pair of Bodies rotates about this (these) common axis(es).
- For an *assembled spherical primitive*, the pair of Body CS origins must be collocated. The two Bodies pivot about this common origin.

If the two Body CSs separate or the joint axes misalign in a way that makes their connecting assembled joint primitives no longer respect the assembly tolerances, the simulation stops with an error.

See also *assembled joint*, *Body CS*, *collocation*, *disassembled joint*, and *joint*.

### **axis-angle rotation**

A representation of a three-dimensional spherical rotation as a rotation axis vector  $\mathbf{n} = (n_x, n_y, n_z)$  of unit length ( $\mathbf{n} \cdot \mathbf{n} = n_x^2 + n_y^2 + n_z^2 = 1$ ) and a rotation angle  $\theta$ . Define the rotation axis by the vector  $\mathbf{n}$ ; rotate about that axis by  $\theta$  using the right-hand rule.

The rotation axis direction is equivalent to specifying two independent angles;  $\theta$  is the third independent angle making up the rotation.

In VRML, you represent body rotations by a vector signal  $[n_x \ n_y \ n_z \ \theta]$ .

See also *degree of freedom (DoF)*, *Euler angles*, *primitive joint*, *quaternion*, *right-hand rule*, *rotation matrix*, and *VRML*.

**base (base body)** The point from which the joint is directed. The joint directionality runs from base to follower body.



Joint directionality sets the direction and the positive sign of all joint position/angle, motion, and force/torque data.

See also *body*, *directionality*, *follower (follower body)*, and *right-hand rule*.

**body** The basic element of a mechanical system or machine. It is characterized by


- Its mass properties (mass and inertia tensor)
- Its position and orientation in space
- Any attached Body coordinate systems

Bodies are connected to one another by joints. Bodies carry no degrees of freedom.

You can attach to a Body block any number of Body coordinate systems (CSs). All SimMechanics Bodies automatically maintain a minimum of one Body CS at the body's center of gravity (CG). The Body block has special axis triad CS ports , instead of the open, round connector ports , to indicate the attached Body CSs.

See also *actuator*, *adjoining CS*, *Body CS*, *center of gravity (CG)*, *convex hull*, *coordinate system (CS)*, *degree of freedom (DoF)*, *equivalent ellipsoid*, *inertia tensor*, *joint*, *local CS*, *mass*, and *sensor*.

**Body CS** A local coordinate system (CS) attached to a body, carried along with that body's motion. In general, bodies accelerate as they move, and therefore Body CSs define noninertial reference frames.

You can attach any number of Body CSs to a Body block, and you can choose where to place the Body CS origins and how to orient the Body CS axes. The Body block has special axis triad CS ports , instead of the open, round connector ports, to give you access to these Body CSs for connecting Joint, Sensor, and Actuator blocks.

Every Body block has an automatic, minimum Body CS at its center of gravity (CG). By default, it also has two other Body CSs for connection to adjacent Joints. The origin and axis orientation of each Body CSs once set by the user

during Body configuration, are interpreted as fixed rigidly in that body during the simulation.

See also *body*, *center of gravity (CG)*, *convex hull*, *coordinate system (CS)*, *ground*, *grounded CS*, *local CS*, *reference frame (RF)*, and *World*.

**center of gravity (CG)** The *center of gravity* or center of mass of an extended body is the point in space about which the entire body balances in a uniform gravitational field. For translational dynamics, the body's entire mass can be considered as if concentrated at this point.

Every Body block has an automatic, minimum Body coordinate system (CS) with its origin at the CG – the CG CS. This origin point and the Body CS coordinate axes remain fixed rigidly in the body during the simulation.

See also *body*, *Body CS*, *degree of freedom (DoF)*, *inertia tensor*, *kinematics*, and *primitive joint*.

**CG** See *center of gravity (CG)*.

**closed loop system** You can disconnect a *closed loop system* into two separate systems only by cutting more than one joint. The number of closed loops is equal to the minimum number, minus one, of cuttings needed to disconnect the system into two systems.

See also *open system* and *topology*.

**collocation** Two points in space are *collocated* if they are coincident, within assembly tolerances.

See also *assembled joint*, *assembly tolerance*, and *disassembled joint*.

**composite joint** A joint compounded from more than one joint primitive and thus representing more than one degree of freedom. The joint primitives constituting a composite joint are the *primitives* of that joint.

A spherical primitive represents three rotational degrees of freedom, but is treated as a primitive.

See also *constrained joint*, *degree of freedom (DoF)*, *joint*, and *primitive joint*.

**connection line** You connect each SimMechanics block to another by using *SimMechanics connection lines*. These lines function only with SimMechanics blocks. They do not carry signals, unlike normal Simulink lines, and cannot be branched. You cannot link connection lines directly to Simulink lines.

Connection lines appear red and dashed if they are not anchored at both ends to a connector port ○. Once you so anchor them, the lines become black and solid.

See also *actuator*, *connector port*, and *sensor*.

**connector port**

A special anchor for a connection line. Each SimMechanics block has one or more open round SimMechanics connector ports ○ for connecting to other SimMechanics blocks. You must connect these round ports only to other SimMechanics round ports. When an open connector port ○ is attached to a connection line, the Port changes to solid •.

A special *Connection Port block* is provided in the Library Browser to create a round SimMechanics connector port for an entire subsystem on that subsystem's boundary.

See also *actuator*, *connection line*, and *sensor*.

**constrained joint**

A composite joint with one or more built-in constraints relating the joint's primitives.

An example is the Screw block, which has a prismatic and a revolute primitive with their motions in fixed ratio. Only one of these degrees of freedom is independent.

See also *degree of freedom (DoF)*, *joint*, and *primitive joint*.

**constraint**

A restriction among degrees of freedom imposed *independently* of any applied forces/torques. A constraint removes one or more independent degrees of freedom, unless that constraint is *redundant* and restricts degrees of freedom that otherwise could not move anyway. Constraints can also create *inconsistencies* with the applied forces/torques that lead to simulation errors.

Constraints are kinematic: they must involve only coordinates and/or velocities. Higher derivatives of coordinates (accelerations, etc.) are determined by the Newtonian force/torque equations and cannot be independently constrained.

Constraints can be *holonomic* (integrable into a form involving only coordinates) or *nonholonomic* (not integrable; that is, irreducibly involving velocities).

The relationship specified by a constraint can be an explicit function of time (*rheonomic*) or not (*scleronomic*). In SimMechanics, scleronomic constraints are called *Constraints*, and rheonomic constraints are called *Drivers*.

SimMechanics Constraint/Driver blocks are attached to pairs of Body blocks.

See also *body*, *degree of freedom (DoF)*, *directionality*, and *driver*.

### **convex hull**

The surface of minimum area with convex (outward-bowing) curvature that passes through all the spatial points in a set. In three dimensions, this set must contain at least four non-coplanar points to make a closed surface with nonzero enclosed volume.

In SimMechanics, the convex hull is an option for visualizing a body. The set of points is all the Body coordinate system (CS) origins configured in that Body block. The visualization of an entire machine is the set of the convex hulls of all its bodies.

If a Body has fewer than four non-coplanar Body CSs, its convex hull is a lower-dimensional figure: three Body CSs produce a triangle without volume; two Body CSs produce a line without area; and one Body CS (the CS at the center of gravity) a point without length.

See also *body*, *Body CS*, and *equivalent ellipsoid*.

### **coordinate system (CS)**

A *coordinate system* is defined, in a particular reference frame (RF), by a choice of *origin* and orientation of *coordinate axes*, assumed orthogonal and Cartesian (rectangular). An observer attached to that CS measures distances from that origin and directions relative to those axes.

SimMechanics has two CS types:

- World: *global* or *absolute inertial CS* at rest
- Local CS:
  - Grounded CS
  - Body CS (includes center of gravity or CG CS)

See also *body*, *Body CS*, *center of gravity (CG)*, *convex hull*, *grounded CS*, *local CS*, *reference frame (RF)*, and *World*.

### **CS**

A *coordinate system (CS)*.

### **degree of freedom (DoF)**

A single coordinate of relative motion between two bodies. Such a coordinate is free only if it can respond without constraint or imposed motion to externally applied forces or torques. For translational motion, a DoF is a linear coordinate along a single direction. For rotational motion, a DoF is an angular coordinate about a single, fixed axis.

A prismatic joint primitive represents a single translational DoF. A revolute joint primitive represents a single rotational DoF. A spherical joint primitive represents three rotational DoFs in angle-axis form. A weld joint primitive represents zero DoFs.

See also *body*, *coordinate system (CS)*, *dynamics*, *joint*, and *kinematics*.

**directionality**

The *directionality* of a joint, constraint, or driver is its direction of forward motion.

The joint directionality is set by the order of the joint's connected bodies and the direction of the joint axis vector. One body is the *base* body, the other the *follower* body. The joint direction runs from base to follower, up to the sign of the joint axis vector. Reversing the base-follower order or the joint axis vector direction reverses the forward direction of the joint.

Joint directionality sets the direction and the positive sign of all joint position/angle, motion, and force/torque data.

Directionality of constraints and drivers is similar, except there is no joint axis, only the base-follower sequence.

See also *base (base body)*, *body*, *follower (follower body)*, *joint*, and *right-hand rule*.

**disassembled joint**

A *disassembled joint* need not respect the assembly tolerance of your machine.

- For a *disassembled prismatic primitive*, the Body coordinate system (CS) origins do not have to lie on the prismatic axis. The Bodies translate relatively along misaligned axes.
- For a *disassembled revolute primitive*, the Body CS origins do not have to be collocated. The Bodies rotate relatively about misaligned axes.
- The a *disassembled spherical primitive*, the Body CS origins do not have to be collocated. The Bodies pivot relatively about these two dislocated origins.

You can only use disassembled joints in a closed loop, with no more than one per loop.

See also *assembled joint*, *assembly tolerance*, *closed loop system*, *collocation*, and *topology*.

**DoF**

A *degree of freedom (DoF)*.



**driver**

A constraint that restricts degrees of freedom as an explicit function of time (a rheonomic constraint) and *independently* of any applied forces/torques. A driver removes one or more independent degrees of freedom, unless that driver is *inconsistent* with the applied forces/torques and leads to a simulation error.

In SimMechanics, you specify the driver function of time in a dialog box in terms of an input Simulink signal from a Driver Actuator.

SimMechanics Driver blocks are attached to pairs of Body blocks.

See also *actuator*, *body*, *constraint*, *directionality*, and *degree of freedom (DoF)*.

**dynamics**

A *forward dynamic* analysis of a mechanical system specifies

- The topology of how bodies are connected
- The degrees of freedom (DoFs) and constraints among DoFs
- All the forces/torques applied to the bodies
- The mass properties (masses and inertia tensors) of the bodies
- The initial condition of all DoFs:
  - Initial linear coordinates and velocities
  - Initial angular coordinates and velocities

The analysis then solves Newton's laws to find the system's motion for all later times.

*Inverse dynamics* is the same, except that the system's motion is specified and the forces/torques necessary to produce this motion are determined.

Dynamics is distinguished from *kinematics* by explicit specification of applied forces/torques and body mass properties.

See also *constraint*, *degree of freedom (DoF)*, *inertia tensor*, *kinematics*, *mass*, and *topology*.

**equivalent ellipsoid**

The *equivalent ellipsoid* of a body is the homogeneous solid ellipsoid, centered at the body's center of gravity, with the same principal moments of inertia and principal axes as the body. A homogeneous solid ellipsoid is the simplest body with three distinct principal moments.

Every body has a unique equivalent ellipsoid, but a given homogeneous ellipsoid corresponds to an infinite number of other, more complicated, bodies. The rotational dynamics of a body depend only on its equivalent ellipsoid

(which determines its principal moments and principal axes), not on its detailed shape.

In SimMechanics, the equivalent ellipsoid is an option for visualizing a body.

See also *body*, *convex hull*, *dynamics*, *inertia tensor*, *principal axes*, and *principal inertial moments*.

**Euler angles**

A representation of a three-dimensional spherical rotation as a product of three successive independent rotations about three independent axes by three independent (Euler) angles.

See also *axis-angle rotation*, *degree of freedom (DoF)*, *primitive joint*, *quaternion*, *right-hand rule*, and *rotation matrix*.

**follower  
(follower body)**

The point to which the joint is directed. The joint directionality runs from base to follower body.

Joint directionality sets the direction and the positive sign of all joint position/angle, motion, and force/torque data.

See also *base (base body)*, *body*, *directionality*, and *right-hand rule*.

**ground**

A *ground* or *ground point* is a special point fixed at rest in the absolute or global inertial World reference frame.

Each ground has an associated grounded coordinate system (CS). The grounded CS's origin is identical to the ground point, and its coordinate axes are always parallel to the coordinate axes of World.

See also *body*, *coordinate system (CS)*, *grounded CS*, and *World*.

**grounded CS**

A local CS attached to a ground point. It is at rest in World, but its origin is wherever the ground point is and therefore in general shifted with respect to the World CS origin. The coordinate axes of a grounded CS are always parallel to the World CS axes.

The World coordinate axes are defined so that:

+x points right

+y points up (gravity in -y direction)

+z points out of the screen, in three dimensions

You automatically create a Grounded CS whenever you set up a Ground block.

See also *adjoining CS*, *body*, *Body CS*, *coordinate system (CS)*, *ground*, *local CS*, and *World*.

**inertia tensor**

The *inertia* or *moment of inertia tensor* of an extended rigid body describes its internal mass distribution and the body's angular acceleration in response to an applied torque.

Let  $V$  be the body's volume and  $\rho(\mathbf{r})$  its mass density, a function of position  $\mathbf{r}$  within the body. Then the components of the inertia tensor  $\mathbf{I}$  are:

$$I_{ij} = \int_V dV [\delta_{ij} |\mathbf{r}|^2 - r_i r_j] \rho(\mathbf{r})$$

This tensor is a real, symmetric 3-by-3 matrix or equivalent MATLAB expression.

SimMechanics always assumes the inertia tensor of a body is evaluated in that body's center of gravity coordinate system (CG CS). That is, the origin is set to the body's CG and the coordinate axes are the CG CS axes.

Because the CG CS of a Body block is fixed rigidly in the body during simulation, the values of the inertia tensor components do not change as the body rotates.

See also *body*, *Body CS*, *equivalent ellipsoid*, *mass*, *principal axes*, and *principal inertial moments*.

**initial condition actuator**

An *initial condition actuator* gives you a way to move a system's degrees of freedom nondynamically to prepare a system for dynamical integration, in a way consistent with all constraints.

In SimMechanics, the initial conditions are applied to a joint primitive.

See also *actuator*, *dynamics*, and *kinematics*.

**joint**

Represents one or more mechanical degrees of freedom between two bodies. Joint blocks connect two Body blocks in a SimMechanics schematic. Joints have no mass properties such as mass or an inertia tensor.

A *joint primitive* represents one translational or rotational degree of freedom or one spherical (three rotational degrees of freedom in angle-axis form). Prismatic and revolute primitives have motion axis vectors. A weld primitive has no degrees of freedom.

A *primitive joint* contains one joint primitive. A *composite joint* contains more than one joint primitive.

Joints have a *directionality* set by their base-to-follower Body order and the direction of the joint primitive axis. The sign of all position/angle, motion, and force/torque data is determined by this directionality.

See also *actuator*, *assembled joint*, *base (base body)*, *body*, *composite joint*, *constrained joint*, *constraint*, *degree of freedom (DoF)*, *directionality*, *disassembled joint*, *follower (follower body)*, *ground*, *inertia tensor*, *massless connector*, *primitive joint*, and *sensor*.

## **kinematics**

A *kinematic* analysis of a mechanical system specifies topology, degrees of freedom (DoFs), motions, and constraints, without specification of applied forces/torques or the mass properties of the bodies.

The *machine state* at some time is the set of all

- Instantaneous positions
- Instantaneous velocities

of all bodies in the system, for both linear (translational) and angular (rotational) DoFs of the bodies.

Specification of applied forces/torques and solution of the system's motion as a function of time are given by the system's *dynamics*.

See also *constraint*, *degree of freedom (DoF)*, *dynamics*, and *topology*.



## **local CS**

A *local coordinate system (CS)* is attached to either a Ground or a Body:

- Grounded CS
- Body CS

You define Body CSs when you configure the properties of a Body. A Grounded CS is automatically defined when you represent a ground point by a Ground block.

A *grounded CS* is always at rest in the World reference frame. The origin of this Grounded CS is the same point as the ground point and therefore in general not the same as the World CS origin.

A *Body CS* is fixed rigidly in the body and carried along with that body's motion. To indicate an attached coordinate system, a Body block has a special axis triad CS port  in place of the open, round connector port .

See also *body*, *Body CS*, *coordinate system (CS)*, *grounded CS*, *reference frame (RF)*, and *World*.

**machine  
precision  
constraint**

A *machine precision constraint* is a constraint numerically implemented on the constrained degrees of freedom to the precision of your computer processor's arithmetic.

The precision to which the constraint is maintained depends on scale or the physical system of units.

See also *constraint*, *stabilizing constraint*, and *tolerancing constraint*.

**mass**

The proportionality between a force on a body and the resulting translational acceleration of that body.

Let  $V$  be the body's volume and  $\rho(\mathbf{r})$  its mass density, a function of position  $\mathbf{r}$  within the body. Then the mass  $m$  is:

$$m = \int_V dV \rho(\mathbf{r})$$

The mass is a real, positive scalar or equivalent MATLAB expression.

A body's mass is insensitive to choice of reference frame, coordinate system origin, or coordinate axes orientation.

See also *body* and *inertia tensor*.

**massless  
connector**

A *massless connector* is equivalent to two joints whose respective axes are spatially separated by a fixed distance. You can specify the gap distance and the axis of separation. The space between the degrees of freedom is filled by a rigid connector of zero mass.

You cannot actuate or sense a massless connector.

See also *disassembled joint* and *joint*.

**open system**

You can disconnect an *open system* into two separate systems by cutting no more than one joint.

Such systems can be divided into two types:

- An *open chain* is a series of bodies connected by joints and topologically equivalent to a line.

- An *open tree* is a series of bodies connected by joints in which at least one body has more than two joints connected to it. Bodies with more than two connected joints define *branch points* in the tree. A tree can be disconnected into multiple chains by cutting the branch points.

The end body of a chain is a body with only one connected joint.

See also *closed loop system* and *topology*.

**physical tree**

You obtain the *physical tree* representation of a machine topology from the full machine topology by removing actuators and sensors and cutting each closed loop once. The physical tree retains bodies, joints, constraints, and drivers.

See also *closed loop system*, *open system*, *spanning tree*, and *topology*.

**primitive joint**

A *primitive joint* expresses one degree of freedom (DoF) or coordinate of motion, if this DoF is a translation along one direction (*prismatic joint*) or a rotation about one fixed axis (*revolute joint*).

In SimMechanics, a *spherical joint* (three DoFs: two rotations to specify directional axis, one rotation about that axis) is also treated as a primitive joint.

These three types of primitive joints are the joint *primitives* from which composite joints are built up.

A weld primitive has no degrees of freedom.

See also *composite joint* and *joint*.

**principal axes**

The inertia tensor of a body is real and symmetric and therefore can be diagonalized, with three real eigenvalues and three orthogonal eigenvectors. The *principal axes* of a body are these eigenvectors.

See also *equivalent ellipsoid*, *inertia tensor*, and *principal inertial moments*.

**principal inertial moments**

The inertia tensor of a body is real, symmetric, and diagonalizable, with three real eigenvalues and three orthogonal eigenvectors. The *principal inertial moments* or *principal moments of inertia* of a body are these eigenvalues, the diagonal values when the tensor is diagonalized.

The principal moments of a real body satisfy the *triangle inequalities*: the sum of any two moments is greater than or equal to the third moment.

If two of the three principal moments are equal, the body has some symmetry and is dynamically equivalent to a *symmetric top*. If all three principal moments are equal, the body is dynamically equivalent to a *sphere*.

See also *equivalent ellipsoid*, *inertia tensor*, and *principal axes*.

**quaternion**

A *quaternion* represents a three-dimensional spherical rotation as a four-component row vector of unit length:

$$q = [n_x \sin(\theta/2) \quad n_y \sin(\theta/2) \quad n_z \sin(\theta/2) \quad \cos(\theta/2)],$$

with  $q^*q = 1$ . The vector  $\mathbf{n} = (n_x, n_y, n_z)$  is a three-component vector of unit length:  $\mathbf{n}^*\mathbf{n} = 1$ . The unit vector  $\mathbf{n}$  specifies the axis of rotation. The rotation angle about that axis is  $\theta$  and follows the right-hand rule.

The axis-angle representation of the rotation is just  $[\mathbf{n} \theta]$ .

See also *axis-angle rotation*, *degree of freedom (DoF)*, *Euler angles*, *primitive joint*, *right-hand rule*, and *rotation matrix*.

**reference frame (RF)**

The state of motion of an observer.

An *inertial RF* is a member of a set of all RFs moving uniformly with respect to one another, without relative acceleration.

An RF is necessary but not sufficient to define a coordinate system (CS). A CS which requires an origin point and a oriented set of three orthogonal axes.

See also *coordinate system (CS)*, *local CS*, and *World*.

**RF**

A *reference frame (RF)*.

**right-hand rule**

The *right-hand rule* is the standard convention for determining the sign of a rotation: point your right thumb into the positive rotation axis and curl your fingers into the forward rotational direction.

See also *degree of freedom (DoF)*, *directionality*, and *joint*.

**rotation matrix**

A representation of a three-dimensional spherical rotation as a 3-by-3 real, orthogonal matrix  $R$ :  $R^T R = R R^T = I$ , where  $I$  is the 3-by-3 identity and  $R^T$  is the transpose of  $R$ .

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}$$

In general,  $R$  requires three independent angles to specify the rotation fully. There are many ways to represent the three independent angles. Here are two:

- You can form three independent rotation matrices  $R_1, R_2, R_3$ , each representing a single independent rotation. Then compose the full rotation matrix  $R$  with respect to fixed coordinate axes (like World) as a product of these three:  $R = R_3 * R_2 * R_1$ . The three angles are *Euler angles*.
- You can represent  $R$  in terms of an *axis-angle rotation*  $\mathbf{n} = (n_x, n_y, n_z)$  and  $\theta$ , with  $\mathbf{n} * \mathbf{n} = 1$ . The three independent angles are  $\theta$  and the two needed to orient  $\mathbf{n}$ . Form the antisymmetric matrix  $\hat{\omega}$ :

$$\hat{\omega} = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}$$

Then *Rodrigues' formula* simplifies  $R$ :

$$R = \exp(\hat{\omega}\theta) = \hat{1} + \hat{\omega}\sin\theta + \hat{\omega}^2(1 - \cos\theta)$$

See also *axis-angle rotation, degree of freedom (DoF), Euler angles, primitive joint, quaternion, and right-hand rule*.

**schematic diagram**

A connected group of SimMechanics blocks. An entire block diagram in a Simulink model window has one or more schematics, each representing a distinct machine.

**sensor**

Measures the motion of, or forces/torques acting on, a body or joint. A sensor can also measure the reaction forces in a constraint or driver constraining a pair of bodies.

In SimMechanics, a Sensor block has an open round SimMechanics connector port  $\circ$  for connecting with a Body or Joint block and an angle bracket  $>$  Simulink output port for connecting with normal Simulink blocks, such as a Sinks block like Scope.

See also *actuator, body, connector port, constraint, driver, joint, and primitive joint*.



- spanning tree** You obtain the spanning tree representation of a machine topology from the full machine topology by removing everything except bodies and joints and cutting each closed loop once.
- See also *closed loop system*, *open system*, *physical tree*, and *topology*.
- stabilizing constraint** Numerically implements a constraint by modifying the dynamics of a system so that the constraint manifold is attractive, without changing the constrained solution. This constraint solver type is computationally the most efficient.
- The precision to which the constraint is maintained depends on scale or the physical system of units.
- See also *constraint*, *machine precision constraint*, and *tolerancing constraint*.
- stiction actuator** Applies discontinuous friction forces to a joint primitive according to the relative velocity of one body with the other body.
- If this relative velocity drops below a specified threshold, the relative motion ceases and the bodies or joints become locked rigidly to one another by static friction.
- Above that threshold, the bodies or joints move relative to one another with kinetic friction.
- See also *actuator*, *composite joint*, *dynamics*, *joint*, and *primitive joint*.
- tolerancing constraint** A *tolerancing constraint* is numerically implemented on constrained degrees of freedom only up to a specified accuracy and/or precision.
- This accuracy/precision is independent of any accuracy/precision limits on the solver used to integrate the system's motion, although constraints cannot be maintained to greater accuracy than the accuracy of the solver.
- The precision to which the constraint is maintained depends on scale or the physical system of units.
- Tolerancing constraints are useful in realistic simulation of slippage (“slop” or “play”) in constraints.
- See also *constraint*, *machine precision constraint*, and *stabilizing constraint*.
- topology** The global connectivity of the elements of a machine.
- For mechanical models, the *elements* are bodies and the *connections* are joints, constraints, and drivers. Two topologies are *equivalent* if you can transform one

system into another by continuous deformations and without cutting connections or joining elements.

An *open system* has no closed loops.

- An *open chain* is topologically equivalent to a line; and each body is connected to only two other bodies, if the body is internal, or one other body if it is at an end.
- An *open tree* has one or more *branch points*. A branch point is where an internal body is connected to more than two other bodies. A tree can be disconnected into multiple chains by cutting at the branch points.

A *closed loop system* has one or more closed loops. The number of closed loops is equal to the minimum number of joints, minus one, that must be cut to dissociate a system into two disconnected systems.

An actual system can have one of these primitive topologies or can be built up from multiple primitive topologies.

See also *body*, *closed loop system*, *joint*, and *open system*.

## **VRML**

*Virtual Reality Modeling Language*, an open, Web-oriented ISO standard for defining three-dimensional virtual worlds in multimedia and the Internet. The Virtual Reality Toolbox uses VRML to create and populate virtual worlds with user-defined bodies.

In VRML, body rotations are represented in the axis-angle form. The SimMechanics RotationMatrix2VR block converts rotation matrices to the equivalent axis-angle forms.

See also *axis-angle rotation* and the Web3D Consortium at [www.web3d.org](http://www.web3d.org).

## **World**

In SimMechanics, *World* is both the absolute inertial *reference frame (RF)* and absolute *coordinate system (CS)* in that RF. World has a fixed origin and fixed coordinate axes that cannot be changed.

The World coordinate axes are defined so that:

- +x points right
- +y points up (gravity in -y direction)
- +z points out of the screen, in three dimensions

See also *adjoining CS*, *coordinate system (CS)*, *ground*, *grounded CS*, and *reference frame (RF)*.

## Selected Bibliography

---

- [1] Gill, P. E., W. Murray, and M. Wright. *Practical Optimization*. San Diego: Academic Press, 1981.
- [2] Goldstein, H. *Classical Mechanics*, Second Edition. Reading, Massachusetts: Addison-Wesley, 1980.
- [3] Goodman, L. E., and W. H. Warner. *Statics*. Mineola, New York: Dover Publications, 2001 (original edition, 1964).
- [4] Goodman, L. E., and W. H. Warner. *Dynamics*. Mineola, New York: Dover Publications, 2001 (original edition, 1963).
- [5] Haug, E. J. *Computer-Aided Kinematics and Dynamics of Mechanical Systems, Volume 1: Basic Methods*. Boston: Allyn & Bacon, 1989.
- [6] José, J. V., and E. J. Saletan. *Classical Dynamics: A Contemporary Approach*. Cambridge: Cambridge University Press, 1998.
- [7] JPL DARTS Web page on spatial operator algebra:  
<http://dsshell.jpl.nasa.gov/references.html>.
- [8] Marrin, C., and B. Campbell. *Teach Yourself VRML 2 in 21 Days*. Indianapolis, Indiana: Sams.net, 1997.
- [9] Marsden, J. E., and T. S. Ratiu. *Introduction to Mechanics and Symmetry*, Second Edition. New York: Springer-Verlag, 1999.
- [10] Meriam, J. L. *Engineering Mechanics*, Fourth Edition, two volumes. New York: John Wiley and Sons, 1997.
- [11] Murray, R. M., Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, Florida: CRC Press, 1994.
- [12] von Schwerin, R. *MultiBody System SIMulation: Numerical Methods, Algorithms, and Software*. Berlin: Springer-Verlag, 1999.

## A

- actuators 4-40
  - initial condition 4-50
  - stiction 4-45
- analysis modes
  - choosing 5-4
  - overview 1-19
  - See also* Forward Dynamics mode, Inverse Dynamics mode, Kinematics mode, Trimming mode
- Angle Driver block 8-10
- assembling joints
  - See* joint
- assembly tolerances 4-23
  - setting 5-5
- axis-angle rotation 3-7

## B

- base body 4-21
- Bearing block 8-13
- block libraries 8-2
  - viewing 2-2
- bodies 4-8
  - actuating 4-40
  - body coordinate systems 4-10
  - modeling 4-9
  - rigid 4-9
  - sensing 4-60
- Bodies block library 2-4
- Body Actuator block 8-29
- Body block 8-18
- body coordinate systems
  - adding and deleting 4-15
- Body CS ports
  - See* ports
- Body Sensor block 8-32

- Body Spring & Damper block 8-36
- Bushing block 8-39

## C

- closed loops
  - choosing cut joint 4-33
  - constraint or driver block in 4-35
  - cutting 4-32
  - disassembled joint in 4-30
- code generation 5-22
- computed force 4-62
- computer-aided design (CAD)
  - See* virtual reality
- connection lines
  - See* lines
- Connection Port block 8-44
- connector ports
  - See* ports
- Constraint & Driver Sensor block 8-45
- constraint solvers
  - properties 5-5
- constraint tolerances
  - definition 5-6
- constraints 4-34
  - directionality 4-35
  - holonomic 4-34
  - modeling 4-34
  - nonholonomic 4-34
  - rheonomic 4-34
  - scleronomic 4-34
  - sensing 4-62
- Constraints & Drivers block library 2-5
- Continuous Angle block 8-48
- convex hulls 6-8
- coordinate systems 3-2

- transformations 3-3
- Custom Joint block 8-51
- cutting joints
  - automatic in closed loops 4-32
  - marking cut joint 5-10
- Cylindrical block 8-58

## D

- damper
  - See* spring-damper
- degrees of freedom 4-17
  - apparent vs. independent 4-67
  - counting 4-67
  - relative 4-17
  - rotational 4-18
  - spherical 4-18
  - translational 4-18
  - weld 4-18
- demo models 2-6
  - opening xiii
- Disassembled Cylindrical block 8-62
- disassembled joints
  - See* joints
- Disassembled Prismatic block 8-67
- Disassembled Revolute block 8-71
- Disassembled Spherical block 8-75
- Distance Driver block 8-78
- Driver Actuator block 8-81
- drivers 4-34
  - actuating 4-49
  - directionality 4-35
  - modeling 4-34
  - sensing 4-62
- dynamics 1-20

## E

- equivalent ellipsoids 6-5
- errors
  - See* simulation
  - fixing errors
- Euler angles 3-8

## F

- follower body 4-21
- force
  - computed force 4-62
  - reaction force 4-62
- Force Elements block library 2-6
- Forward Dynamics mode
  - overview 1-20
- friction
  - pure kinetic friction 4-41
  - See also* stiction

## G

- Gear Constraint block 8-84
- Gimbal block 8-88
- Ground block 8-93
- grounds
  - ground point 4-8
  - grounded coordinate system 4-9
  - modeling 4-8

## H

- Handle Graphics
  - controlling simulation 6-20
  - machine display 6-16
  - perspective and window size 6-19
  - recording animations 6-20

- saving display settings 6-22
- setting up visualization 6-10
- special SimMechanics symbols 6-12
- special SimMechanics tool bar 6-22
- standard Handle Graphics controls 6-12

## I

- inertia tensor 4-10
  - time-varying 4-43
- initial conditions
  - setting 4-50
- In-Plane block 8-95
- internal forces
  - represented by force elements 4-55
  - via sensor-actuator feedback 4-58
- Inverse Dynamics mode
  - finding forces from motion 7-2
  - overview 1-20
  - setting up motion actuation 7-2
  - simulating in 7-7

## J

- Joint Actuator block 8-100
- Joint Initial Condition Actuator block 8-105
- Joint Sensor block 8-109
- Joint Spring & Damper block 8-117
- Joint Stiction Actuator block 8-121
- joints 4-17
  - actuating 4-44
  - assembly restrictions 4-23
  - automatic assembly of disassembled 4-30
  - automatic cutting 4-32
  - directionality 4-21
  - disassembled joints 4-29
  - joint primitives 4-17

- manual assembly of 4-23
- massless connectors 4-26
- modeling 4-17
- primitive axis 4-20
- primitive vs. composite 4-18
- sensing 4-62

## Joints block library 2-5

- Disassembled Joints block sublibrary 2-5
- Massless Connectors block sublibrary 2-5

## K

- kinematics 1-20
  - machine state 3-9
  - representing motion 3-4
- Kinematics mode
  - finding forces from motion 7-2
  - overview 1-21
  - setting up motion actuation 7-2
  - simulating in 7-3

## L

- Linear Driver block 8-128
- linearization
  - linearizing SimMechanics models 7-23
- lines
  - connection lines 4-4
  - signal lines 4-4

## M

- machine
  - representing with blocks 4-2
  - SimMechanics model of 4-2
- mass 4-10
  - time-varying 4-43

massless connectors

*See* joints

Mechanical Branching Bar block 8-131

Mechanical Environment Settings 5-7

Constraints panel 5-9

Linearization panel 5-10

Parameters panel 5-7

Visualization panel 5-12

## N

Newton's laws 1-20

## O

ODE

*See* Simulink

## P

Parallel Constraint block 8-135

Planar block 8-138

Point-Curve Constraint block 8-143

ports

Body CS ports 4-4

connector ports 4-4

Simulink ports 4-4

Prismatic block 8-152

## Q

quaternion 3-9

## R

reaction force 4-62

reference frames 3-2

Revolute block 8-156

Revolute-Revolute block 8-160

Revolute-Spherical block 8-165

rotation matrix 3-7

RotationMatrix2VR block 8-170

rotations

representing 3-7

## S

schematic 4-2

spanning tree 4-64

Screw block 8-172

sensors 4-60

Sensors & Actuators block library 2-5

simulation

fixing errors 5-16

internal SimMechanics steps 5-14

Simulink

choosing ODE solver 5-2

setting ODE solver tolerances 5-3

singularities

handling 5-3

setting robust handling 5-10

Six-DoF block 8-177

spanning tree

*See* schematic

Spherical block 8-181

Spherical-Spherical block 8-186

spring-damper 4-55

stiction

classical Coulomb theory 8-124

mixed static-kinetic friction 4-48

modeling 4-45

*See also* friction 4-48

subsystem

in SimMechanics 4-5

masking 4-7



**T**

technical vocabulary

summary 3-11

Telescoping block 8-190

tolerances

*See* assembly tolerances

topology

invalid 4-66

model 4-64

validity of 4-65

Trimming mode

overview 1-21

simulating in 7-12

with constrained systems 7-18

with unconstrained systems 7-12

troubleshooting

*See* simulation

fixing errors

**U**

Universal block 8-194

Utilities block library 2-6

**V**

Variable Mass & Inertia Actuator block 8-199

Velocity Driver block 8-203

virtual reality

computer-aided design (CAD) 6-31

custom interface to SimMechanics 6-30

observer controls 6-26

setting up visualization 6-24

special symbols 6-25

Virtual Reality Toolbox

as related product 1-4

used in visualization 1-3

using with SimMechanics 6-24

visualization

body shape 6-5

customized virtual reality animation 6-30

Handle Graphics 6-10

overview 1-21

rendering vs. animation 6-3

virtual reality 6-24

**W**

Weld block 8-207

World coordinate system 3-3

