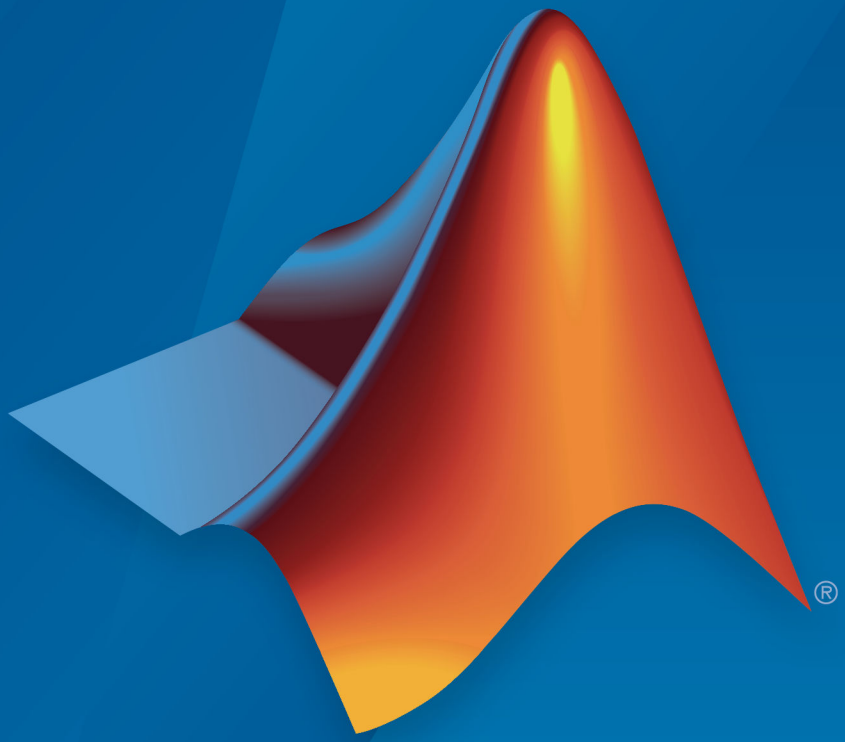
Simscape™ Multibody™

User's Guide



MATLAB® & SIMULINK®

R2017b

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Simscape™ Multibody™ User's Guide

© COPYRIGHT 2002–2017 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

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

Revision History

March 2012	Online only	New for Version 4.0 (Release R2012a)
September 2012	Online only	Revised for Version 4.1 (Release R2012b)
March 2013	Online only	Revised for Version 4.2 (Release R2013a)
September 2013	Online only	Revised for Version 4.3 (Release R2013b)
March 2014	Online only	Revised for Version 4.4 (Release R2014a)
October 2014	Online only	Revised for Version 4.5 (Release R2014b)
March 2015	Online only	Revised for Version 4.6 (Release R2015a)
September 2015	Online only	Revised for Version 4.7 (Release R2015b)
March 2016	Online only	Revised for Version 4.8 (Release R2016a) (Renamed from *SimMechanics™ User's Guide*)
September 2016	Online only	Revised for Version 4.9 (Release R2016b)
March 2017	Online only	Revised for Version 5.0 (Release R2017a)
September 2017	Online only	Revised for Version 5.1 (Release R2017b)

Contents

Multibody Modeling

1 Bodies

Bodies Workflow **1-2**
Bodies in the Context of a Model **1-2**
Step 1: Study the Bodies to Model **1-2**
Step 2: Model the Solids in Each Body **1-3**
Step 3: Connect the Solids Through Frames **1-3**
Step 4: Verify the Body Subsystems **1-4**

Modeling Bodies **1-5**
What Are Bodies? **1-5**
Body Elements **1-6**
Relevant Blocks **1-8**
Body Visualization **1-9**
See It: A Typical Body **1-10**
Boundaries of Bodies **1-13**
Bodies as Simulink Subsystems **1-16**

Compounding Body Elements **1-19**
Compounding as a Modeling Strategy **1-19**
Try It: Create a Compound Geometry **1-19**
Try It: Create a Compound Inertia **1-23**

Working with Frames **1-25**
What Are Frames? **1-25**
Role of Frames **1-25**
Custom Solid Frames **1-27**
What Are Frame Transforms? **1-28**
Visualizing Frame Transforms **1-29**
Try It: Specify a Frame Transform **1-30**

Creating Connection Frames **1-35**
Frames as a Connection Points **1-35**
Creating and Transforming Frames **1-35**
See It: Frames in a Typical Body **1-36**
Planning Connection Frames **1-38**
Addressing Assembly Errors **1-39**

Representing Solid Geometry **1-42**
Geometry in a Model **1-42**
Preset Solid Shapes **1-45**
Imported Solid Shapes **1-46**
Compound Solid Shapes **1-48**

Modeling Extrusions and Revolutions **1-50**
Extrusions and Revolutions **1-50**
The Cross-Section Profiles **1-52**
Cross-Sections with Holes **1-55**
From Cross-Sections to Solids **1-58**

Visualize a Model and Its Components **1-60**
Visualize a Complete Multibody Model **1-60**
Visualize an Individual Solid Geometry **1-62**
A Note on Imported Geometries **1-63**

Representing Solid Inertia **1-65**
Representing Inertias **1-65**
Compounding Solids and Inertias **1-70**

Specifying Custom Inertias **1-73**
Key Inertia Conventions **1-73**
Inertia Matrix Definitions **1-73**
CAD as an Inertia Data Source **1-78**
Automatic Inertia Calculation **1-82**

Specifying Variable Inertias **1-84**
Modeling Variable Inertias **1-84**
Visualizing Variable Inertias **1-85**
Modeling Body Interactions **1-85**
Model a Variable-Mass Oscillator **1-86**

Creating Custom Solid Frames **1-92**
Solid Frames **1-92**
Opening the Frame-Creation Interface **1-93**

Geometry-Based Frame Placement 1-93
Primary and Secondary Axes 1-94
Try It: Create a Custom Solid Frame 1-94

Manipulate the Color of a Solid 1-102
Visual Property Parameterizations 1-102
RGB and RGBA Vectors 1-103
Simple Visual Properties 1-104
Advanced Visual Properties 1-104
Adjust Solid Opacity 1-105
Adjust Highlight Color 1-106
Adjust Shadow Color 1-106
Adjust Self-Illumination Color 1-107

2 Multibody Systems

Multibody Assembly Workflow 2-2
Study the Joints and Constraints to Model 2-2
Assemble Bodies Using Joints and Constraints 2-2
Guide Model Assembly 2-3
Verify Model Assembly 2-3

Modeling Joint Connections 2-4
Role of Joints in a Model 2-4
Joint Degrees of Freedom 2-4
Joint Primitives 2-6
Joint Inertia 2-8

How Multibody Assembly Works 2-9
Model Assembly 2-9
Connecting Joints 2-9
Orienting Joints 2-10
Guiding Assembly 2-11
Verifying Model Assembly 2-12

Counting Degrees of Freedom 2-15

Model an Open-Loop Kinematic Chain 2-16
Model Overview 2-16

Build Model 2-17
Guide Model Assembly 2-18
Visualize Model and Check Assembly Status 2-18
Simulate Model 2-20
Open Reference Model 2-20

Model a Closed-Loop Kinematic Chain 2-21
Model Overview 2-21
Build Model 2-22
Specify Block Parameters 2-24
Guide Assembly and Visualize Model 2-24
Verify Model Assembly 2-25
Simulate Model 2-26
Open Reference Model 2-26

Troubleshoot an Assembly Error 2-27
Model Overview 2-27
Explore Model 2-28
Update Model 2-31
Troubleshoot Assembly Error 2-31
Correct Assembly Error 2-33
Simulate Model 2-34

Modeling Gear Constraints 2-36
Gear Constraints and Applications 2-36
Gear Assemblies as Kinematic Loops 2-37
Gear Assembly Restrictions 2-39
Gear Pitch Circles 2-40
Modeling Gear Geometries 2-41
Limitations of Gear Constraints 2-42

Assemble a Gear Model 2-44
Gear Examples 2-44
Bevel Gear 2-46
External Spur Gear 2-50
Internal Spur Gear 2-54
Rack and Pinion 2-57
Worm and Gear 2-61

Model a Compound Gear Train 2-66
Model Overview 2-66
Model Sun-Planet Gear Set 2-67
Constrain Sun-Planet Gear Motion 2-70

Add Ring Gear 2-72
Add Gear Carrier 2-75
Add More Planet Gears 2-80

Constrain a Point to a Curve 2-82
Open the Flap Assembly Model 2-82
Effect of Constraints on the Model 2-83
Create the Connection Frames 2-84
Connect the Constraint Blocks 2-86
Specify the Flap Constraint Curves 2-87
Simulate the Model 2-88

3 Internal Mechanics, Actuation and Sensing

Modeling and Sensing System Dynamics 3-2
Provide Joint Actuation Inputs 3-2
Specify Joint Internal Mechanics 3-2
Model Body Interactions and External Loads 3-3
Sense Dynamical Variables 3-3

Modeling Gravity 3-4
Gravity Models 3-4
Gravitational Force Magnitude 3-5
Force Position and Direction 3-6
Gravitational Torques 3-7

Model Gravity in a Planetary System 3-9
Model Overview 3-9
Step 1: Start a New Model 3-10
Step 2: Add the Solar System Bodies 3-11
Step 3: Add the Degrees of Freedom 3-15
Step 4: Add the Initial State Targets 3-17
Step 5: Add the Gravitational Fields 3-21
Step 6: Configure and Run the Simulation 3-23
Open an Example Model 3-24

Specifying Joint Actuation Inputs 3-25
Actuation Modes 3-25
Motion Input 3-28

Input Handling 3-30
Assembly and Simulation 3-30
Specifying Motion Input Derivatives 3-32

Joint Actuation Limitations 3-34
Closed Loop Restriction 3-34
Motion Actuation Not Available in Spherical Primitives 3-34
Redundant Actuation Mode Not Supported 3-34
Model Report and Mechanics Explorer Restrictions . . . 3-35
Motion-Controlled DOF Restriction 3-35

Actuating and Sensing with Physical Signals 3-36
Exposing Physical Signal Ports 3-36
Converting Actuation Inputs 3-36
Obtaining Sensing Signals 3-37

Sensing 3-39
Sensing Overview 3-39
Variables You Can Sense 3-40
Blocks with Sensing Capability 3-40
Sensing Output Format 3-40

Force and Torque Sensing 3-42
Blocks with Force and Torque Sensing 3-42
Joint Forces and Torques You can Sense 3-43
Force and Torque Measurement Direction 3-45

Motion Sensing 3-46
Sensing Spatial Relationships Between Joint Frames . . 3-46
Sensing Spatial Relationships Between Arbitrary Frames 3-48

Rotational Measurements 3-51
Rotation Sensing Overview 3-51
Measuring Rotation 3-51
Axis-Angle Measurements 3-51
Quaternion Measurements 3-52
Transform Measurements 3-54

Translational Measurements 3-56
Translation Sensing Overview 3-56
Measuring Translation 3-56

Cartesian Measurements 3-57
Cylindrical Measurements 3-59
Spherical Measurements 3-61

Measurement Frames 3-64
Measurement Frame Purpose 3-64
Measurement Frame Types 3-65

Sense Motion Using a Transform Sensor Block 3-67
Model Overview 3-67
Modeling Approach 3-68
Build Model 3-68
Guide Model Assembly 3-70
Simulate Model 3-70
Save Model 3-72

Specify Joint Actuation Torque 3-73
Model Overview 3-73
Four-Bar Linkages 3-74
Modeling Approach 3-76
Build Model 3-77
Simulate Model 3-80

Analyze Motion at Various Parameter Values 3-85
Model Overview 3-85
Build Model 3-85
Specify Block Parameters 3-87
Create Simulation Script 3-88
Run Simulation Script 3-89

Sense Forces and Torques Acting at Joints 3-91
Overview 3-91
Open Model 3-92
Sense Actuation Torque 3-92
Sense Constraint Forces 3-95
Sense Total Forces 3-97

Sense Constraint Forces 3-99
Model Overview 3-99
Add Weld Joint Block to Model 3-100
Add Constraint Force Sensing 3-101
Add Damping to Joints 3-102
Simulate Model 3-102

Plot Constraint Forces 3-103

Specify Joint Motion Profile 3-105
Model Overview 3-105
Build Model 3-105
Simulate Model 3-108

Prescribe Joint Motion in Planar Manipulator Model 3-110
Model Overview 3-110
Add Virtual Joint 3-111
Prescribe Motion Inputs 3-112
Sense Joint Actuation Torques 3-116
Simulate Model 3-117

Simulation and Visualization

4 Simulation

Update and Simulate a Model 4-2
Create or Open a Model 4-2
Update the Block Diagram 4-2
Examine the Model Assembly 4-3
Configure the Solver Settings 4-3
Run Simulation and Analyze Results 4-4

Multibody Simulation Issues 4-5

5 Visualization and Animation

Enable Mechanics Explorer 5-2

Working with Animation **5-3**
Animation Playback **5-3**
Looping Playback **5-3**
Changing Playback Speed **5-3**
Jumping to Playback Time **5-4**

Manipulate the Visualization Viewpoint **5-5**
Model Visualization **5-5**
Select a Standard View **5-5**
Set View Convention **5-6**
Rotate, Roll, Pan, and Zoom **5-7**
Split Model View **5-8**

Visualization Cameras **5-10**
Camera Types **5-10**
Global Camera **5-11**
Dynamic Cameras **5-12**
Camera Trajectory Modes **5-12**
Dynamic Camera Selection **5-13**
Dynamic Camera Reuse **5-14**

Create a Dynamic Camera **5-15**
Start a New Camera Definition **5-15**
Define a Keyframes Camera **5-16**
Define a Tracking Camera **5-17**
Select a Dynamic Camera **5-17**

Selective Model Visualization **5-19**
What Is Visualization Filtering? **5-19**
Changing Component Visibility **5-20**
Visualization Filtering Options **5-21**
Components You Can Filter **5-21**
Model Hierarchy and Tree Nodes **5-22**
Filtering Hierarchical Subsystems **5-22**
Updating Models with Hidden Nodes **5-23**
Alternative Ways to Enhance Visibility **5-24**

Selectively Show and Hide Model Components **5-25**
Visualization Filtering **5-25**
Open Example Model **5-26**
Update Example Model **5-26**
Hide Half-Cylinder Subsystem **5-27**
Show Solid in Hidden Subsystem **5-28**

Show Only Piston Subsystem 5-29
Show Everything 5-30

Visualize Simscape Multibody Frames 5-32
What Are Frames? 5-32
Show All Frames 5-32
Highlight Specific Frames 5-33
Visualize Frames via Graphical Markers 5-35

Go to a Block from Mechanics Explorer 5-37

Create a Model Animation Video 5-39
UI and Command-Line Tools 5-39
Before Creating a Video 5-39
Create a Video Using Video Creator 5-39
Create a Video Using smwritevideo 5-40

Multibody Model Import

6 CAD and URDF Model Import

CAD Translation 6-2
Translating a CAD Model 6-2
What's in a Translated Model? 6-3
What's in a Data File? 6-6
Exporting a CAD Model 6-7
Importing a CAD Model 6-8
Simplifying Model Topology 6-9
Updating an Existing Data File 6-10

Install the Simscape Multibody Link Plug-In 6-12
What is Simscape Multibody Link? 6-12
Alternatives to Simscape Multibody Link 6-12
Before You Begin 6-13
Step 1: Get the Installation Files 6-13
Step 2: Run the Installation Function 6-13
Step 3: Register MATLAB as an Automation Server . . . 6-14

Step 4: Enable the Simscape Multibody Link Plug-In . . 6-14

Import a CAD Assembly Model 6-15
Before You Begin 6-15
Example Files 6-15
Import a Model 6-15
After Model Import 6-16

Import a Robotic Arm CAD Model 6-17
Example Overview 6-17
Example Files 6-17
Import the Model 6-17
Visualize the Model 6-18
Build on the Model 6-20

Onshape Import 6-21
What Is Onshape? 6-21
What's in an Onshape Model? 6-21
Preparing a Model for Import 6-22
Importing an Onshape Model 6-22
What Can You Import? 6-24
User Authentication and Account Permissions 6-24
Mapping to Simscape Multibody Blocks 6-25
Onshape Import Warnings and Errors 6-25
Physical Units 6-26
Obtaining Onshape Models to Import 6-26

Import an Onshape Humanoid Model 6-27
Onshape Import 6-27
Example Overview 6-27
Export the Model 6-28
Import the Model 6-28

URDF Primer 6-31
What is URDF? 6-31
What's in a URDF File? 6-32
XML Hierarchies and Kinematic Trees 6-33
Required and Optional URDF Entities 6-36
Create a Simple URDF Model 6-37
Obtaining URDF Models to Import 6-40

URDF Import 6-41
Importing a URDF Model 6-41

URDF Entities That You Can Import 6-41
Mapping to Simscape Multibody Blocks 6-42
Import a Simple URDF Model 6-44
URDF Import Limitations 6-47
Mesh Geometries 6-48
Physical Units 6-48
Differences from CAD Import 6-48

Import a URDF Humanoid Model 6-50
URDF Import 6-50
Example Overview 6-50
Import the Model 6-51

Deployment

7 Code Generation

Code Generation Applications 7-2
Code Generation Overview 7-2
Simulation Acceleration 7-2
Model Deployment 7-3

Code Generation Setup 7-4
Before You Begin 7-4
Solver Selection 7-4
Target Selection 7-4
Run-Time Parameters 7-4
Compiler Optimization 7-5

Generate Code for a Multibody Model 7-7

Configure Model for Rapid Accelerator Mode 7-9
Model Overview 7-9

Multibody Modeling

Bodies

- "Bodies Workflow" on page 1-2
- "Modeling Bodies" on page 1-5
- "Compounding Body Elements" on page 1-19
- "Working with Frames" on page 1-25
- "Creating Connection Frames" on page 1-35
- "Representing Solid Geometry" on page 1-42
- "Modeling Extrusions and Revolutions" on page 1-50
- "Visualize a Model and Its Components" on page 1-60
- "Representing Solid Inertia" on page 1-65
- "Specifying Custom Inertias" on page 1-73
- "Specifying Variable Inertias" on page 1-84
- "Creating Custom Solid Frames" on page 1-92
- "Manipulate the Color of a Solid" on page 1-102

Bodies Workflow

In this section...
“Bodies in the Context of a Model” on page 1-2
“Step 1: Study the Bodies to Model” on page 1-2
“Step 2: Model the Solids in Each Body” on page 1-3
“Step 3: Connect the Solids Through Frames” on page 1-3
“Step 4: Verify the Body Subsystems” on page 1-4

Bodies in the Context of a Model

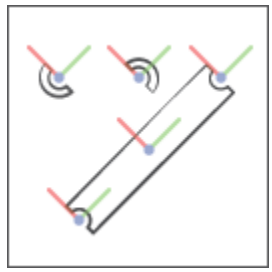
Bodies are the core constituents of a model. In order to assemble bodies into a mechanism or machine, you must first model those bodies. Here are the recommended steps to follow when performing this task.

Step 1: Study the Bodies to Model

Bodies are rigid collections of solids and occasionally other body elements. Start your model by conceptually breaking down each body into shapes that you can specify using the Solid block. Obtain the dimensions, inertia, and color of each solid. For more information on bodies and their nature as rigid collections of body elements, see:

- “Modeling Bodies” on page 1-5
- “Compounding Body Elements” on page 1-19

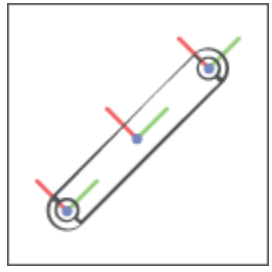
Step 2: Model the Solids in Each Body



Body elements are defined by their material attributes. Specify these attributes using Solid, Inertia, and other blocks from the Body Elements library. For more information on body elements and their attributes, see:

- “Representing Solid Geometry” on page 1-42
- “Representing Solid Inertia” on page 1-65
- “Manipulate the Color of a Solid” on page 1-102

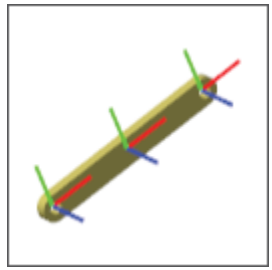
Step 3: Connect the Solids Through Frames



The placement of a body element in a body depends on how its frames are defined. Create and connect the frames required to assemble the various body elements into a complete body. For more information on frames and frame connections, see:

- “Working with Frames” on page 1-25
- “Creating Connection Frames” on page 1-35
- “Creating Custom Solid Frames” on page 1-92

Step 4: Verify the Body Subsystems



Visualize each body and verify its geometry, color, and frames. You can use the Solid block visualization pane to visualize an individual solid or Mechanics Explorer to visualize a complete body. For more information about visualization, see:

- “Visualize a Model and Its Components” on page 1-60
- “Selective Model Visualization” on page 5-19
- “Manipulate the Visualization Viewpoint” on page 5-5

Modeling Bodies

In this section...
“What Are Bodies?” on page 1-5
“Body Elements” on page 1-6
“Relevant Blocks” on page 1-8
“Body Visualization” on page 1-9
“See It: A Typical Body” on page 1-10
“Boundaries of Bodies” on page 1-13
“Bodies as Simulink Subsystems” on page 1-16

What Are Bodies?

Bodies are the basic constituents of a multibody model. They are the solid components that you connect when assembling a model and the subjects upon which all forces and torques in that model ultimately act. The planets and the sun serve as bodies in a model of the solar system, for example, and wings do too in a model of a flapping mechanism.

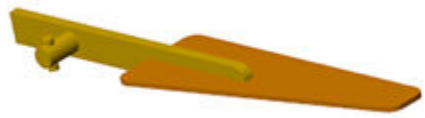
A Flapping Mechanism as an Assembly of Bodies

If you are familiar with CAD modeling, you can loosely think of bodies as equivalents of CAD parts, each a modular component with geometry and material. The modeling approach may differ—for example, there are no sketches to draw—but, conceptually, the end result is the same: something that you can connect, constrain, act upon, and visualize.

Body Elements

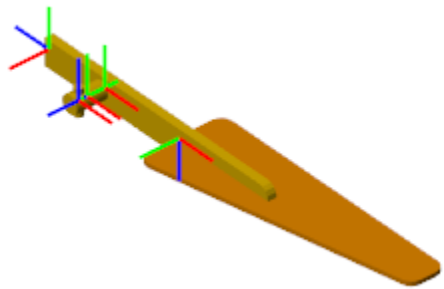
A body is a rigid collection of simpler body elements: solids with geometry and inertia and, less often, plain inertias (without assigned geometries) and plain geometries (without associated inertias). The body elements that you model and the ways in which you connect them determine the overall attributes of the body.

Bodies with simple shapes often require a single solid. Those with complex shapes or inertias may require several solids and the occasional plain inertia or plain geometry. Like bricks in a modular build set, the properties of the various body elements can differ from each other. It is up to you to combine them in a way that produces the desired body.



A Wing Body as a Rigid Collection of Solids

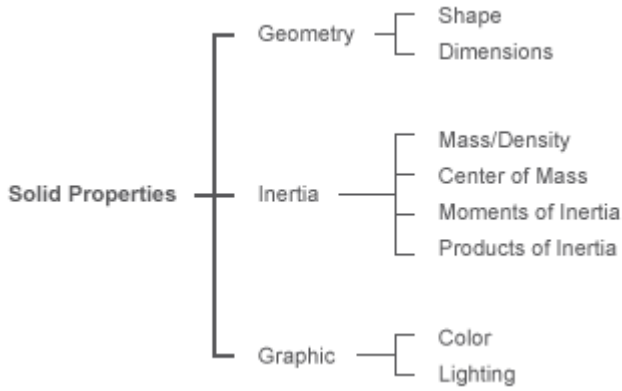
Each body element comprises one or more frames and a set of material attributes. The frames determine the placement of the body elements relative to each other and provide the attachment points for joints and constraints. The attributes factor in the dynamic behavior of the body elements and, in those with geometry, help determine their visual appearance.



Frames on a Wing Body

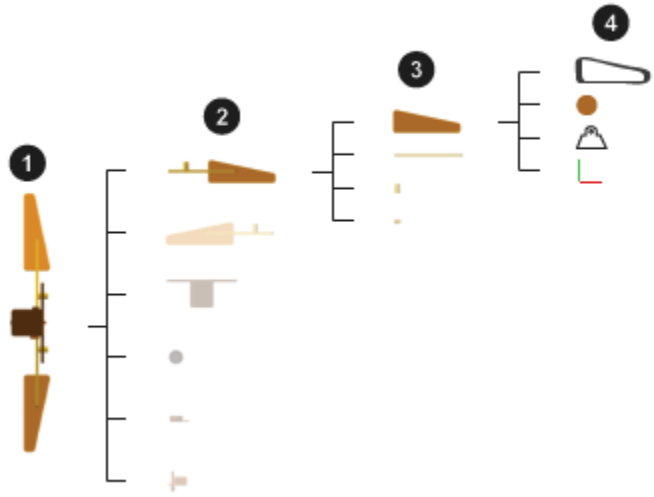
Material Attributes

The attributes of solids include geometry, inertia, and color. Geometry enables the automatic calculation of inertia parameters and, in conjunction with color, visualization. Inertia quantifies the resistance to changes in motion and factor into the calculation of the forces and torques required to induce an acceleration.



The attributes of other body elements are more limited in scope. Those of plain inertias include only inertia, with parameters that in the general case encompass mass, center of mass, moments of inertia, and products of inertia. The attributes of plain geometries include geometry, with parameters such as shape and dimensions, as well as color.

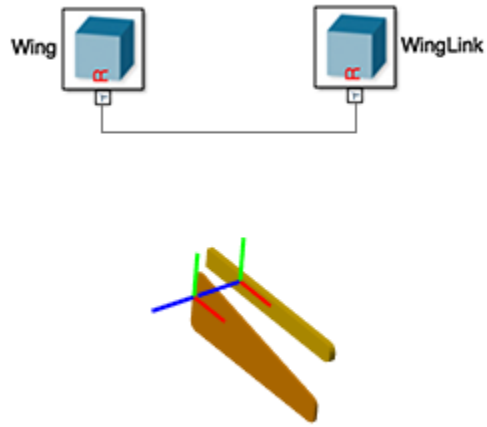
The figure summarizes the structure of a typical body (here a wing) in the context of a typical multibody model (here a flapping wing mechanism). Multibody assemblies (**1**) comprise bodies (**2**), bodies comprise body elements (**3**)—often all solids—and body elements comprise frames and any material attributes relevant to them (**4**).



Relevant Blocks

You can model most bodies using two blocks: Solid and Rigid Transform. The Solid block represents its namesake—a solid element. The block parameters set the attributes of the solid and a frame port, labeled **R**, provides a reference frame for connection to a model. You can create additional, custom, frames and position them using a variety of solid features.

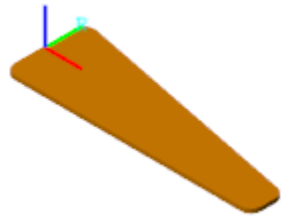
The Rigid Transform block represents a fixed spatial relationship between two frames. Whenever you add a Rigid Transform block to a frame connection line, you replace the coincidence relationship originally set by that line with a rotational and translational offset that you specify in the block dialog box.



Only rarely do bodies contain other blocks. Those that remain in the Body Elements library—Graphic, Inertia, and all in the Variable Mass sublibrary—serve special cases. Use them, for example, to add graphic markers, adjust inertia through compounding, or allow inertia to vary during simulation.

Body Visualization

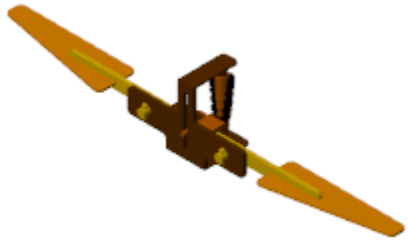
You can visualize individual solids and complete models. The right visualization tool to use depends on which of the two you want to visualize. You can visualize individual solids directly through their respective Solid blocks. In its dialog box, The Solid block provides a visualization pane that shows the geometry, frames, and color of the solid that it represents. The visualization works even if the block diagram is incomplete or invalid.



A Typical Solid Visualization

You can visualize a model, and all the bodies within it, using the Simscape Multibody visualization utility, Mechanics Explorer. The model must be free of kinematic conflicts, such as those due to mutually incompatible joints and constraints. It must also contain one Solver Configuration block for every topologically distinct multibody network—each a group of Simscape Multibody blocks that connect without breaks in their frame connection lines.

Mechanics Explorer opens a static model visualization in its initial configuration whenever you do a diagram update (**Simulation > Update Diagram**). The initial configuration is the aggregate result of all initial joint positions and angles, whose values you can specify using joint state targets. The visualization becomes dynamic when you run a simulation, although this is a task you are unlikely to perform while still modeling bodies.

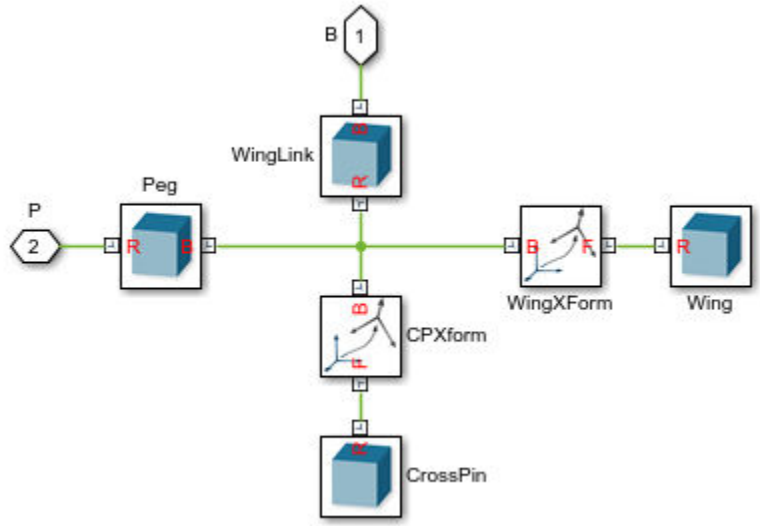


A Typical Model Visualization

For more information about visualization, see “Visualize a Model and Its Components” on page 1-60.

See It: A Typical Body

At the MATLAB command prompt, enter `sm_cam_flapping_wing`. A model of a flapping wing mechanism opens up. Look inside the mask of the subsystem block named `RightWing`. You can do this by clicking the down arrow at the bottom left corner of the block. The blocks inside the subsystem are a typical representation of a body.



The Solid blocks each represent a section of the wing body. The frame connection lines between the Solid blocks, and the Rigid Transform blocks that some of the connection lines contain, define the spatial relationships that exist between the solid sections. Open the dialog box of the Solid block named `CrossPin` and explore its attributes:

- The **Geometry** parameters are by default expanded. The block uses a `Cylinder` shape with its relevant dimensions (**Radius**, **Length**) parameterized in terms of MATLAB variables (`Rp`, `Lp`). All variables are defined numerically in the subsystem mask.

Geometry			
Shape	Cylinder		
Radius	Rp/2	cm	Compile-time
Length	3*Rp	cm	Compile-time

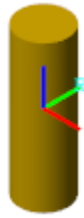
- Expand the **Inertia** parameters. The block is configured to calculate the bulk of the inertia parameters from geometry and a mass parameter (**Density**). This parameter too is parameterized in terms of a MATLAB variable (`rho`).

Inertia			
Type	Calculate from Geometry		
Based on	Density		
Density	rho	kg/(m^3)	Compile-time

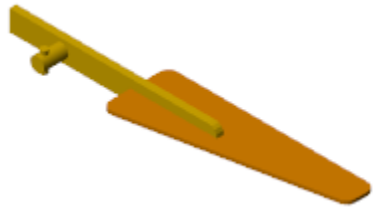
- Expand the **Graphic** parameters. The block uses a `Simple` color model with the visual properties of the solid (**Color**, **Opacity**) parameterized in terms of MATLAB variables (`lclr`) or specified numerically (`1.0`).

Graphic		
Type	From Geometry	
Visual Properties	Simple	
Color	lclr	Compile-time
Opacity	1.0	Compile-time

- In the visualization toolstrip, click the frame button. The visualization pane shows the frames associated with the solid. This solid has a single frame, the local reference frame that by default every Solid block has. The placement of this frame relative to the geometry impacts the placement of the geometry in the context of the model.



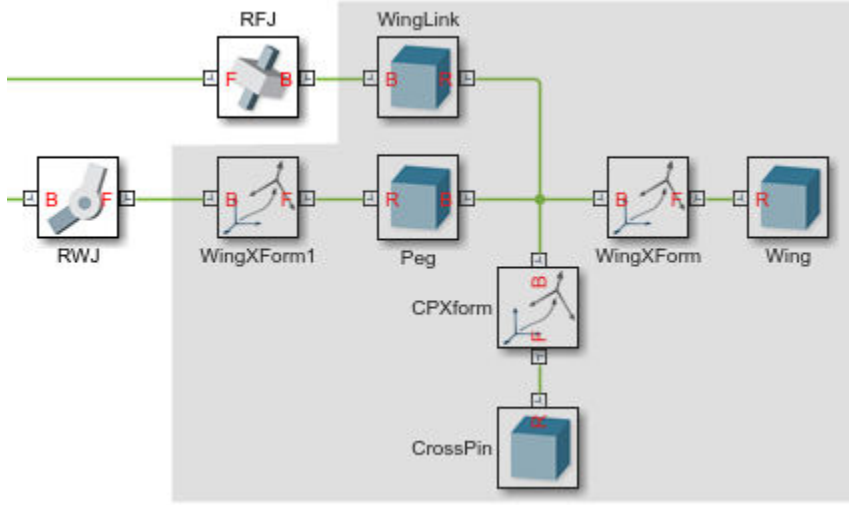
- Update the block diagram. Mechanics Explorer opens with a static visualization of the flapping wing model in its initial configuration. In the tree view pane (located on the left side of Mechanics Explorer), right-click the **RightWing** node and select `Show Only This`. The visualization pane updates to show only the body elements that compose the selected component.



Boundaries of Bodies

Direct connection lines and Rigid Transform blocks unite body elements into a single body. Such connections are treated as internal to the bodies they belong to. Joint and constraint blocks in turn separate solids into different bodies. These blocks identify the boundaries of the bodies that they connect. This distinction has practical consequences in models with Gravitational Field blocks.

The figure shows a flattened portion of the `sm_cam_flapping_wing` block diagram. The joint blocks named`RFJ` and `RWJ` separate the blocks that compose the body to their right (identified by the shaded area) from the neighboring bodies to which it connects (not shown).

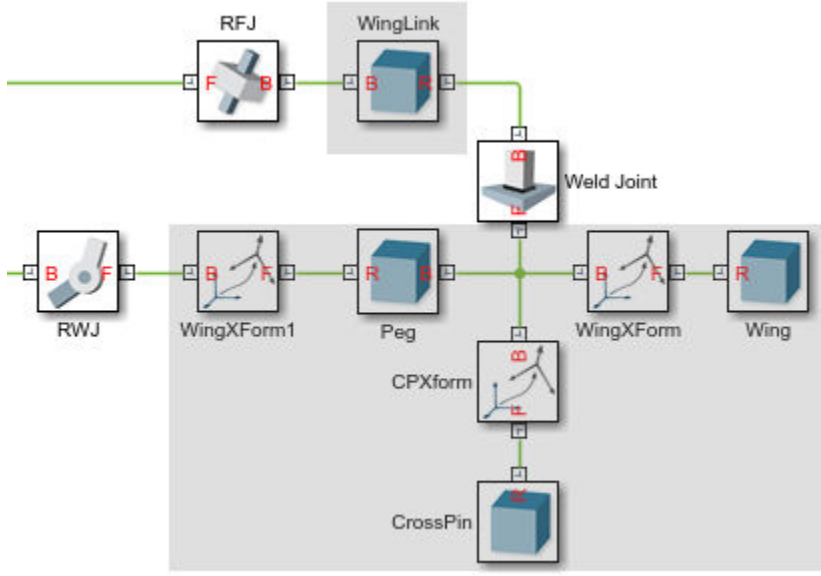


Impact on Gravitational Fields

By design, the Gravitational Field block exerts a force on the center of mass of a body. The center of mass is determined from the aggregate of all Body Elements blocks—often just Solid—that comprise the body. If two Solid blocks connect through a Rigid Transform block, they belong to the same body. A single gravitational force then acts at the center of mass of that body.

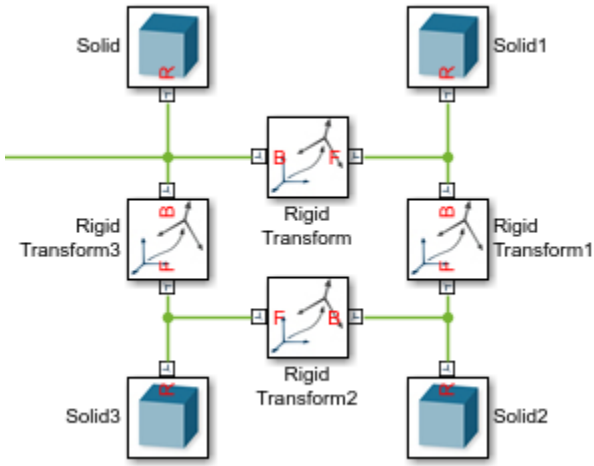
If, however, the Solid blocks connect through a Revolute Joint block, they belong to separate bodies. Two gravitational forces then apply at the individual centers of mass of those bodies. The same is true even if you replace the Revolute Joint block with a Weld Joint block. Regardless of its type, every joint block separates the body elements that it connects into separate bodies.

The figure shows the effect of adding a Weld Joint block to a block network that originally composed a single body. This joint block divides the body into two bodies, one comprising only the Solid block named `WingLink`, the other comprising the remaining Solid and Rigid Transform blocks. Gravitational Field blocks in your model, if any, would in this case exert a force at the calculated center of mass of each body.



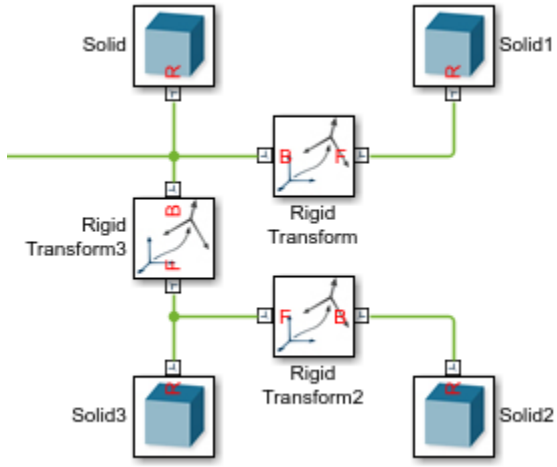
A Note on Rigid Loops

The blocks that comprise a body connect rigidly and, in the ideal case, in series. The result is a tree structure in which the path between any two frames is unique. It is technically possible, however, to rigidly connect blocks so that they form a rigid loop, a closed structure formed by connecting the open ends of a branch or tree.



A Rigid Transform Loop

Rigid transform loops are disallowed in a model. They contain redundant (and unnecessary) rigid connections and these could lead to preventable numerical errors if included in a model. If your model contains a rigid kinematic loop, you must break that loop by removing of its redundant rigid connections. The figure shows an example.



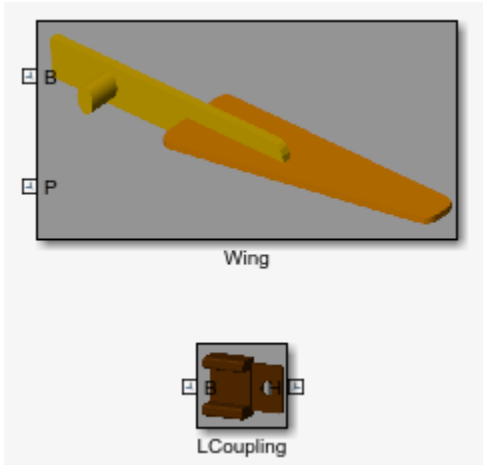
Bodies as Simulink Subsystems

It is common practice to enclose the blocks that belong to a body inside a Simulink Subsystem block. You can save the resulting Subsystem block in a Simulink library for later use in different models. The mask of the Subsystem block provides a place to define the variables and parameters that are common to the enclosed blocks—often some length, density, and color—without cluttering the workspace of a model.

You can model a body and convert it to a Simulink subsystem at any time prior to its use in a multibody model. For example, when modeling a piston engine, you can defer work on a piston body until you are ready to connect the piston to the engine housing or to the connecting rod. However, given that the shape and size of one body often depend on those of another, you should at least consider those attributes prior to focusing on assembly.

See It: A Typical Body Subsystem

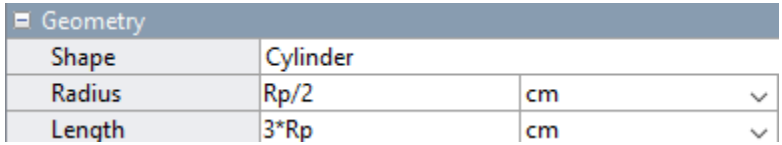
At the MATLAB command prompt, enter `sm_cam_flapping_wing_lib`. A Simulink library opens with subsystem blocks representing two of the bodies used in the `sm_cam_flapping_wing` model. Each of the blocks has frame ports for connection to a model.



Double-click the subsystem block named `Peg`. A custom dialog box opens with the key parameters required to completely define this body. The values specified here are used in the Solid and Rigid Transform blocks that compose the subsystem block.

Right-click the subsystem block and select **Mask > View Mask**. The Simulink Mask Editor opens with the parameters and code relevant to the wing body. The parameters and the MATLAB variables associated with them are defined in the **Parameters & Dialog** tab. The code used to generate the wing and spar shape profiles is defined in the **Initialization** tab.

Right-click the subsystem block and select **Mask > Look Under Mask**. The block diagram corresponding to this subsystem opens up. Open the dialog boxes of some of the blocks and note the MATLAB variables used to define many of their parameters. These are the variables defined in the subsystem mask and specified in the subsystem block dialog box.



See Also

More About

- “Compounding Body Elements” on page 1-19
- “Creating Connection Frames” on page 1-35
- “Representing Solid Geometry” on page 1-42
- “Representing Solid Inertia” on page 1-65

Compounding Body Elements

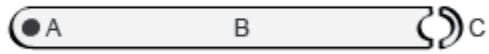
Compounding as a Modeling Strategy

It is often simpler to specify the attributes of several simple solids than those of a single complex body. Compounding is a modeling strategy whereby you can model a body as a combination of simpler body elements. You can use compounding to obtain complex geometries and inertias that you cannot otherwise (or easily) specify. The `RightWing` body in the `sm_cam_flapping_wing` model, for example, is a product of this modeling strategy.

An Example of a Compound Body

Try It: Create a Compound Geometry

Combine three `General Extrusion` shapes using Rigid Transform blocks to specify the fixed spatial relationships shared by the solid reference frames. The result of this example is not merely a compound solid geometry—it is a compound rigid body. You must use the deliberately incomplete model `smdoc_compound_link` that by default comes with your Simscape Multibody installation.



Explore the Compound Body Model

Start by exploring the `smdoc_compound_link` model and the geometry variables defined in its workspace:

1. At the MATLAB command prompt, enter the example model name, `smdoc_compound_link`. The model opens. In it are six unconnected blocks—three Solid, two Rigid Transform, and one Solver Configuration.

   The Solid blocks represent the elementary sections of a binary link and the Rigid Transform blocks the spatial relationships between the solid reference frames. The Solver Configuration block is required only for visualization in Mechanics Explorer.

2. In the Simulink menu bar, select **Tools** > **Model Explorer**. Model Explorer is a Simulink tool that you can use to explore your model workspace. All the relevant solid dimensions, including the `General Extrusion` cross-sections, are defined there.

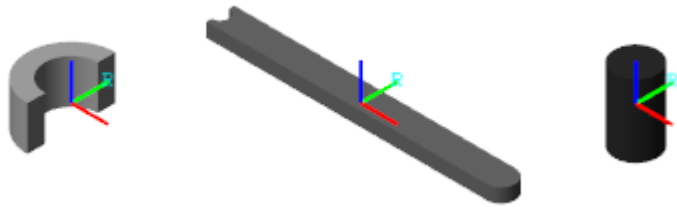
3. In the **Model Hierarchy** pane, located on the left, expand the node named after your model and select the **Model Workspace** subnode. The **Model Workspace** pane opens on the right prefilled with several lines of MATLAB code.

```
% Body Geometry Parameters
l = 20;     % Hole-to-hole distance
w = 2;      % Link width
d = 1.2;    % Hole diameter
t = 1;      % Link thickness

% Main Solid Cross-section:
A = linspace(-pi/2,pi/2)';
B = linspace(pi/2,-pi/2)';
csRight = [l/2+w/2*cos(A) w/2*sin(A)];
csLeft = [-l/2 w/2; -l/2 + d/2*cos(B) d/2*sin(B); -l/2 -w/2];
csMain = [csRight; csLeft];

% Hole Solid Cross-section:
C = linspace(pi/2,3*pi/2)';
D = linspace(3*pi/2,pi/2)';
csHole = [w/2*cos(C) w/2*sin(C);
d/2*cos(D) d/2*sin(D)];
```

   This code defines the [x, y] coordinates of the `General Extrusion` cross-sections. The cross-sections are parameterized in terms of the relevant solid dimensions, namely length, width, and hole diameter. Note the link dimensions specified in the code. The distance between the link holes (variable `l`), is 20 in what will later be units of `cm`. The link width (`w`) is 2 and the hole diameter (`d`) 1.2 in the same units.

4. Open each Solid block dialog box. The visualization pane shows the solid geometry, derived partly from the code in the model workspace, corresponding to the respective

block. In the **Geometry** parameters section, not the **Shape** parameter setting. Two solids have `General Extrusion` shapes and one a `Cylinder` shape.

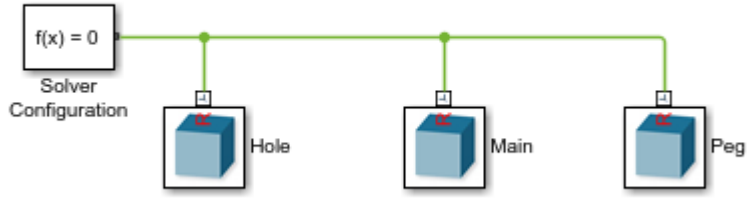


In the visualization toolstrip, click the **Show Frames** button. The visualization pane displays the solid reference frame. The placement of a reference frame relative to a solid geometry becomes important when considering the rigid transforms that you must apply between the various solid reference frames.

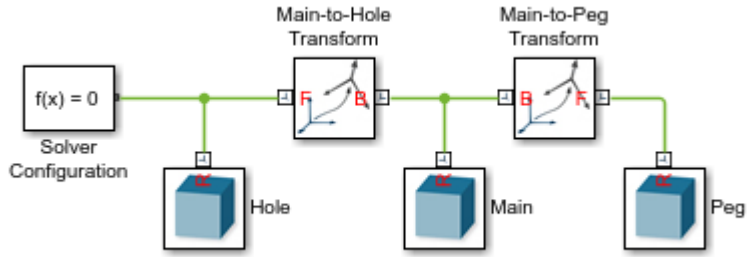
Combine the Solids Through Rigid Transforms

Complete the model by rigidly connecting the solids and specifying their spatial relationships:

1 Connect the Solid blocks as shown in the figure. The solid reference frames are, for the moment only, coincident with each other.



2 Drop the Rigid Transform blocks on the connection lines as shown in the figure. Simulink automatically connects the frame ports to the connection lines.



Pay special attention to the port positions—the B ports should both face the Solid block named `Main`. Flipping the port connections would change the relative placement of the solids in the final body.

3 In the dialog box of the Rigid Transform block named `Main-to-Hole Transform`, specify the **Translation** parameters listed below. These parameters describe a translation of half the binary link length along the -*x* axis of the base (**B**) frame—in this model held coincident with the reference (**B**) frame of the solid named `Main`.

- **Method**: `Standard Axis`
- **Axis**: `-X`
- **Offset**: `l/2`, units of `cm`

4 In the dialog box of the Rigid Transform block named `Main-to-Peg Transform`, specify the **Translation** parameters listed below. These parameters describe a translation of half the binary link length along the +*x*-axis and a translation equal to the binary link thickness along the +*z*-axis of the base (**B**) frame.

- **Method**: `Cartesian`
- **Offset**: `[l/2 0 t]`, units of `m`

5 In the Simulink menu bar, select **Simulation > Update Diagram**. Mechanics Explorer opens with a visualization of the binary link model. The body is compound—it comprises multiple solids—and can therefore be visualized in its entirety using Mechanics Explorer only. For emphasis, the solids are shown in different shades of gray.

A More Detailed Cross-Section Example

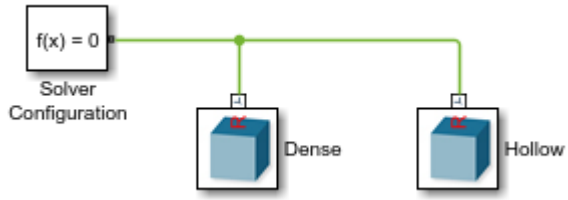
For an example showing how to specify a `General Extrusion` cross-section, see “Try It: Define a Simple Cross-Section” on page 1-53. The cross-section in that example is based on a similar, though not identical, model of a binary link. That link is treated as a simple body—one modeled as a single solid—with neither pegs nor holes. However, the strategy demonstrated there applies to other `General Extrusion` cross-sections as well. For an extension of that example showing how to include holes in a cross-section, see “Try It: Define a Cross-Section with Two Holes” on page 1-56.

Try It: Create a Compound Inertia

While it is more commonly used to represent complex geometries, compounding serves also to represent complex inertias. In particular, you can combine the inertia of a positive mass with the inertia of a negative mass, effectively subtracting one from the other.

Use this strategy to subtract the inertia associated with a bore from a cylindrical solid originally modeled without one. Represent the dense and hollow regions using Solid blocks. Set the cylinder length to 1 m and the radius to 0.25 m:

1. At the MATLAB command prompt, enter `smnew`. A new model based on the Simscape Multibody template opens up. The model contains commonly used blocks and is configured with suitable solver settings for multibody models.
2. Add a copy of the Solid block and connect it to the existing Solid and Solver Configuration blocks. The frame connection line between the blocks make their reference frames coincident in space. You can delete the remaining blocks.



3 In the dialog box of the leftmost Solid block, set the **Geometry > Shape** parameter to `Cylinder`, the **Radius** parameter to `0.25 m`, and the **Length** parameter to `1 m`. Name this block `Dense`.

4 In the dialog box of the rightmost Solid block, set the **Geometry > Shape** parameter to `Cylinder`, the **Radius** parameter to `0.20 m`, and the **Length** parameter to `1 m`. Name this block `Hollow`.

5 Set the **Inertia > Density** parameter of the `Hollow` block to the negative of the value used in the `Dense` block: `-1000 kg/m^3`. The compound body represented by the Solid blocks now has the inertia of a hollow cylinder with a bore `0.2 m` in radius.

6 Expand the **Inertia > Derived Values** node and click the **Update button** to display the inertia parameters of the `Hollow` solid. Do the same for the `Dense` solid. The mass and moments of inertia have opposite signs, as expected from the density inputs.

Derived Values		Update
Mass	-1.256637e+02	kg
Center of Mass	[+0.000000e+00, +0.000000e+00, +0.000000e...	m
Moments of Ine...	[-1.172861e+01, -1.172861e+01, -2.513274e+0...	kg*m^2
Products of Iner...	[-0.000000e+00, -0.000000e+00, -0.000000e+0...	kg*m^2

See Also

More About

- “Modeling Bodies” on page 1-5
- “Representing Solid Geometry” on page 1-42
- “Representing Solid Inertia” on page 1-65

Working with Frames

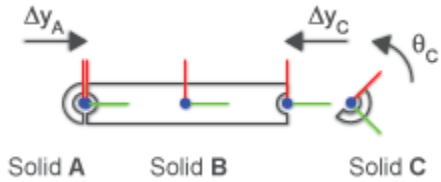
In this section...
“What Are Frames?” on page 1-25
“Role of Frames” on page 1-25
“Custom Solid Frames” on page 1-27
“What Are Frame Transforms?” on page 1-28
“Visualizing Frame Transforms” on page 1-29
“Try It: Specify a Frame Transform” on page 1-30

What Are Frames?

Frames are axis triads that encode position and orientation data in a 3-D multibody model. Each triad consists of three perpendicular axes that intersect at an origin. The origin determines the frame position and the axes determine the frame orientation. The axes are color-coded, with the *x*-axis in red, the *y*-axis in green, and the *z*-axis in blue.

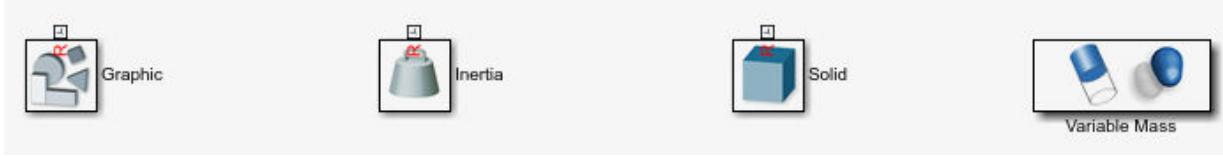
Role of Frames

Every solid component has one or more local frames to which it is rigidly attached. By positioning and orienting the component frames, you position and orient the components themselves. This is the role of frames in a model—to enable you to specify the spatial relationships between components.

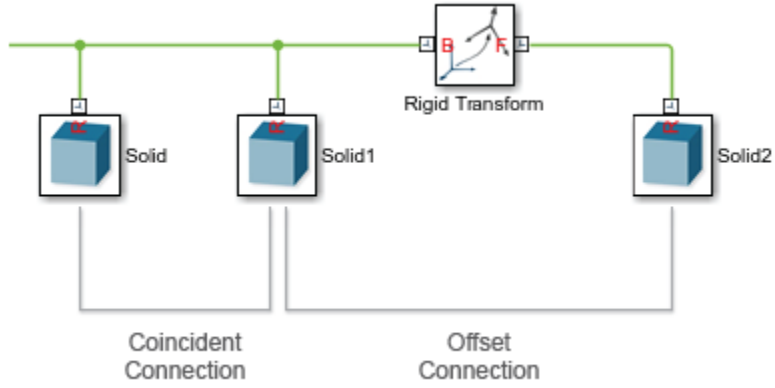


Working with Frames

A frame port identifies a local frame on a component. For example, the R frame port of a Solid block identifies the local reference frame of a solid. Every block has one or more frame ports that you connect in order to locate the associated components in space. The figure shows the reference frame ports on the **Body Elements** blocks.

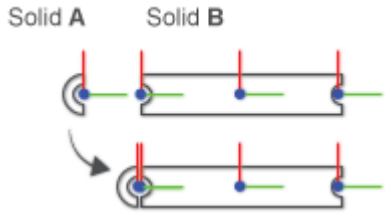


The connections between frame ports determine the spatial relationships between their frames. A direct frame connection line makes the connected frames coincident in space. A Rigid Transform block sets the rotational and translational offsets between the frames. The figure shows examples of coincident and offset frame connections.



A coincident relationship between solid frames does not, by itself, constitute a coincident relationship between solid geometries. The spatial arrangement of two solid geometries depends not only on the spatial arrangement of the respective reference frames, but also on how the geometries are defined relative to those frames.

If two geometries differ from each other, or if their positions and orientations relative to their reference frames differ from each other, then making the reference frames coincident will cause the solid geometries to be offset. In the figure, connecting the frame of Solid A to the left frame of Solid B joins the solids such that their geometries are offset from each other.



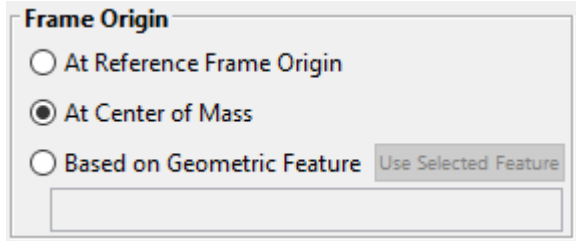
Custom Solid Frames

The Solid block provides a frame creation interface that you can use to create new, *custom*, frames. You can position and orient a custom frame using geometry features such as vertices, edges, and faces. More conveniently from an inertia standpoint, you can do the same using the center of mass and three principal axes of the solid.

Try It: Create a Custom Solid Frame

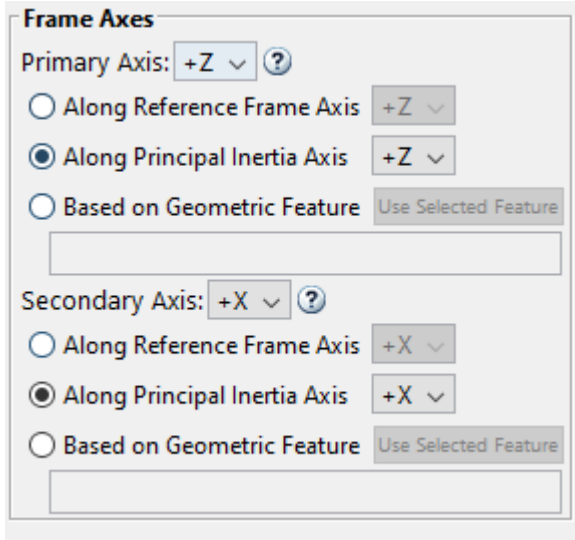
Create a custom frame using the frame creation interface of the Solid block. Then, place the frame origin at the center of mass and align the frame axes with the principal axes of inertia. The result is a frame that coincides with the principal reference frame—one in which the inertia matrix is diagonal and the products of inertia are zero.

1. At the MATLAB command prompt, enter `smdoc_lbeam_inertia`. A model opens with a solid possessing the shape of an L-beam.
2. In the Solid block dialog box, click the **Create Frame** button. The Solid block dialog box switches to a frame creation view.
3. Change the **Frame Name** parameter to `P` (for "Principal Frame"). The visualization pane and the frame port use this label to identify your new frame.
4. Under **Frame Origin**, select the radio button labeled **At Center of Mass**.

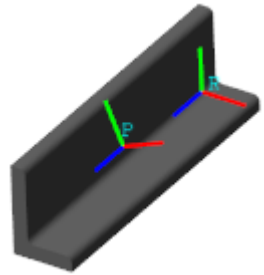


5. Under **Frame Axes > Primary Axis** and **Frame Axes > Secondary Axis**, select the radio button labeled **Along Principal Inertia Axis**. Accept the default axis

options (`+Z` and `+X`, respectively) and click **Save**. The block dialog box switches back to the main (parameters) view.

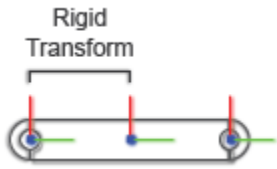


6 In the visualization toolstrip, click the **Toggle visibility of frames** button. The visualization pane shows the frames of the solid, including your new custom frame, **P**.



What Are Frame Transforms?

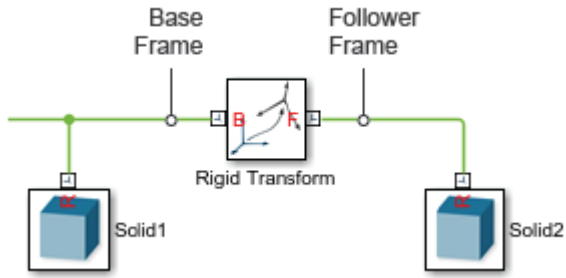
The rotational and translational offsets between frames are called transforms. If the transforms are constant through time, they are called rigid. Rigid transforms enable you to fix the relative positions and orientations of components in space, e.g., to assemble solids into rigid bodies.



Working with Frame Transforms

You use the Rigid Transform block to specify a rotational, translational, or mixed rigid transform between frames. The transforms are directional. They set the rotation and translation of a frame known as follower relative to a frame known as base.

The frame port labels on the Rigid Transform block identify the base and follower frames. The frame connected to port B serves as base. The frame connected to port F serves as follower. Reversing the port connections reverses the direction in which the frame transform is applied.



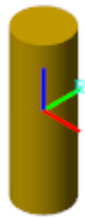
You can specify a transform using different methods. For rotational transforms, these include axis-angle pairs, rotation matrices, and rotation sequences. For translational transforms, they include translational offset vectors defined in Cartesian or cylindrical coordinate systems.

If the rotational and translational transforms are both zero, the connected frames are coincident in space. This relationship is known as *identity* and it is equivalent to a direct frame connection line between frame ports—i.e., one without a Rigid Transform block.

Visualizing Frame Transforms

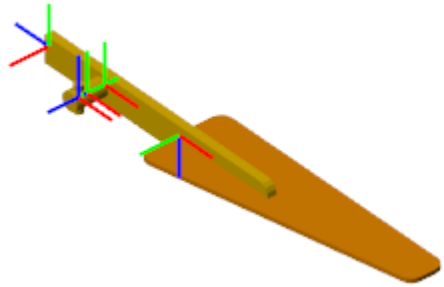
You can visualize frames and examine the transforms between frames using the Solid block visualization pane or Mechanics Explorer. Use the Solid block visualization pane to

examine the frames of a single solid element. Click the **Toggle visibility of frames** button in the visualization toolstrip to show all the solid frames.



A Frame on a Solid

Use Mechanics Explorer to visualize the frames of more than a single solid element—e.g., in compound bodies, multibody subsystems, or complete multibody models. Select **View > Show Frames** in the Mechanics Explorer menu to show all frames. Select a node from the tree view pane to show only those frames belonging to the selected component.



Frames on a Body

Try It: Specify a Frame Transform

This example shows how to offset two solids relative to each other by specifying a frame transform between the solid reference frames. The transform consists of a `-45 deg` rotation about the z axis followed by a `1 m` translation along the x-axis and a `1 m` translation along the y-axis.

Add the solids to the model

1 Drag two Solid blocks from the Body Elements library and place them in a new model.

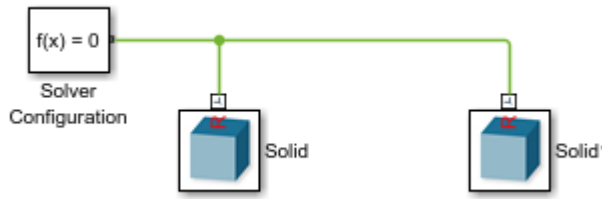
Each Solid block specifies the default geometry of a cube `1 m` in width.

2 Connect the Solid block R frame ports.

The frame connection line makes the reference frames—and cubes—coincident in space.

Visualize the solid frames

1 Drag a Solver Configuration block from the Simscape Foundation Utilities library and connect it anywhere on the model.



This block is required for model update and simulation.

2 Update the diagram, e.g., by selecting **Simulation** > **Update Diagram**.

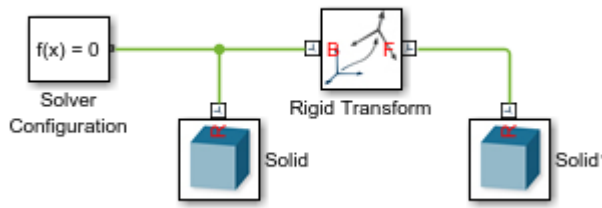
Mechanics Explorer opens with a model visualization.

3 In the tree view pane, alternately click the **Solid** and **Solid1** nodes.

The visualization pane shows the solid reference frames. The frames are coincident in space.

Apply the rotation transform

1 Drag a Rigid Transform block from the Frames and Transforms library and connect it between the two Solid blocks.



2 In the Rigid Transform block dialog box, set:

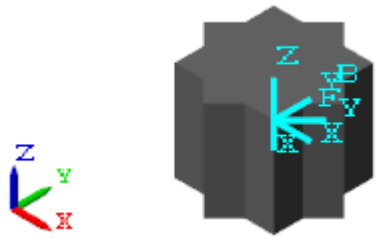
- **Rotation** > **Method** to `Standard Axis`.

- **Rotation** > **Axis** to `-Z`.
- **Rotation** > **Angle** to `45`.

3 Click **OK** and update the block diagram.

The model visualization updates to show the rotated—and still overlapping—solids.

4 In the tree view pane, click the **Rigid Transform** node.



The visualization pane shows the rotated frames.

Apply the translation transform

1 In the Rigid Transform block dialog box, set:

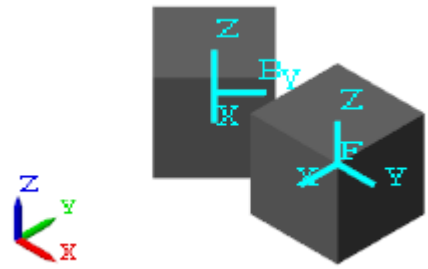
- **Translation** > **Method** to `Cartesian`.
- **Translation** > **Offset** to `[1 1 0]`.

  The array elements are the translation offsets along the base frame *x*, *y*, and *z* axes.

2 Click **OK** and update the block diagram.

The model visualization updates to show the translated solids.

3 In the tree view pane, click the **Rigid Transform** node.



The visualization pane shows the translated frames.

The Rigid Transform block always applies the rotation transform first. The translation transform is relative to the rotated frame resulting from the rotation transform. To apply the translation transform first, use separate Rigid Transform blocks for each transform and connect them in the desired order between the Solid blocks.

See Also

More About

- “Modeling Bodies” on page 1-5
- “Creating Connection Frames” on page 1-35
- “Visualize Simscape Multibody Frames” on page 5-32

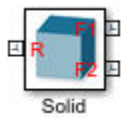
Creating Connection Frames

Frames as a Connection Points

The frames of bodies provide the connection points for the joints and constraints in your model. They determine also the relative orientations of those joints and constraints, and therefore the directions of motion that are allowed during simulation. To successfully connect bodies through joints and constraints, you must create suitable connection frames, and this is a task that is best done when modeling the bodies themselves.

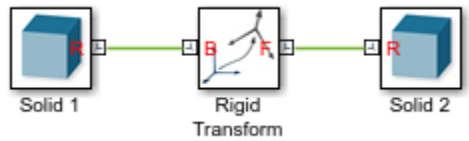
Creating and Transforming Frames

You can add frames directly to solids using the frame creation interface of the Solid block. This interface enables you to define the position and orientation of a frame interactively, in terms of key geometry features, such as vertices, edges, and faces, or in terms of key inertia features, such as the center of mass and the principal axes of inertia. Frames that you create are known as custom and appear as frame ports on the Solid block to which they belong. The figure shows a Solid block with two custom frame ports labeled **F1** and **F2**.



You can also create frames using the Rigid Transform block. This block enables you to define the position and orientation of a frame numerically, in terms of rotation and translation transforms. You can use a variety of transform parameterizations, including rotation matrices and rotation sequences in the case of rotation transforms, and Cartesian and cylindrical offset coordinates in the case of translation transforms. The figure shows a new frame (**F**) created using a Rigid Transform block from an existing solid frame.

Frames that you create using Rigid Transform blocks are independent of any specific solid features. You can place them anywhere relative to another frame as long as you can determine the transform required to obtain that placement. When placing a Rigid Transform block between two frame ports, you can more aptly think of the block as a means to offset frames that already exist. The figure shows an offset specified through a Rigid Transform block placed between two existing frames.

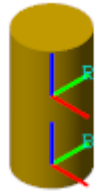


For more information about frames and transforms, see “Working with Frames” on page 1-25.

See It: Frames in a Typical Body

At the MATLAB command prompt, enter `sm_cam_flapping_wing`. A model of a flapping wing mechanism opens up. Look inside the mask of the body subsystem named `RightWing`. Note that two of the Solid blocks, `Peg` and `WingLink`, each have two frame ports. One of the ports identifies a custom frame created using the Solid block. Explore one of the custom frames:

1. Open the dialog box of the Solid block named `Peg`.
2. In the visualization toolstrip, click the frame button. The visualization pane updates to show the reference and custom frames of the solid.

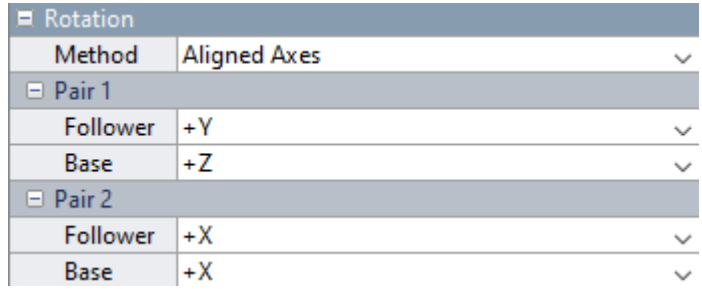


3. In the **Properties** section of the dialog box, expand the **Frames** node and click the **Edit** button. The frame creation interface opens with the current frame definition:
   - The frame origin has been placed at the center of the bottom surface of the cylinder.

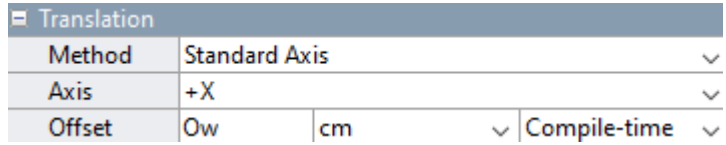
- The frame axes have been left in alignment with those of the local reference frame.
- The name of the frame matches the label of the corresponding frame port (**B**).

Note also that the frame connection lines between some of the Solid blocks contain Rigid Transform blocks, named `CPXform` and `WingXForm`. These blocks specify the rotational and translational offsets between the solid frames that they connect. Explore the transforms specified in one of the blocks:

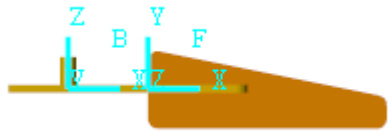
1 Open the dialog box of the Rigid Transform block named `WingXForm`. Note that the rotation transform is set by aligning two axes of the follower frame relative to two axes of the base frame.



2 Expand the **Translation** parameters. Note that the translational offset is specified along the axis (of the base frame). The translational offset is parameterized in terms of a MATLAB variable, `Ow`, whose value you specify in the `WingLink` subsystem block.



3 Update the block diagram. Mechanics Explorer opens with a static visualization of the flapping wing model. In the tree view pane, expand the `RightWing` node and click the `WingXForm` node to highlight the frames belonging to the Rigid Transform block.

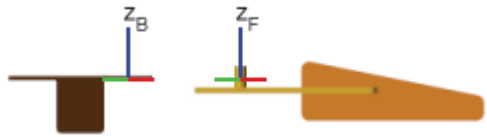


Planning Connection Frames

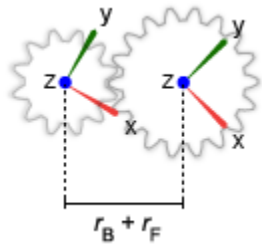
You must consider the target of a connection frame—the specific joint or constraint—when defining its placement on a body. Joints and constraints often impose special assembly requirements on the frames that they connect. These requirements impact the proper placement of a connection frame. You can find them in the reference page of the joint or constraint block.

If the target is a joint, take note of its degrees of freedom—the types of motion allowed between the joint connection frames—and of the frame axes to which they correspond. For example, the Revolute Joint block provides one rotational degree of freedom about the common *z*-axis of the connection frames on the base and follower bodies.

To connect two bodies through a Revolute Joint block, you must then place their joint connection frames so that their *z*-axes each align with the desired rotation axis on the respective body. The figure shows an example: the `Housing` and `RightWing` bodies of the `sm_cam_flapping_wing` model with connection frames properly positioned for a Revolute Joint block.



If the target of a connection frame is a constraint, such as that characteristic of gears in mesh, take note of the connection frame placements required for assembly. For example, the Common Gear Constraint block requires that the connection frames on the base and follower bodies be apart by a distance equal to the sum of their pitch radii when the meshing type is set to `External`. It requires also that the *z*-axes be parallel, and that the *x* - and *y*-axes of one be coplanar with those of the other.



The assembly requirements of constraint blocks specify how the remainder of the model must hold the connection frames, partly through body definitions and partly through other joints and constraints, for the constraint to apply without error. To see how to place connection frames on gear bodies for assembly via gear constraint blocks, see “Assemble a Gear Model” on page 2-44.

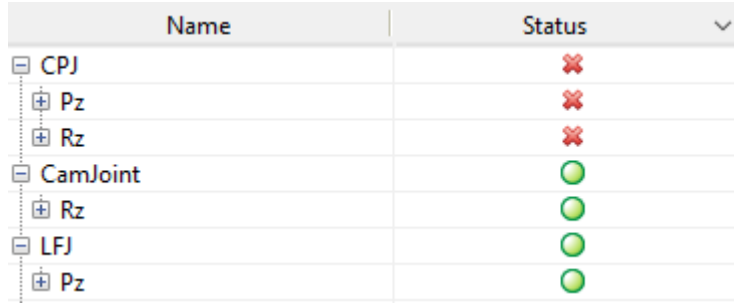
Addressing Assembly Errors

Closed-loop models, such as those of four-bar and crank-slider mechanisms, are limited in the positions and orientations that their constituent bodies can take. If the placement of a connection frame renders a joint or constraint incompatible with any other in the loop—that is, if successfully connecting one requires that another break—then assembly fails.

This is the case in a planar four-bar mechanism, for example, when the distance between the connection frames on any one link exceeds the sum of the equivalent distances in the remaining links—or when the orientations of the connection frames force the rotation axes of the joints into anything but a parallel alignment.

You can prevent many assembly errors by carefully defining the connection frames on a body with its future connections in mind. Consider both the assembly requirements specific to a joint or constraint block and the kinematic constraints imposed by the remainder of the model. In general, if you encounter an assembly failure, you must:

1. Identify the unassembled joint or constraint. Use the Simscape Multibody Model Report (**Tools > Model Report** in the Mechanics Explorer menu bar) or the Simscape Variable Viewer (**Analysis > Simscape > Variable Viewer** in the Simulink menu bar).



2 Examine the corresponding connection frames. Use Mechanics Explorer to visualize these frames. Click the name of the unassembled joint or constraint in the tree view pane to highlight its frames in the visualization pane. Compare the placement of those frames to the assembly requirements of the joint or constraint.

3 Transform the connection frames to satisfy the joint or constraint assembly requirements. You can use the Rigid Transform block to apply the required rotation and translation transforms. If a connection frame is a custom frame belonging to a Solid block, you can use that block instead to edit the frame definition.

For an example showing how to resolve an assembly failure caused by an improperly placed connection frame, see “Troubleshoot an Assembly Error” on page 2-27.

See Also

More About

- “Modeling Bodies” on page 1-5
- “Working with Frames” on page 1-25
- “Creating Custom Solid Frames” on page 1-92

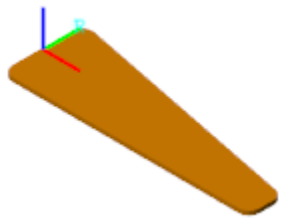
- “Visualize Simscape Multibody Frames” on page 5-32

Representing Solid Geometry

In this section...
“Geometry in a Model” on page 1-42
“Preset Solid Shapes” on page 1-45
“Imported Solid Shapes” on page 1-46
“Compound Solid Shapes” on page 1-48

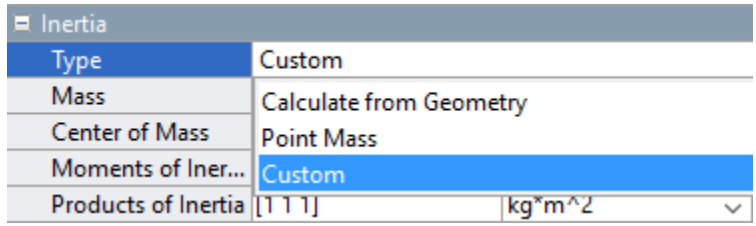
Geometry in a Model

Geometry is a key attribute of solids and of the bodies they comprise. It features in the solid visualizations provided by Solid blocks as visual aides during modeling. It features also in the multibody visualizations displayed in Mechanics Explorer following model assembly and during simulation. This is one purpose of solid geometry: to enable visualization for an entire modeling workflow, from the conception of a single solid to the simulation of a complete multibody model.



Geometry of a Body Element

Solid geometry serves a second, less visible, purpose: to simplify the specification of inertia in the Solid block. The bulk of solid inertia parameters are readily computed if both geometry and mass, or, alternatively, mass density, are known. The Solid block provides an inertia parameterization, `Calculate from Geometry`, that performs these calculations for you. You specify the solid geometry and a measure of its mass; the block carries out the required numerical integrations to obtain the remaining inertia parameters—the moments of inertia, products of inertia, and center of mass.



Geometry in Body Elements Blocks

Solid geometry differs in a practical way from frames and inertia. The latter are attributes that you can model in isolation using blocks such as Rigid Transform and Inertia. There is no equivalent, dedicated block for solid geometry. The Graphic and Spline blocks represent geometries—and provide a visualization means for those geometries—but neither is an adequate replacement for an actual solid geometry.

The Graphic block merely adds a marker to a frame, typically as a means of highlighting that frame. The Spline block adds a plane or space curve largely intended for use with the Point on Curve Constraint block. If you want to visualize solids and bodies, or benefit from the automatic inertia calculations that solid geometry enables, you must use the Solid block.

Try It: Specify a Simple Cylinder Shape

Use the Solid block to model a body with a simple preset shape—a cylinder with a radius of 5 cm and a length of 20 cm. Visualize the solid in the visualization pane of the Solid block. Ignore the relative placement of the solid in the (incomplete) model.

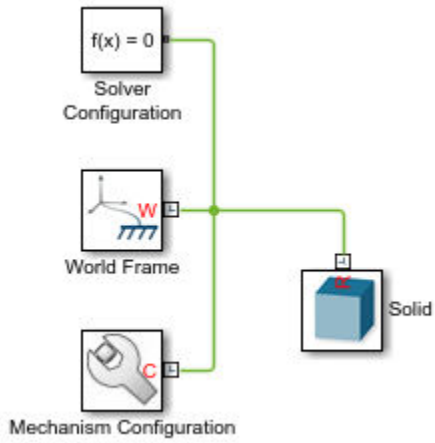
1. Add a Solid block to a new Simulink model and open the block dialog box. Note the **Geometry** parameters section, which by default specifies a `Brick` shape 1 m in side.
2. From the **Geometry** > **Shape** dropdown list, select `Cylinder`. The **Geometry** parameters section updates to show the required `Cylinder` properties—**Radius** and **Length**.
3. In the **Radius** parameter line, enter a value of `5` and select units of `cm`. You can select your units from the dropdown list or enter them manually.
4. In the **Length** parameter line, enter a value of `20` and again select units of `cm`. Note the warning in the visualization pane urging you to update the solid visualization.
5. In the visualization toolstrip, click the **Update Visualization** button. The visualization pane refreshes with the new solid geometry but, due to its small

dimensions, it is barely visible. Click the **Fit to View** button to optimize the zoom level. Ensure that the solid geometry is as expected.

6 Expand the **Inertia** parameters section and take note of the **Type** parameter setting. The automatic calculation of inertia properties from geometry is by default enabled. To complete the model of your solid, you need only ensure that its mass or mass density is set to the correct value. Click **OK** to accept the current solid settings.

Positioning and Orienting a Solid in a Model

If a Solid block is unconnected, the relative placement of that solid is undefined. To resolve the solid pose—its position and orientation—in a model, you must connect the reference frame port (**B**) or, if you prefer, a custom frame port, belonging to the Solid block. For example, connecting the **R** port to the **W** port of a World Frame block would align the solid so that its reference frame is coincident with the world frame. The figure shows such a connection



Specifying spatial relationships such as this is key to modeling in the Simscape Multibody environment. You can rotate and translate two frames with respect to one another by applying operations called *rigid transforms* between those frames. To learn more about frames and transforms, see “Working with Frames” on page 1-25.

For ease of modeling, the Solid block provides a frame creation interface. You can use this interface to append and align new frames to select geometry features, such as vertices, edges, faces, and volumes. To learn how to create frames using this interface, see “Creating Custom Solid Frames” on page 1-92

.

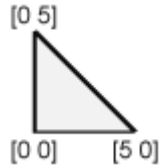
Preset Solid Shapes

The Solid block provides a sizeable array of preset shapes—those with simple parameterizations featuring readily accessible parameters, such as **Radius** and **Length**, as inputs. Preset shapes make it possible to quickly model spherical, cylindrical, and prismatic solids, among others. For greater versatility, the preset shapes include `General Extrusion` and `Revolution`—shapes whose cross-sections, be they along or about an axis, you can modify. To learn more about these shapes, see “Modeling Extrusions and Revolutions” on page 1-50.

Try It: Specify a Simple Revolution Shape

Use the Solid block to model a solid of revolution—a cone with a height of 5 ft and a base radius also of 5 ft. Visualize the solid in the visualization pane of the Solid block. Ignore the relative placement of the solid in the (still incomplete) model.

1. Add a Solid block to a Simulink model. You can reuse the Solid block from the previous solid geometry example.
2. From the **Geometry > Shape** dropdown list, select `Revolution`. The **Geometry** parameters section updates to show the required `Revolution` parameters: **Cross-Section** and **Length**.
3. In the **Cross-Section** parameter line, enter the coordinate matrix `[0 0; 5 0; 0 5]` and select units of `ft`. Each matrix row provides an [*x z*] coordinate pair, specified in that order, for a cross-section point.



4 Click the **Update Visualization** button and the **Fit to View** button. Ensure that the solid geometry is as expected. Click **OK** to accept the new solid geometry and close the block dialog box.

Specifying the Solid Cross-Sections

The Solid block generates the `Revolution` shape by sweeping the specified *xz* cross-section about the z-axis. To consistently generate a valid shape without errors, the Solid block enforces a few rules. Foremost among these is the requirement that, as you proceed from one point in the coordinate matrix to the next, the solid region lie to your left and the empty (or hollow) region to your right. The same rule applies to `General Extrusion` shapes, with one distinction: the cross-section coordinates are (*x*, *y*) pairs and the cross-section lies in the *xy* plane. To learn more about `Revolution` and `General Extrusion` cross-sections, see “Modeling Extrusions and Revolutions” on page 1-50.

Imported Solid Shapes

Rather than specify a solid shape, you can import one from an external geometry file. This option, available from the **Geometry > Shape** dropdown list of the Solid block (as `From File`), provides your best route for modeling complex or intricate solid geometries. You must obtain the desired geometry in a supported format—currently STEP (also referred to as STP) or STL.

The STEP format is recommended as it leads to what are generally smaller files than equivalent STL geometries. STEP is also the only of the two formats that allows for

automatic inertia calculation from geometry. You must explicitly specify the moments of inertia, products of inertia, and center of mass of the solid when importing an STL geometry.

Note that very large files may load slowly and delay the usually fast model update step (**Simulation > Update Diagram**). The size of a STEP or STL file depends to an extent on the application used to generate the file. You can, in some cases, reduce size by using a different application to export your solid geometry.

Try It: Import a STEP Geometry File

Use the Solid block to import a detailed bevel gear geometry. The gear geometry was created in CAD software and subsequently exported in STEP format. Visualize the solid in the visualization pane of the Solid block and ignore the relative placement of the solid in the model.

1. Add a Solid block to a Simulink model. You can reuse the Solid block from the previous solid geometry example.
2. From the **Geometry > Shape** dropdown list, select `From File`. The **Geometry** parameters section updates to show the required file import properties—**File Type**, **File Name**, and, for STL files only, **Units**.
3. From the **File Type** dropdown list, select `STEP`. This is the recommended geometry file type. STEP files are generally smaller than their STL counterparts and enable the automatic calculation from geometry.
4. In the **File Name** parameter field, enter `bevel_c.step`. This file name corresponds to an example STEP geometry that is by default on your MATLAB path. If you experience any issues, you can enter the file path:

   `matlabroot/toolbox/physmod/sm/docexamples/bevel_gear/bevel_c.step`

   where *matlabroot* is the root folder of your MATLAB installation. If you are unsure of the location of your root folder, at the MATLAB command prompt, enter `matlabroot`.
5. Click the **Update Visualization** button and then the **Fit to View** button. Ensure that the solid geometry is as expected. Click **OK** to accept the new solid geometry and close the block dialog box.

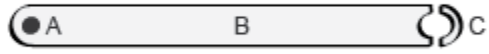
Obtaining the Solid Geometry Files

You can obtain a STEP or STL geometry file from a CAD model. Most CAD applications enable you to export your part geometries in these (among other) formats. If you are adept at using a CAD application, or have the support of someone who is, you can create a detailed solid geometry in CAD, export it in a STEP or STL file, and import the final geometry file into a Solid block.

If you lack a license to a professional CAD application, open-source software such as FreeCAD may provide a suitable alternative. Onshape, a professional, full-cloud CAD application, provides free subscription plans. This tool has the advantage of allowing you to import complete multibody assemblies into the Simscape Multibody environment using the `smexportonshape` function. For more information, see “Onshape Import” on page 6-21.

Compound Solid Shapes

If you cannot obtain a STEP or STL file with the desired solid geometry, you can still approximate that geometry—by combining simpler preset shapes into a larger, compound, shape. You must use multiple Solid blocks—one for each preset solid shape. Often, you must also use Rigid Transform blocks, to specify the spatial relationships that exist between the solid reference frames. The figure shows a solid geometry that you can model as a compound shape—a binary link with a hole section (labeled **A**), a main section (**B**), and a peg section(**C**).



For an example showing how to model this compound body, see “Try It: Create a Compound Geometry” on page 1-19.

See Also

More About

- "Modeling Extrusions and Revolutions" on page 1-50
- "Representing Solid Inertia" on page 1-65
- "Manipulate the Color of a Solid" on page 1-102
- "Visualize a Model and Its Components" on page 1-60

Modeling Extrusions and Revolutions

In this section...
“Extrusions and Revolutions” on page 1-50
“The Cross-Section Profiles” on page 1-52
“Cross-Sections with Holes” on page 1-55
“From Cross-Sections to Solids” on page 1-58

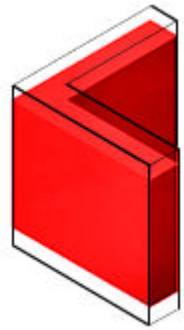
Extrusions and Revolutions

For solids with custom cross-sections, the Solid block provides the more versatile `General Extrusion` and `Revolution` shapes. These shapes enable you to create truer solid representations than simpler shapes such as `Brick`, `Cylinder`, and `Sphere` often allow. Use them when modeling solids that have arbitrary yet constant cross-sections along or about an axis.

What is a General Extrusion?

A `General Extrusion` is a linear sweep of a custom cross-section along an axis that is normal to the cross-section plane. The sweep spans the length specified in the Solid block dialog box. The cross-section can have an arbitrary outline and one or more hollow sections—though the rules for specifying cross-sections differ slightly when holes are present.

`General Extrusion` examples include straight beams, plates, spars, struts, and rods. The figure shows an angle beam, a `General Extrusion` whose cross-section consists of two thin rectangles arranged in an L shape. The Solid block sweeps the L shape linearly out of the cross-section plane to obtain the final beam geometry.



`General Extrusion` as a Linear Sweep

A Note on Regular Extrusion Shapes

The Solid block provides a second extrusion shape, named `Regular Extrusion`. This shape is a simpler version of `General Extrusion` and it is suited only for solids whose cross-sections are regular polygons—those with sides of the same length. The cross-section outline is fixed by the number of sides of the polygon and it cannot contain holes.

What is a Revolution?

A `Revolution` is an angular sweep of a cross-section about an axis that lies on the same plane as the cross-section. The sweep can span a full revolution or a lesser angle between `0` and `360` degrees. As with `General Extrusion` shapes, the cross-section can have an arbitrary outline with or without holes.

`Revolution` examples include cones, domes, pistons, gear shafts, and pipe bends. The figure shows a cylindrical peg, a `Revolution` whose cross-section, like the angle beam, consists of two thin rectangles arranged in an L shape. The Solid block sweeps the L shape about an axis lying on the cross-section plane to obtain the final peg geometry.

`Revolution` as an Angular Sweep

The Cross-Section Profiles

You specify the cross-sections numerically, as MATLAB matrices populated with the coordinate pairs of select cross-section points. Each matrix row provides a coordinate pair for one point. There is no upper bound on the number of rows of a coordinate matrix, but a minimum of three is required to completely define a closed shape.

```
% Coordinate Matrix Example:
% Square Cross-Section with Center at [0, 0] and Side Length 2
[
-1 -1; % Lower left corner
1 -1; % Lower right corner
1 1; % Upper right corner
-1 1; % Upper left corner
]
```

The coordinate pairs are treated as (*x*, *y*) values in the case of `General Extrusion` shapes and as (*x*, *z*) values—specified in that order—in the case of `Revolution` shapes. The coordinates are resolved in the reference frame of the Solid block, with the (0, 0) pair coinciding with the origin of that frame. It is common practice to parameterized the coordinates in terms of MATLAB variables associated with key solid dimensions—for example, radius or length.

From Coordinates to Cross-Sections

The coordinate pairs connect sequentially in the order implicit in the coordinate matrix. The connections are by means of straight line segments. The result is a closed polyline

that separates the region to be filled with material (the solid part) from the region to be left hollow (any holes that might be present and the empty surroundings).

The boundary between the two regions is such that, as you proceed along the polyline from one point to the next, the solid region lies to your left and the hollow region to your right. The first and last coordinate pairs are often the same, but if they are not, a connection line is inserted between them to ensure that the cross-section is in fact closed. The animated figure shows the drawing of a binary link cross-section without holes.

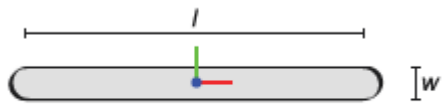
Note that the cross-section is invalid if at any point the polyline crosses itself. However, it is okay for two line segments to be arbitrarily close or even coincident with each other. In fact, you can exploit this property to specify a cross-section that has one or more holes.

A Special Constraint for Revolutions

`Revolution` coordinate matrices are subject to a special constraint: *x*-coordinates cannot be negative. This rule follows partly from the revolution axis (*z*) used by the Solid block. When sweeping the cross-section, any areas to the left of this axis (those with negative *x*-coordinates) become overlapped with those to the right (positive *x*), resulting in an unexpected solid geometry. To prevent this issue, an error is issued if a `Revolution` cross-section is found with negative *x*-coordinates.

Try It: Define a Simple Cross-Section

Consider the cross-section shown in the figure. This cross-section belongs to a binary link with round ends and no holes. Parameterize the cross-section in terms of the dimensions shown and specify it in the form of a coordinate matrix.



Binary Link Cross-Section (No Holes)

Start by opening a new MATLAB script and save it in a convenient location under the name `modelParams`. Add two variables for the dimensions shown in the figure, length (`l`) and width (`w`). Set the length to `20` and the width to `2` (in what will later be units of `cm`).

```
l = 20;
w = 2;
```

Define the round ends as semicircles. First, generate two arrays with the angular spans of the left and right ends. These arrays enable you to parameterize the (`x`, `y`) coordinates using simple trigonometric expressions. Each array has five points, but for smoother shapes you can specify more. The transpose symbol (`'`) ensures that `A` and `B` are column arrays.

```
A = linspace(-pi/2, pi/2, 5)';
B = linspace(pi/2, 3*pi/2, 5)';
```

Define the coordinate matrices of the right end (`csRight`) and left end (`csLeft`). The first column of each matrix corresponds to the *x*-coordinate. The second column corresponds to the *y*-coordinate. The *x*-coordinates of the two ends are offset in opposite directions by `l`/2.

```
csRight = [l/2 + w/2*cos(A) w/2*sin(A)];
csLeft = [-l/2 + w/2*cos(B) w/2*sin(B)];
```

Combine the coordinate matrices into a single matrix named `cs`. This is the matrix that you must specify in the **Cross-Section** parameter of the Solid block. Note that the straight segments of the cross-section are automatically generated when the end points of the semicircles are connected.

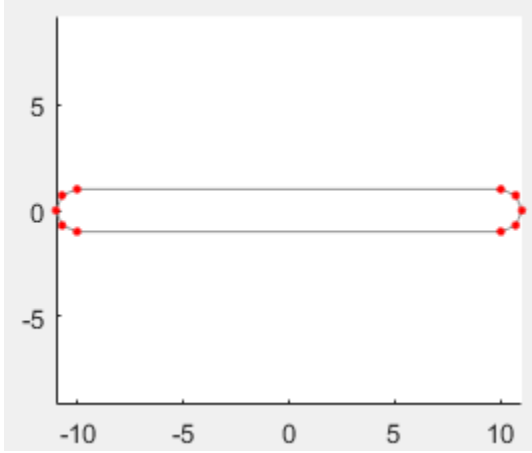
```
cs = [csRight; csLeft];
```

You can visualize the cross-section outline using the MATLAB® `plot` command. Enter the code shown below at the MATLAB command prompt.

```
figure; hold on; axis equal;
plot(cs(:,1), cs(:,2), 'Color', [0.6 0.6 0.6], 'Marker', '.',...
'MarkerSize', 9, 'MarkerEdgeColor', [1 0 0]);
```

The plot shows the cross-section of the binary link. The points in the coordinate matrix are shown as red dots. The resulting cross-section outline is shown as a light gray line.

Notice that the end sections each comprise five points—the number specified in the angular span arrays.



Cross-Sections with Holes

The coordinate matrix should always represent a single continuous path. This rule works well when specifying a cross-section without holes but it demands extra care when holes exist. Because the outline of a hole is not contiguous with the outline of the cross-section, you must now add a thin cut between the two. The cut enables you to traverse the cross-section and hole outlines in a single loop.

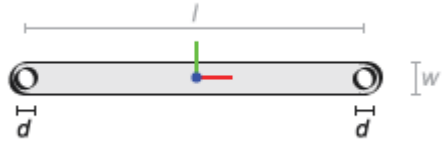
Consider a binary link with a hole at one end. The cross-section of this body comprises two closed paths—one for the cross-section outline, the other for the hole. The paths are physically separated. However, you can connect them by cutting each path at a vertex and joining the cut vertices with additional line segments. The animated figure shows the drawing of a binary link cross-section with one hole.

You can extend this approach to cross-sections with multiple holes. Note that each hole must have a cut. There is no single best way to approach the cuts. The key is to plan them so that you can traverse the cross-section—and all of its holes—in a single

continuous path while keeping the polyline from intersecting itself. The animated figure shows the drawing of a binary link cross-section with two holes.

Try It: Define a Cross-Section with Two Holes

Modify the coordinate matrix in your `modelParams` script to include two identical holes as shown in the figure. Save the script often as you go.



Binary Link Cross-Section (Two Holes)

Start by adding a new variable for the hole diameter (`d`). Set the diameter to `1.2` (in what will later be units of `cm`).

```
d = 1.2;
```

Generate a new angular span array for the left and right holes. The holes are drawn in the same order and a single array suffices. The array elements are ordered in a clockwise direction, ensuring that when generating the cross-section the solid region stays to the left. The number of array elements has doubled to reflect the wider angular span of the holes (360° vs 180°).

```
C = linspace(3*pi/2, -pi/2, 10)';
```

Define the outlines of the left hole (`csLeftHole`), the right hole (`csRightHole`), and the connection line between the two (`csConnLine`). The x-coordinates are shifted left by half the length (`l/2`) for the left hole and right by the same distance for the right hole.

```
csLeftHole = [-l/2 + d/2*cos(C) d/2*sin(C)];
```

```
csRightHole = [+l/2 + d/2*cos(C) d/2*sin(C)];
csConnLine = [-l/2 -w/2; +l/2 -w/2];
```

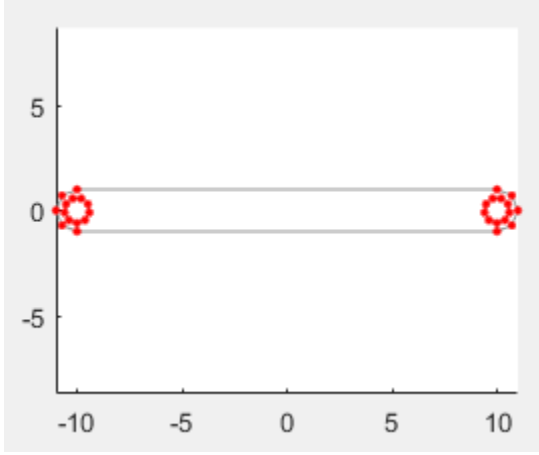
Add the new coordinate matrices to the existing `cs` matrix. The order of the matrices determines the order in which the complete cross-section is drawn. The result is a variable that you can specify in the **Cross-section** parameter of the Solid block.

```
cs = [csRight; csLeft; csLeftHole; ...
    csConnLine; csRightHole];
```

As before, you can visualize the cross-section outline using the MATLAB `plot` command. Ensure that the plotting code shown below is included in your script. Then, run the script to generate the plot.

```
figure; hold on; axis equal;
plot(cs(:,1), cs(:,2), 'Color', [0.6 0.6 0.6], 'Marker', '.',...
'MarkerSize', 9, 'MarkerEdgeColor', [1 0 0]);
```

The plot shows the cross-section of the binary link. The points in the coordinate matrix are shown as red dots. The resulting cross-section outline is shown as a light gray line. Note that the hole sections each comprise ten points—the number specified in the angular span arrays.



From Cross-Sections to Solids

The *z*-axis of the reference frame of the Solid block serves as the sweep axis in both the `General Extrusion` and `Revolution` solids. The specified cross-section is swept along this axis in the case of `General Extrusion` solids and about this axis in the case of `Revolution` solids. The sweep is symmetrical with respect to the cross-section plane: it runs half of the sweep length or angle in each direction of the sweep axis. This symmetry leaves the cross-section plane—and therefore the origin of the reference frame—halfway between the ends of the sweep.

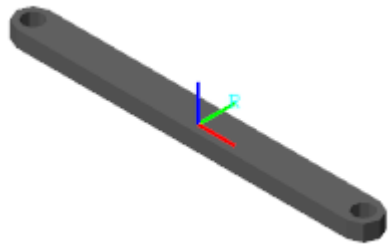
Try It: Use Your Cross-Section to Model a Solid

1. Open a new Simulink®model and, from the Bodies library, add a Solid block. You can click the model canvas, type the block name, and make a selection from the options shown. Save the model in a convenient location as `binaryLinkSolid`.
2. In the Solid block dialog box, set the **Geometry** parameters as shown in the table. Set the parameter units to `cm`. The **Cross-Section** parameter is defined in terms of the `cs` variable in your `modelParams` script.

Parameter	Value
Shape	`General Extrusion`
Cross-Section	`cs`
Length	1

3. Load your `modelParams` script to your model workspace:
   a. In the Simulink menu bar, select **Tools** > **Model Explorer**. You use this tool to load the `modelParams` script that you previously created onto your model workspace.
   b. In the **Model Hierarchy** pane, expand the node corresponding to your model (**binaryLinkSolid**) and select **Model Workspace**.
   c. In the **Model Workspace** pane, set the **Data source** parameter to `MATLAB File` and browse for your `modelParams` script. Click the **Reinitialize from Source** button to load variables defined in the script.
4. In the Solid block dialog box, click the **Update Visualization** button, . The visualization pane refreshes with the final solid geometry.

Click the **Fit to View** button to scale the binary link to the size of the visualization pane. Click the **Toggle visibility of frames** button to show the solid reference frame. The reference frame origin coincides with the [0, 0] cross-section coordinate and lies halfway between the extrusion ends.



Note the jagged appearance of the round ends and holes. This effect results from the small number of points used in the round portions of the coordinate matrix—`csLeftEnd`, `csRightEnd`, `csLeftHole`, and `csRightHole`. Increase the number of elements in the angular span arrays to obtain a smoother geometry.

See Also

More About

- "Representing Solid Geometry" on page 1-42
- "Representing Solid Inertia" on page 1-65
- "Manipulate the Color of a Solid" on page 1-102
- "Visualize a Model and Its Components" on page 1-60

Visualize a Model and Its Components

In this section...
"Visualize a Complete Multibody Model" on page 1-60
"Visualize an Individual Solid Geometry" on page 1-62
"A Note on Imported Geometries" on page 1-63

Visualize a Complete Multibody Model

Model visualizations open in **Mechanics Explorer**—the Simscape Multibody visualization utility. By default, Mechanics Explorer starts automatically when you first update a model (**Simulation > Update Diagram**) or simulate a model (**Simulation > Run**). To change this setting, see "Enable Mechanics Explorer" on page 5-2.

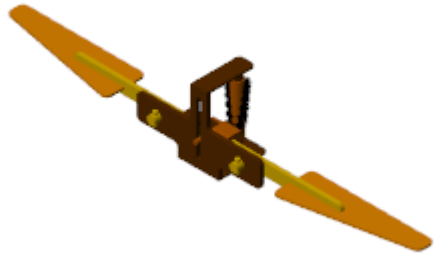
The visualizations consist of the multibody subassemblies, compound bodies, and simple bodies present in your models. Mechanics Explorer displays model entities such as solid geometries, spline curves, inertia icons, and frames. You can selectively show and hide individual entities using the context-sensitive menu of the Mechanics Explorer tree view pane. See "Selective Model Visualization" on page 5-19.

A visualization is static when you update a model and dynamic when you simulate a model. The static visualization in this case shows the model in its initial configuration, with the joints in their initial states. The dynamic visualization shows a 3-D animation that you can record—using either **Video Creator** or the `smwritevideo` function—see "Create a Model Animation Video" on page 5-39.

You can manipulate the viewpoint using the tool strip located above the visualization pane. The tool strip enables you to rotate, roll, pan, and zoom the model view. A **Camera Manager** enables you to create dynamic cameras that move with the model to keep it in view during simulation—see "Visualization Cameras" on page 5-10.

Try It: Visualize a Multibody Model

1. At the MATLAB command prompt, enter `sm_cam_flapping_wing`. The model shown in the flapping-wing featured example opens.
2. In the Simulink menu bar, select **Simulation > Update Diagram**. Mechanics Explorer opens with a view of the model in its initial state.



3 Select **Simulation** > **Run**. Mechanics Explorer plays an interactive 3-D animation based on the simulation results.

A Note About Inertia Ellipsoids

Equivalent inertia ellipsoids provide an intuitive means to visualize variable inertias (modeled using the General Variable Mass block). These ellipsoids are rendered dynamically, with their dimensions and poses obtained at each time step from the specified instantaneous inertial properties. Ellipsoid visualization is not available on model update. Variable inertial properties are specified through physical signals whose values are inaccessible to the blocks until the simulation begins.

Inertia Ellipsoid in a Tank Truck Model

Visualize an Individual Solid Geometry

Blocks such as Solid and Spline provide visualization panes that show the geometries specified in the block **Parameters** sections. Use the block visualizations to catch geometry errors as they occur—for example, incorrect dimensions, colors, and, in Solid blocks, frame placements.

The block visualizations are similar to those provided in Mechanics Explorer. You can rotate, roll, pan, and zoom the view using a tool strip located above the visualization pane. The tool strip includes buttons for standard views such as `Front`, `Top`, and `Isometric`. You can show or hide the frames of the block.

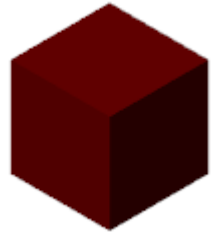
You can refresh block visualizations without updating or simulating a model. The visualizations refresh the moment you click the **Update Visualization** button in the visualization tool strip. Using MATLAB variables in block parameters does not affect your ability to refresh a visualization.

Try It: Visualize a Solid

1. At the MATLAB command prompt, enter `smnew` to open the Simscape Multibody model template. The template contains commonly used blocks, including Solid.
2. Open the Solid block dialog box. The visualization pane is by default expanded to show an isometric view of the default solid—a gray brick.

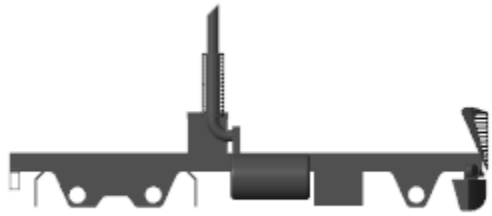
3 Set the **Graphic** > **Color** parameter to `[0.5 0 0]`—an RGB vector corresponding to a dark red color.

4 Click the **Update Visualization** button, located above the solid visualization pane. The visualization updates to show what is now a red brick.



A Note on Imported Geometries

You can import solid geometries from STEP or STL files. Corrupt or invalid geometry files cause visualization and simulation issues. Geometries associated with such files are not shown during model visualization. The corresponding solid inertias cannot be automatically computed from the solid geometries and the simulation fails if the **Inertia** > **Type** parameter is set to `Calculate from Geometry`.



An Imported Solid Geometry

See Also

Related Examples

- “Manipulate the Visualization Viewpoint” on page 5-5
- “Create a Model Animation Video” on page 5-39
- “Go to a Block from Mechanics Explorer” on page 5-37

Representing Solid Inertia

In this section...
“Representing Inertias” on page 1-65 “Compounding Solids and Inertias” on page 1-70

Representing Inertias

Inertia is a basic attribute of anything you might construe as a body. It is a resistance to a change in one’s state of motion, and, equivalently, a measure of the force or torque needed to induce a certain acceleration. Unlike other solid attributes, such as geometry or color, it is strictly required for the simulation of a model. In particular, the ends of a joint—its *frames*—must each connect to an inertia, which is to say that where motion is allowed, there must exist an inertia for an applied force or torque to act upon.

You can model an inertia element in isolation, without the intent to represent a body. Such inertias are useful, for example, when simulating the vibrations induced by a clump of mud on a rotating automobile wheel. The clump is separate from the wheel body and you can model it as such. In addition, its geometry and color are in this case trivial details and you can disregard them for modeling purposes. In so doing, you treat the clump as a plain inertia—one lacking any attributes other than inertia.

Isolated plain inertias are uncommon in a model. Generally, you account for inertia in the course of modeling a complete body—something with geometry and color, like a wing in the flapping wing mechanism discussed in “Modeling Bodies” on page 1-5. You start with a concept of the body, model that body as a rigid collection of solids, and specify the attributes of those solids to obtain a complete representation of the body. Solids are the things that you model and inertia merely one of their attributes.

A Body (1) and a Plain Inertia (2)

Relevant Blocks

You add inertia to a model using blocks from the Body Elements library. Relevant blocks include Solid, Inertia, and those in the Variable Mass sublibrary. You can model a

complete solid or a plain inertia. Either can have fixed or variable inertia parameters, though the exact parameterization, and therefore the type of solid or inertia, depends on the block. The term “solid” is used here to denote an element whose attributes extend beyond merely inertia and the term “inertia,” when used to refer to an element, one whose attributes encompass only inertia.

Fixed Solids

Use the Solid block when modeling solids and the bodies they comprise. This block enables you to specify geometry and color, key attributes if solid visualization is important to you. It enables you also to have the less accessible parameters of rotational inertia automatically computed from the solid geometry and either mass or mass density. Even in cases where geometry and color are superfluous details, the Solid block is often the most convenient means of specifying inertia. Note that the geometry and inertia parameters of the Solid block are strictly constant. To model solids with either as a variable attribute, you must use blocks from the Variable Mass sublibrary.

Variable Solids

Use the solid blocks in the Body Elements > Variable Mass library to model complete solids with variable inertia parameters, such as mass, and inertia-dependent dimensions, such as length and radius, that can vary dynamically with the inertia inputs. Blocks that represent solids are identified as such by having the word *Solid* in their names—for example, Variable Cylindrical Solid and Variable Brick Solid. These blocks differ from the Solid block in the parent library in that one or more inertia parameters can change, and from the General Variable Mass block in that they possess geometry and color.

Fixed Inertias

Use the Inertia block as a means of adjusting the inertia of a solid or body. Geometry and color are considered irrelevant for modeling purposes. You can subtract a mass to account for the existence of a hollow region, such as an empty compartment in a vessel originally modeled without one. You can also add a mass to account for the presence of small disturbances, such as the clumps of mud that sometimes linger on an automobile wheel. Note that you can make the same adjustments, sometimes more intuitively, using the more sophisticated Solid block.

Variable Inertias

Reserve the General Variable Mass block for the special cases in which mass, center of mass, or the inertia tensor must change in response to some input—often just time itself —without making assumptions about solid geometry. You can model events such as the scooping of a load by a backhoe (an example of a variable mass), the movement of an

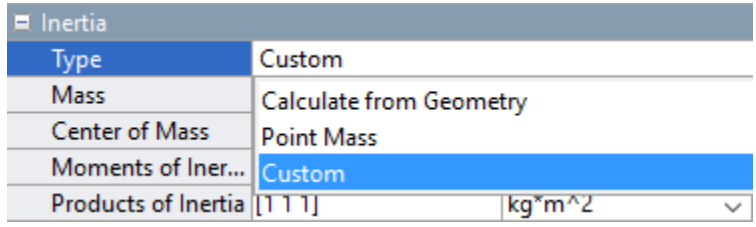
occupant on a manlift (an example of a variable center of mass), and the sloshing of a fluid load contained in a tank truck (an example of a variable inertia tensor).

Inertia Parameters

Solid blocks have access to geometry data and can therefore calculate inertia parameters given a shape and a mass. This feature greatly reduces the number of parameters that you must specify in a model. Automatic inertia calculation is always enabled in variable solid blocks, such as Variable Cylindrical Solid and Variable Cylindrical Brick. It is by default enabled in the Solid block, meaning that you can in this case change this setting.

You can also specify the inertia parameters explicitly, for example, to precisely capture the inertia of a body for which you have only a rough geometry. An example is an odd-shaped link, for example one typical of a backhoe excavator arm, that you have approximated using a simple `Brick` shape. The solid geometry is in this case not very accurate and you may prefer to specify the inertia parameters using CAD (or other) data.

If you choose to specify inertia explicitly, there are two parameterizations that you can use. One enables you to treat the solid or inertia as a point mass: the `Point Mass` parameterization. The other enables you to treat the solid or inertia as a distributed mass: the `Custom` parameterization. You can select the option best suited for your application using the **Inertia** > **Type** dropdown list.



Note that the `Point Mass` and `Custom` parameterizations are available only in those blocks that support the explicit specification of inertia. The variable solid blocks in the Body Elements > Variable Mass library do not provide either. In those blocks, the center of mass and inertia tensor are strictly constrained to the solid geometry and density and are, for this reason, always automatically calculated during simulation.

The Point Mass Approximation

A point mass is an approximation that has as its only inertial parameters the center of mass and the total mass—a measure of translational inertia and therefore of the resistance to a sudden change in translational velocity. Rotational inertia is assumed

negligible and is ignored. The location of the center of mass can vary with respect to the origin of the local reference frame.

Inertia		
Type	Point Mass	
Mass	1	kg

Custom Mass Distributions

A distributed mass is a more general representation of inertia. It has among its inertial parameters not only the total mass and center of mass, but also the moments of inertia, and products of inertia. The moments and products of inertia comprise what is known as the inertia tensor or matrix. Together, these parameters suffice to completely describe, from a multibody modeling perspective, the distribution of a mass in space.

Inertia		
Type	Custom	
Mass	1	kg
Center of Mass	[0 0 0]	m
Moments of Iner...	[1 1 1]	kg*m^2
Products of Inertia	[1 1 1]	kg*m^2

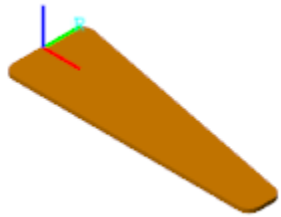
A Note on Joint Connections

Use caution when connecting inertias with zero moments of inertia, such as point masses, to joints with rotational degrees of freedom—those composed at least in part of revolute or spherical joint primitives. The combined moment of inertia about the rotational axes of the joint must be nonzero on each side. The reason for this is simple: the angular acceleration about an axis becomes infinite regardless of the torque applied if the moment of inertia about that axis is zero. This behavior is not physical and is disallowed in a model.

Reference Frames

Blocks in the Body Elements library have reference frame ports that you connect to resolve the placement of the respective elements—solids, inertias—in the context of a model. The reference frames are a rigid part of those elements and naturally move with them as a unit. They are used, directly or indirectly, to define the inertias and, in solids, the geometries of the elements.

If the concept of a frame is foreign to you, see “Working with Frames” on page 1-25. Succinctly, a frame is an axis triad much like a Cartesian coordinate system. It has a position and orientation that you can define using the frame creation interface of the Solid block or the parameters of the Rigid Transform block. All positions and orientations in a model—of solids, inertias, joints and constraints, forces and torques, sensors—are defined through frames.



Reference Frame of a Solid

Visualization Options

You can visualize solids and inertias in a model. The type of visualization that you get depends on the block that you use. Solid blocks, including those from the Body Elements > Variable Mass library, enable you to visualize their respective elements using the geometries that you specify. You can also visualize the solid using a simple graphic marker such as a sphere—for example, to highlight its position in cases where geometry is known to be inaccurate.

Inertias lack geometry and color and naturally do not support geometry-based visualization. You must visualize such elements using alternative means. If the element is associated with a General Variable Mass block, you can use the same simple graphic markers available in the Solid block or an equivalent inertia ellipsoid—a shape whose dimensions depend directly on the inertia parameters that you specify. The visualization of an element associated with an Inertia block is always in the form of an inertia icon.

For more information on visualization, see “Visualize a Model and Its Components” on page 1-60.

Try It: Add an Inertia to a Model

Add a plain fixed inertia to a double-pendulum model, position it at the free end of the outer link, and set its mass to 25 g using the `Point Mass` parameterization:

1. At the MATLAB command prompt, enter `smdoc_double_pendulum`. A model of a double-pendulum opens up. In it are three Simulink Subsystem blocks, each representing a body. Save the model under a different name in a convenient folder.
2. From the Body Elements library, add an Inertia block and connect its reference frame port (labeled **R**) to the rightmost frame port of the `Binary Link A1` block. The frame associated with this port is located at the free end of the double pendulum.
3. In the Inertia block dialog box, set the Mass parameter to `25 g`—a value roughly equivalent to a quarter of the mass of a binary link (`130 g`). The `Point Mass` parameterization in this block enables you to ignore the rotational inertia parameters.
4. Simulate the model (**Simulation** > **Run**). Mechanics Explorer opens with a dynamic visualization of your updated double-pendulum. Notice the inertia icon used to denote the location of your inertia element.

See “Try It: Specify a Custom Inertia” for an example showing how to specify the parameters of a Custom inertia.

Compounding Solids and Inertias

As solid shapes grow in complexity, inertia parameters become increasingly cumbersome to specify and a different approach may suit you better: compounding. You can conceive of a complex solid or inertia as a collection of simpler elements and specify their inertia parameters explicitly, if using the Inertia or General Variable Mass block, or configure them for automatic calculation, if using a solid block.

When you rigidly connect the simpler elements—via frame connection lines and, if needed, Rigid Transform blocks—you obtain an aggregate whose inertia properties mirror those of the complex solid or inertia you intended to represent. The binary link shown in the figure serves as an example. You can divide the link into three sections, represent each section using a separate block, and connect the respective reference frames using appropriate rigid transforms.

For an example showing how to specify the geometry of a binary link through compounding, see “Try It: Create a Compound Geometry” on page 1-19.

Negative Inertia as Subtraction

There is no requirement in the Simscape Multibody environment that the inertia parameters be positive. This includes the mass and moments of inertia, both parameters that in the physical world are strictly positive. Negative inertias enable you to model compound inertias with hollow sections by subtraction and are therefore useful in certain models.

The binary link again serves as an example. You can represent the link as a single piece without holes using one block, and subtract from its ends the inertias of the holes using additional blocks. As before, you must use rigid transforms to properly position the inertia reference frames relative to each other.

For an example, showing how to specify an inertia by compounding, see “Try It: Create a Compound Inertia” on page 1-23.

See Also

More About

- “Modeling Bodies” on page 1-5
- “Compounding Body Elements” on page 1-19
- “Specifying Custom Inertias” on page 1-73

- “Specifying Variable Inertias” on page 1-84

Specifying Custom Inertias

In this section...
“Key Inertia Conventions” on page 1-73
“Inertia Matrix Definitions” on page 1-73
“CAD as an Inertia Data Source” on page 1-78
“Automatic Inertia Calculation” on page 1-82

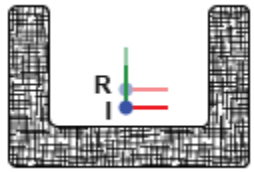
Key Inertia Conventions

Simscape Multibody software adopts a number of conventions in its inertia definitions. Take note of these as they may impact your inertia calculations if you perform them manually. The conventions may also bear on what additional processing, if any, your inertia data needs—for example, if it was sourced from a CAD application or other third-party software. In particular:

- The products of inertia are defined using a negated convention, one with a minus sign inserted, explicitly, in the mass integrals. An alternate convention exists in which a minus sign does not accompany the mass integrals. Recall that the products of inertia are the off-diagonal elements of the inertia matrix.
- The center of mass is defined with respect to the local reference frame of the block. In solids with imported CAD shapes, this frame is generally that assumed by your CAD application in its inertia calculations. It is possible, however, to modify a solid geometry file so that the two frames no longer match.

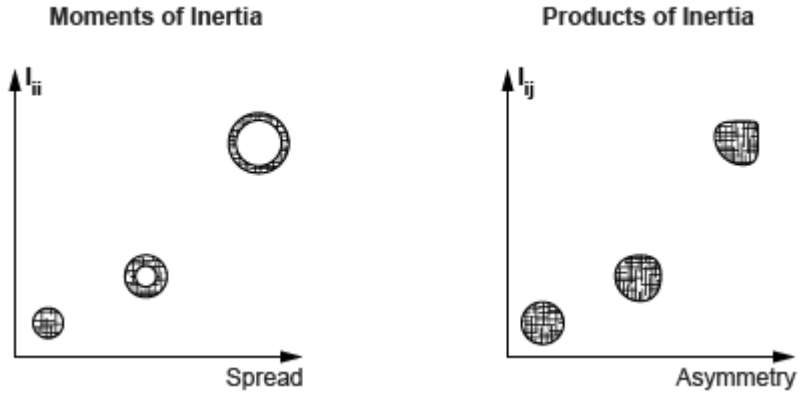
Inertia Matrix Definitions

The inertia matrix captures the spatial distribution of matter about a local frame referred to here as the inertia frame of resolution. This frame is labeled **I** in the figure. Its axes are parallel to those of the local reference frame, associated with frame port **R** and correspondingly labeled **R**. However, its origin coincides instead with the local center of mass.



The inertia matrix comprises the moments and products of inertia. The moments of inertia occupy the diagonal matrix positions and measure the dispersion, or spread, of the mass distribution about the axes of the inertia frame of resolution. The greater the spread about an axis, the larger the moment of inertia corresponding to that axis.

The products of inertia occupy the off-diagonal positions and measure the asymmetry of the mass distribution with respect to the planes of the inertia frame of resolution. The greater the asymmetry about a plane, the larger the products of inertia associated with any axis in that plane. The figure illustrates these relationships.



The Inertia Equations

The matrix is symmetric with respect to the main diagonal line: off-diagonal elements whose indices are reciprocals of each other share the same value. This constraint reduces the number of unique products of inertia from the original six (all those in off-diagonal positions) to the three that you must specify in a block (those with a unique combination of indices):

- $I_{yz} = I_{zy}$

- $I_{zx} = I_{xz}$
- $I_{xy} = I_{yx}$

The products of inertia, I_{ij}, are defined using the prevalent, negated, convention adopted by a number of CAD applications:

- $I_{yz} = -\int_V (yz)\,\rho dv$
- $I_{zx} = -\int_V (zx)\,\rho dv$
- $I_{xy} = -\int_V (xy)\,\rho dv$

where ρ is mass density, v is volume, and V is the total volume of integration. The x, y, and z coordinates are the Cartesian components of the distance vector spanning from the center of mass to an infinitesimal element of mass ρdv. The moments of inertia, I_{ii}, are similarly defined:

- $I_{xx} = \int_V \left(y^2 + z^2\right)\rho dv$
- $I_{yy} = \int_V \left(z^2 + x^2\right)\rho dv$
- $I_{zz} = \int_V \left(x^2 + y^2\right)\rho dv$

When applied to simple shapes such as cylindrical shells and rectangular beams, these definitions give rise to well-known algebraic equations that are often published in standard engineering tables. You can reference such tables when specifying the inertia parameters explicitly. The complete inertia matrix, according to the Simscape Multibody convention, is:

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

Principal Axes of Inertia

The moments of inertia are by definition positive numbers. However, the products of inertia can be either positive, negative, or zero. They are zero if the axes of the inertia frame of resolution happen to coincide with the principal axes of inertia. The moments of inertia are then called the principal moments of inertia and the inertia matrix is said to be *diagonal*:

$$I = \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix}$$

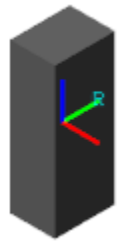
The number of nontrivial inertia matrix elements that you must specify is in this case reduced to three—the principal moments of inertia. For this reason, the principal axes of inertia can be a convenient frame in which to specify the inertia matrix elements. This is the inertia frame of resolution assumed in the highly symmetrical preset shapes of the Solid block.

The same, however, is not generally true of `General Extrusion` or `Revolution` solid shapes, nor is it of those imported via STEP or STL files. In `General Extrusion` and `Revolution` shapes, the frame placement depends closely on how you define the geometrical cross-sections. In imported shapes, it depends on how, relative to the local zero coordinate, the part geometries were modeled.

As a best practice, always consider the placement of the inertia frame of resolution when specifying the elements of the inertia matrix explicitly, particularly when using the Solid block. The frame position is always that of the center of mass, but its orientation relative to a solid geometry, when using the Solid bock, may not always coincide with the principal axes of inertia.

Try It: Specify a Custom Inertia

Consider the rectangular beam shown in the figure. Determine its mass, center of mass, moments of inertia, and products of inertia. Specify the calculated parameters explicitly in a Solid block using a `Custom` inertia parameterization.



Material and Dimensions

Assume a construction of aluminum and a corresponding mass density of 0.09754 lbm/in^3. Use the beam dimensions:

- Width x = 3 in
- Height y = 4 in
- Length z = 10 in

Prepare the Beam Model

Add a Solid block to a Simscape Multibody model. In the Solid block dialog box, specify the beam geometry: set the **Geometry > Shape** parameter to `Brick` and the **Geometry > Dimensions** parameter to `[2 3 10] in`. This array corresponds to the beam dimensions [x y z].

The geometry type affects the placement of the local reference frame (**R**) and therefore the inertia calculations themselves. In the visualization toolstrip, click the **Toggle visibility of frames** button. Frame **R** is located at the center of mass and its axes are parallel to the beam dimensions (x, y, and z).

Specify the Inertia Parameters

Calculate the inertia parameters from the density and dimensions of the beam. Then, specify the calculated values in the **Inertia** section of the Solid block parameters:

- **Mass** — Product of mass density (ρ) and volume ($x \cdot y \cdot z$):
  $m = \rho(x \cdot y \cdot z) = 11.7\,\text{lb}$
- **Center of Mass** — Centroid coordinates with respect to the local reference frame (**R**):
  $[\bar{x}, \bar{y}, \bar{z}] = [0, 0, 0]$

- **Moments of Inertia** — From standard expressions with respect to the inertia frame of resolution (**I**):

  $$\left[I_{xx}, I_{yy}, I_{zz}\right] = \frac{m}{12}\left[\left(y^2 + z^2\right), \left(z^2 + x^2\right), \left(x^2 + y^2\right)\right] = [113.1, 106.3, 24.4] \text{ lbm*in^2}$$

- **Products of Inertia** — From symmetry with respect to the inertia frame of resolution (**I**):

  $$\left[I_{yz}, I_{zx}, I_{xy}\right] = [0, 0, 0]$$

CAD as an Inertia Data Source

CAD applications often provide the inertia data for your part models. Examples include SolidWorks software, through its Mass Properties tool, and Onshape software, through its version of the same tool. You can reference this data and specify it manually in the Simscape Multibody environment.

Alternate Inertia Conventions

Some CAD applications, SolidWorks among them, use an alternate inertia convention to define the elements of the inertia matrix. This convention removes the minus sign from the product-of-inertia definitions. The I_{yz} product of inertia, for example, becomes:

$$I_{yz} = \int_V (yz)\,\rho dv$$

If your source of inertia data adopts this convention, you must explicitly negate the products of inertia before specifying their values in the Simscape Multibody environment. As an example, consider a SolidWorks inertia matrix given as:

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix}$$

To correctly specify the matrix elements in the Simscape Multibody environment, you must treat them as follows:

$$I = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{pmatrix}$$

CAD Import as an Alternative

Rather than reference the inertia data in a CAD assembly model, you can import that model into the Simscape Multibody environment. CAD import is based on the smimport function, which parses a multibody description file in XML format and generates an equivalent block diagram with all block parameters prespecified—inertia parameters included.

You must export your CAD model in a valid XML format, meaning one that conforms to the Simscape Multibody XML schema, before you can import it. This option may suit you only if you have a complete CAD assembly model. For individual CAD parts, use the STEP file import feature of the Solid block and set the **Inertia** > **Type** parameter to `Calculate from Geometry`.

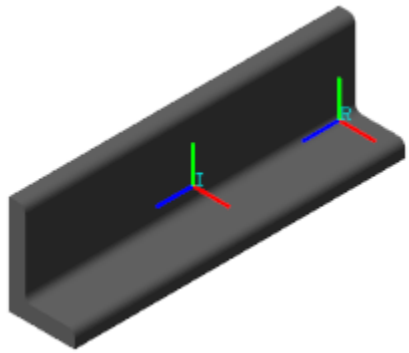
For more information, see “CAD Translation” on page 6-2.

Try It: Reference a SolidWorks Model

Determine the inertia parameters for the L-beam shape shown in the figure. Then, specify them explicitly in a Solid block by setting the inertia parameterization to `Custom`. Use the mass properties data provided in this example for a SolidWorks model of the beam.

Open the Solid Model

At the MATLAB command prompt, enter `smdoc_lbeam_inertia`. A simple model opens with a Solid block representing the L-beam solid. Open the Solid block and explore its **Geometry** parameters. The beam geometry is imported from a STEP file previously exported from a SolidWorks model. This geometry is shown in the figure.



In the visualization pane, click the **Toggle visibility of frames** button. The visualization pane shows two frames, one labeled **R** and the other **I**.

Frame **R** is the local reference frame of the solid. It coincides with what SolidWorks users refer to as the *output coordinate system* of the part model. This frame is located at the lower corner of the L-shape on one of the two longitudinal ends of the beam. You must specify the center of mass relative to this frame.

Frame **I** is a custom solid frame included for your convenience. This frame coincides with the inertia frame of resolution. Its origin is at the center of mass and its axes are parallel to those of the local reference frame. You must specify the moments and products of inertia relative to this frame.

Review the SolidWorks Data

The SolidWorks model provides for the L-beam part the following mass properties data:

```
Mass properties of l_beam_solid
     Configuration: Default
     Coordinate system: -- default --

Density = 0.10 pounds per cubic inch
Mass = 2.19 pounds
Volume = 22.41 cubic inches
Surface area = 101.91 square inches

Center of mass: ( inches )
    X = 0.58
    Y = 1.08
    Z = 5.00
```

```
Principal axes of inertia and principal moments of inertia:
( pounds * square inches )
Taken at the center of mass.
     Ix = ( 0.00,  0.00,  1.00)       Px = 2.49
     Iy = ( 0.38, -0.92,  0.00)       Py = 18.65
     Iz = ( 0.92,  0.38,  0.00)       Pz = 20.35

Moments of inertia: ( pounds * square inches )
Taken at the center of mass and aligned with
the output coordinate system.
    Lxx = 20.10    Lxy = -0.60    Lxz = 0.00
    Lyx = -0.60    Lyy = 18.89    Lyz = 0.00
    Lzx = 0.00    Lzy = 0.00    Lzz = 2.49

Moments of inertia: ( pounds * square inches )
Taken at the output coordinate system.
    Ixx = 77.40    Ixy = 0.76    Ixz = 6.33
    Iyx = 0.76    Iyy = 74.39    Iyz = 11.79
    Izx = 6.33    Izy = 11.79    Izz = 5.76
```

The data includes the coordinates of the center of mass with respect to the "output coordinate system." This coordinate system coincides with the local reference frame (**R**) of the corresponding Simscape Multibody solid.

The data includes also a matrix with the moments and products of inertia "taken at the center of mass and aligned with the output coordinate system." This coordinate system coincides with the inertia frame of resolution (**I**) of the Simscape Multibody solid.

Specify the Inertia Parameters

Expand the **Inertia** parameters section of the Solid block dialog box. Then, change the inertia parameterization by setting the **Inertia** > **Type** parameter to `Custom`. The complete set of inertia parameters appears for you to specify.

1. Set the **Mass** parameter to `2.19 lb`. This is the mass corresponding to the density of aluminum.
2. Set the **Center of Mass** parameter to `[0.58 1.08 5.00] in`. These are the [*x y z*] coordinates of the center of mass shown in the SolidWorks report.
3. Set the **Moments of Inertia** parameter to `[20.10 18.89 2.49] lbm*in^2`. These are the [*Lxx Lyy Lzz*] moments of inertia shown in the SolidWorks report.
4. Set the **Products of Inertia** parameter to `[0 0 0.6] lbm*in^2`. These are the negated [*Lyz Lzx Lxy*] products of inertia shown in the SolidWorks report.

Automatic Inertia Calculation

The Solid block provides an option to automatically compute the majority of inertia parameters given a solid geometry. This option, available from the **Inertia** > **Type** drop-down list and by default on, requires you to specify only the geometry parameters and either the mass or mass density.

The block uses the geometry and mass parameters to compute the remaining inertia parameters—the center of mass, moments of inertia, and products of inertia—relative to the appropriate frame of reference. The calculations are based on the assumption of a mass density that is constant and uniform.

You can view the calculation results inside the Solid block, in an expandable section named **Display Inertia**. The center of mass is given relative to the local reference frame (**R**), and the moments and products of inertia relative to the inertia frame of resolution (**I**). These are the same frames relative to which you might specify these parameters.

Try It: Display the Calculated Inertia Results

Configure the Solid block of the `smdoc_lbeam_inertia` model to calculate the inertia parameters from the solid geometry and its mass density. Then, view the calculated parameters.

1. In the dialog box of the Solid block, switch the **Inertia** > **Type** parameter to `Calculate from Geometry`. A **Display Inertia** node appears below the **Density** parameter.
2. Set the **Density** parameter to `0.09754 lbm/in^2`. This value corresponds to a solid of aluminum construction and it is the same assumed in the SolidWorks data provided in "Try It: Use Inertia Data from SolidWorks."
3. Expand the **Display Inertia** node and click the **Update** button. The inertia parameters under **Display Inertia** are populated with the calculated values. Compare them to the values provided in the SolidWorks mass properties data.

Derived Values		Update
Mass	+9.916786e-01	kg
Center of Mass	[+1.469190e-02, +2.734326e-02, +1.270000e-...	m
Moments of Ine...	[+5.871301e-03, +5.518708e-03, +7.268190e-...	kg*m^2
Products of Iner...	[+8.673617e-19, -2.168404e-19, +1.756705e-04]	kg*m^2

See Also

More About

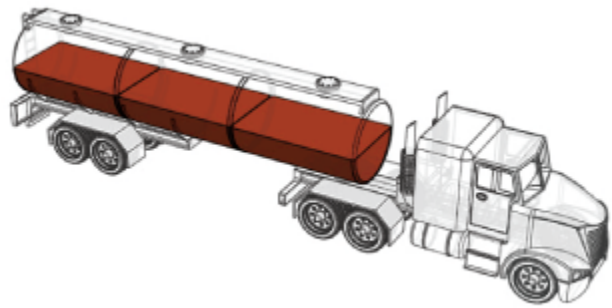
- “Modeling Bodies” on page 1-5
- “Compounding Body Elements” on page 1-19
- “Specifying Variable Inertias” on page 1-84

Specifying Variable Inertias

In this section...
“Modeling Variable Inertias” on page 1-84
“Visualizing Variable Inertias” on page 1-85
“Modeling Body Interactions” on page 1-85
“Model a Variable-Mass Oscillator” on page 1-86

Modeling Variable Inertias

A variable inertia is a mass element whose mass, center of mass, or inertia tensor can vary through time. Variable inertias include the scooped contents of a backhoe bucket, the moving occupants of a boom manlift, and the sloshing fluid load of a decelerating tank truck. You model a variable inertia using the General Variable Mass block from the **Body Elements > Variable Mass** library. This block accepts the various inertial properties as constants or variables. Physical signal ports provide the means to specify the variable properties.



A Fluid Load as a Variable Inertia

Specifying the Variable Inputs

You can specify your variable inputs using Simscape or Simulink blocks. You must convert any Simulink signals into physical signals using the Simulink-PS Converter block. Avoid sudden changes as these can increase model stiffness and slow down simulation. Ensure that the signal dimensions agree with the ports:

- Scalar for mass (port **m**)
- Three-element vector for center of mass (port **com**)
- Nine-element matrix for inertia tensor (port **I**)

Visualizing Variable Inertias

Variable inertias associated with General Variable Mass blocks have no geometry. You must visualize these inertias as graphical markers or as equivalent inertia ellipsoids. The ellipsoid dimensions and geometry center vary with mass, center of mass, and inertia tensor, lending the ellipsoids to more informative model visualizations. Inertia markers are shown on model update and during simulation. Variable inertia ellipsoids are shown during simulation only. The figure shows an inertia visualization representing a fluid load carried by a tank truck.

Equivalent Inertia Ellipsoid Visualization

Modeling Body Interactions

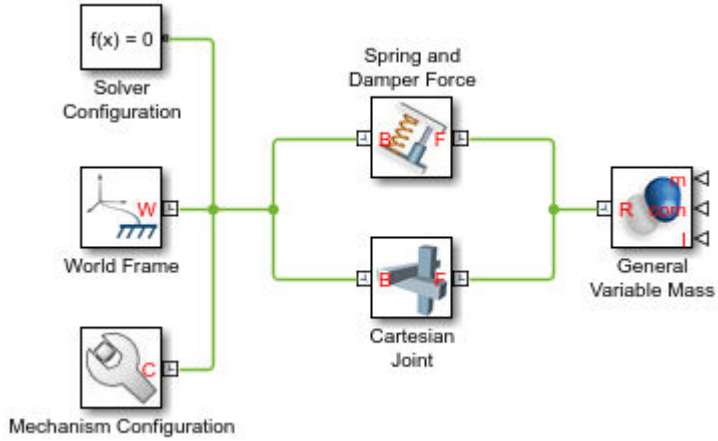
The General Variable Mass block captures inertial effects only. Any interactions between variable inertias and other model components must be modeled explicitly. Examples of interactions include contact forces between the fluid load of a tank truck and the surrounding enclosure. They include also the change in momentum obtained by expelling the combustion products of a propulsion system—for example, in a marine or space vehicle. Use other Simscape Multibody, Simscape, and Simulink blocks to capture interactions like these.

Model a Variable-Mass Oscillator

Create a simple model of a mass-spring system to simulate under constant-mass and variable-mass conditions. The model uses a General Variable Mass block to represent a container onto which a load of sand is progressively dropped. A Cartesian Joint block provides the variable-mass body with three translational degrees of freedom, although only one—along the vertical *z*-axis—is relevant during simulation. A Spring and Damper Force block represents the spring element, which connects the variable-mass body to the World frame.

Create Block Diagram

1. At the MATLAB command prompt, enter `smnew`. The command opens a model template with commonly used Simscape Multibody blocks.
2. Add the following blocks to the model canvas:
   - General Variable Mass (**Body Elements > Variable Mass**)
   - Cartesian Joint (**Joints**)
   - Spring and Damper Force (**Forces and Torques**)
3. Connect the blocks as shown in the figure and delete the remaining blocks. Ensure that the joint block orientation is as shown, with the base frame port facing the World Frame block.



4 In the Spring and Damper Force block dialog box, set the **Natural Length** parameter to `0.2` m and the **Spring Stiffness** parameter to `10` N/m.

5 In the Cartesian Joint block dialog box, expand the **Z Prismatic Primitive (Pz)** area, select the **State Targets** > **Specify Position Target** check box, and set the **Value** parameter to `0.1` m.

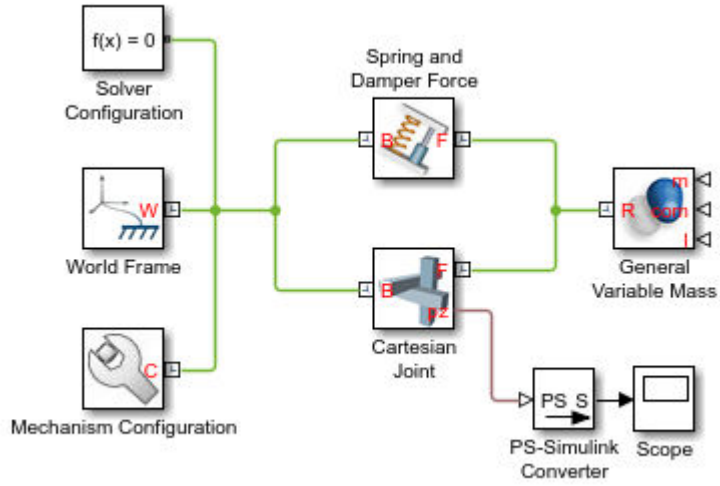
Add Position Sensing

1 In the Cartesian Joint block dialog box, expand the **Z Prismatic Primitive (Pz)** area and select the **Sensing** > **Position** check box. The block exposes a physical signal output port with the oscillator frame position.

2 Add the following blocks to the model canvas:

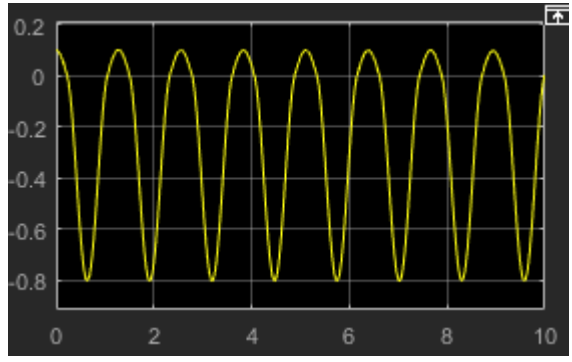
- PS-Simulink Converter (**Simscape** > **Foundation Library** > **Utilities**)
- Scope (**Simulink** > **Sinks**)

3 Connect the blocks as shown in the figure.

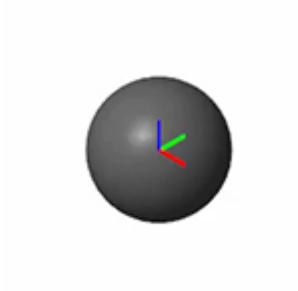


Simulate with Constant Mass

1. In the General Variable Mass block dialog box, set the **Type** parameter to `Custom`. This option enables you to model a variable mass distribution with rotational inertia.
2. Set the **Mass**, **Center of Mass**, and **Inertia Matrix** parameters to `Constant` and the **Mass > Value** parameter to `0.2` kg.
3. Run the simulation (**Simulation > Run**) and open the Scope block. The plot shows the position of the reference frame of the variable mass. Note that the oscillation frequency and amplitude stay constant throughout simulation.

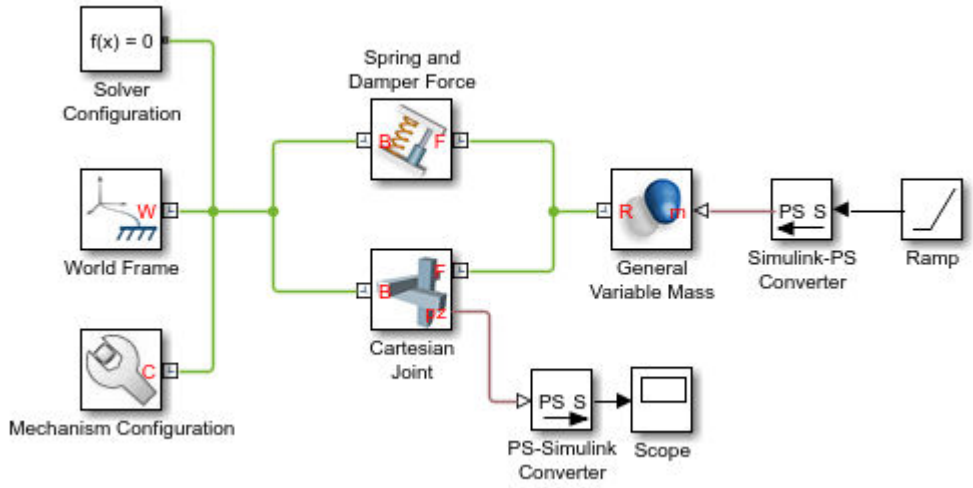


Mechanics Explorer opens with a 3-D animation of the model. The visualization comprises only an inertia ellipsoid—here a sphere due to the symmetry of the default inertia tensor used in the model. In the menu bar, select **View > Show Frames** to show all the frames in the model. Note that the ellipsoid dimensions stay constant during simulation, reflecting the constant inertial properties specified in the model.



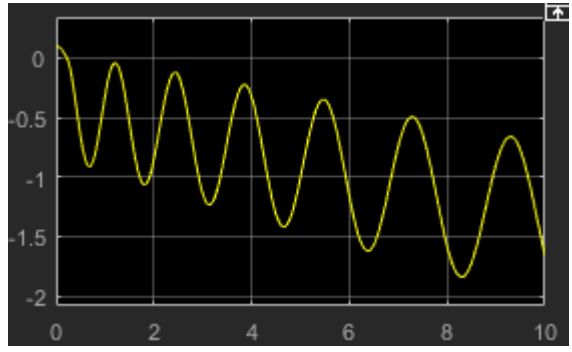
Simulate with Variable Mass

1. In the General Variable Mass block dialog box, set the **Mass** parameter to `Time-Varying`. The block exposes physical signal port **m**. You use this port to specify the variable mass.
2. Add these blocks to the model canvas:
   - Simulink-PS Converter (**Simscape > Foundation Library > Utilities**)
   - Ramp (**Simulink > Sources**)
3. Connect the blocks as shown in the figure.

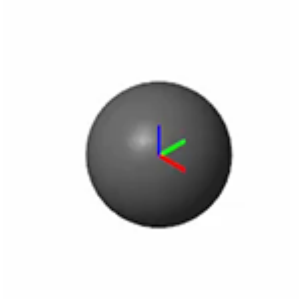


4 In the Ramp block dialog box, set the **Slope** parameter to `0.1` and the **Initial output** parameter to `0.2`. The Ramp signal is passed to the General Variable Mass block in the default Simscape units of mass, kg. The signal corresponds to a steadily increasing mass that starts at `0.2` kg and ends at `1.2` kg following a 10-second simulation.

5 Run the simulation and open the Scope block. The position plot shows a variable oscillation frequency and amplitude. The increasing mass causes the oscillation frequency to increase and the amplitude to decrease.



Mechanics Explorer updates the visualization results. Note that the ellipsoid dimensions decrease as the simulation progresses, reflecting their inverse proportionality to the variable mass.



See Also

More About

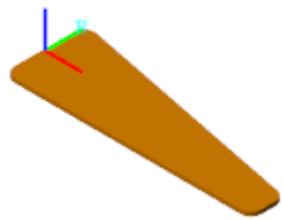
- “Modeling Bodies” on page 1-5
- “Compounding Body Elements” on page 1-19
- “Specifying Custom Inertias” on page 1-73
- “Specifying Variable Inertias” on page 1-84

Creating Custom Solid Frames

In this section...
“Solid Frames” on page 1-92
“Opening the Frame-Creation Interface” on page 1-93
“Geometry-Based Frame Placement” on page 1-93
“Primary and Secondary Axes” on page 1-94
“Try It: Create a Custom Solid Frame” on page 1-94

Solid Frames

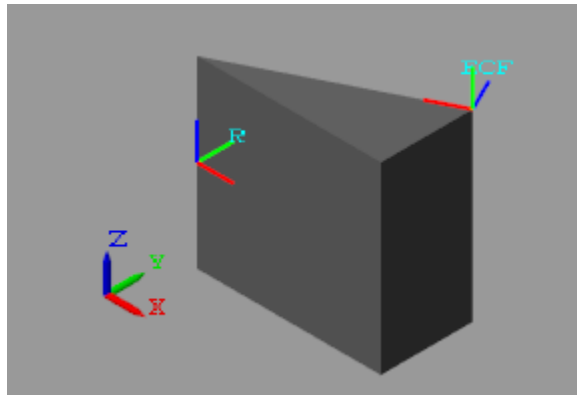
By default, the Solid block provides only a reference frame port, labeled **R**. In simple shapes, such as `Brick`, `Sphere`, and `Cylinder`, the reference frame origin coincides with the solid center of mass. The same is generally not true of the more sophisticated `General Extrusion` and `Revolution` shapes, nor is it of imported solid shapes.



Reference Frame of a Solid

In many applications, the reference frame of a solid is inadequate for connecting joints and constraints or for applying forces and torques. In such cases, you can create new frames external to the Solid block using the Rigid Transform block. This block enables you to define the new frame by specifying translation and rotation transforms numerically.

An alternative approach, and one that is often more intuitive, is to create new frames directly in the Solid block dialog box using the frame-creation interface. This interface enables you to define new frames interactively by aligning the frame origin and axes with geometric features such as planes, lines, and points.



Reference and Custom Frames of a Solid

Opening the Frame-Creation Interface

The frame-creation interface is accessible through the Solid block dialog box. To open the interface, in the **Frames** expandable area, select the **Create** button. If you change any of the block parameters, you must first update the solid visualization by selecting the **Update Visualization** button.

Geometry-Based Frame Placement

You can define frames based on geometric features of the solid or a choice of two frames —reference and principal inertia frames. The reference frame is the default frame of the solid. The principal inertia frame is one whose origin coincides with the center of mass and whose axes coincide with the principal axes of the solid.

Frames that you define by geometric features are specific to the shapes the features belong to. If you make the frame origin coincident with the vertex of a brick, the new frame is valid only for that particular brick shape. If you change shapes, you must edit or delete the new frame, as the geometric features it depends on no longer exist.

The frame-creation interface has three sections for specifying the following:

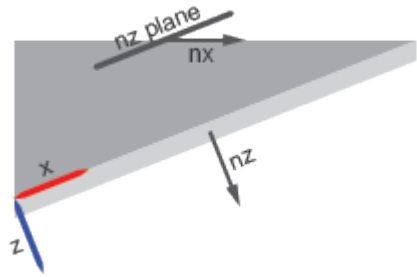
- Frame origin
- Primary axis

- Secondary axis

Primary and Secondary Axes

The primary axis constrains the possible directions of the remaining two axes. These axes must lie in the normal plane of the primary axis. If the axis or geometric feature used to define the secondary axis does not lie on this plane, the secondary axis is the projection of that axis or feature onto the normal plane.

The figure shows a top view of the three-sided extrusion you model in this tutorial. You align the primary axis (z) with the surface normal vector nz and the secondary axis (x) with the line vector nx. Because nx is not normal to the primary axis, the secondary axis is the projection of nx onto the normal plane of the primary axis.



Try It: Create a Custom Solid Frame

Create a frame on a solid using the frame-creation interface. The solid shape is a general extrusion with three unequal sides. This shape helps to demonstrate the difference between the primary and secondary frame axes that you specify in the frame creation interface.

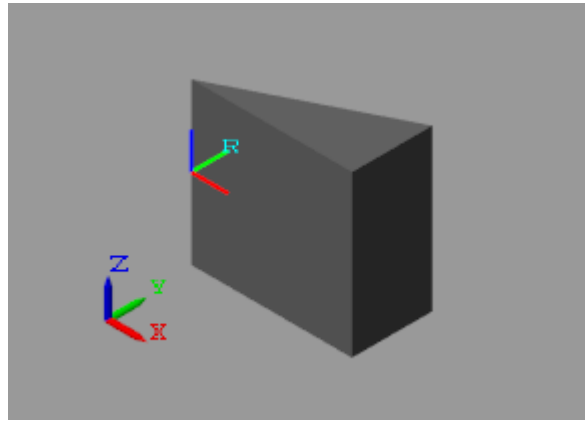
Specify the Solid Shape

1. From the Body Elements library, add one Solid block to a new model. The Solid block provides its own visualization utility. You do not need to update the block diagram to visualize the solid shape or its frames.
2. In the Solid block dialog box, specify these parameters.

Parameter	Value
Geometry > Shape	`General Extrusion`
Geometry > Cross-section	`[0,0;1,0;1,0.5]`

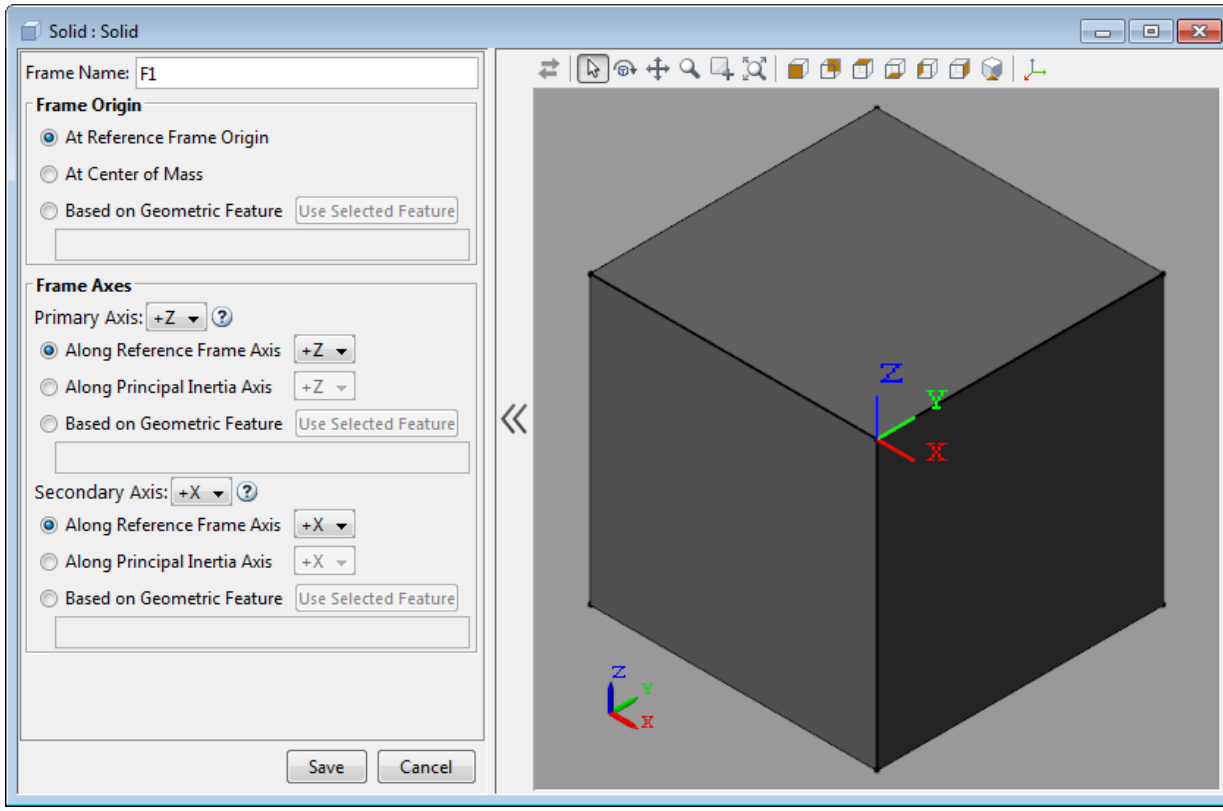
3 In the visualization toolstrip, select the **Update Visualization** button . The visualization pane updates with the three-sided extrusion that you specified.

4 Select the **Toggle visibility of frames** button. The visualization pane shows all the frames in the solid. At this point, the solid has a single frame—its reference frame. The reference frame origin coincides with the [0,0] cross-section coordinate in the midplane of the extrusion.



Create the Frame

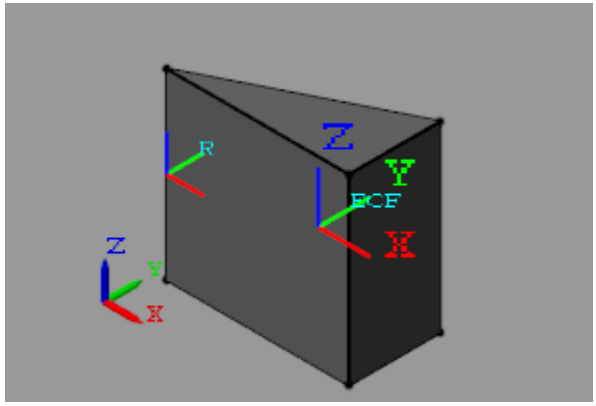
In the **Frames** expandable area of the Solid block dialog box, select the **Create** button . The Solid block opens the frame-creation interface.



In the **Frame Name** parameter, enter `ECF` (short for Extrusion Corner Frame). The frame name identifies the new frame in the Solid block visualization pane. It also appears as the frame port label on the Solid block.

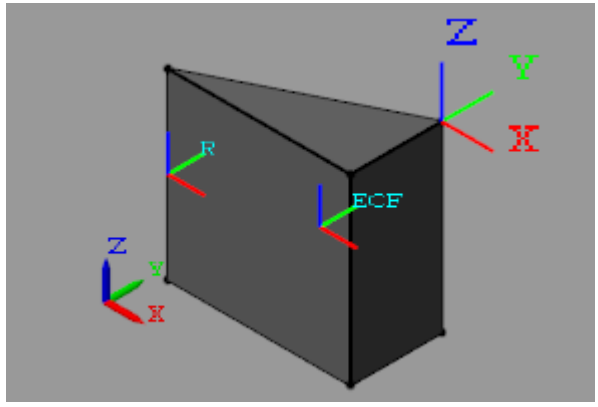
Specify the Frame Origin

Under **Frame Origin**, select **At Center of Mass**. The visualization pane updates with the new frame at the center of mass of the solid. This frame has the default frame orientation, that of the reference frame. The label ECF identifies the new frame.



Experiment with other frame origin locations. Define the origin location using one of the extrusion vertices.

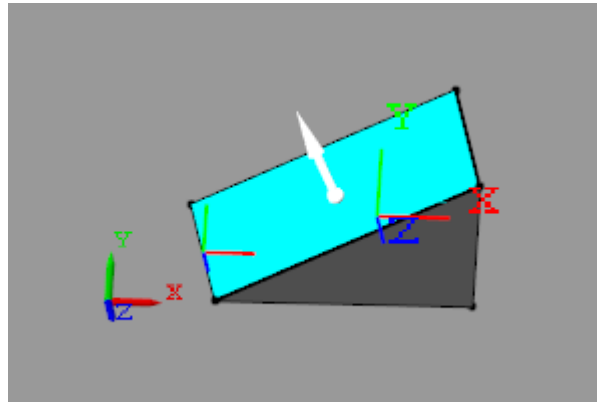
1. Under **Frame Origin**, select **Based on Geometric Feature**. This option enables you to select a point or the center of a plane or line as the frame origin.
2. In the visualization pane, select the vertex shown in the figure. The vertex is in the top plane of the extrusion. Ensure the view point is set to **Isometric**. In the **Frame Origin** area, ensure the vertex is named `Location of top point 3`.
3. Under **Frame Origin**, select the **Use Selected Feature** button. The visualization pane updates with the frame origin at the selected corner.



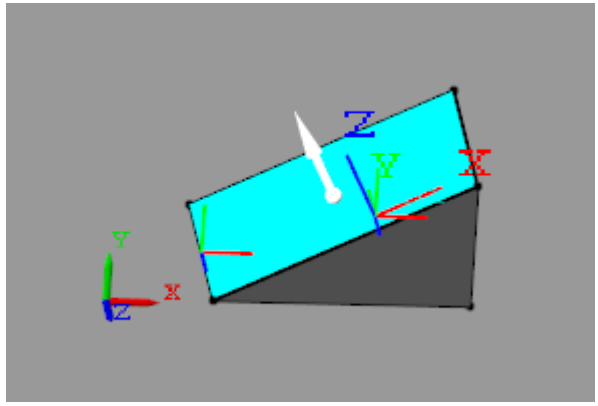
Specify the Primary Axis

The primary axis constrains the remaining two axes to lie on its normal plane. In this sense, the primary axis plays the dominant role in setting the orientation of the frame. Make the primary axis normal to the surface that contains the cross-section hypotenuse:

1. In the **Frame Axes** area under **Primary Axis**, select **Based on Geometric Feature**. The direction you specify in the next steps is that of the default primary axis, `+Z`.
2. In the visualization pane, rotate the solid and select the surface shown. The visualization pane highlights the surface and shows its normal vector. In the **Frame Axes** area under **Primary Axis**, ensure the surface is named `Surface normal of side surface 3`.



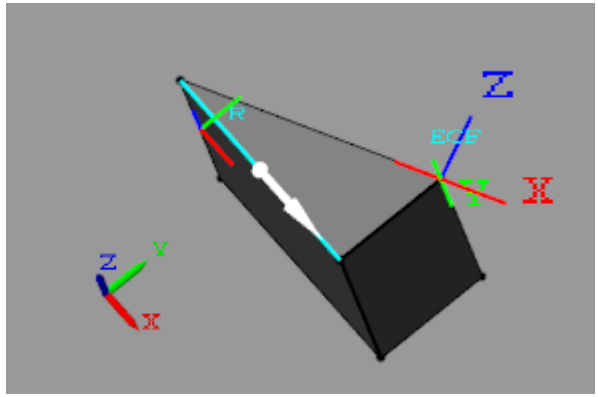
3. In the **Frame Axes** area under **Primary Axis**, select the **Use Selected Feature** button. The visualization pane updates with the z axis of the ECF frame, shown in dark blue, parallel to the normal vector of the selected surface.



Specify the Secondary Axis

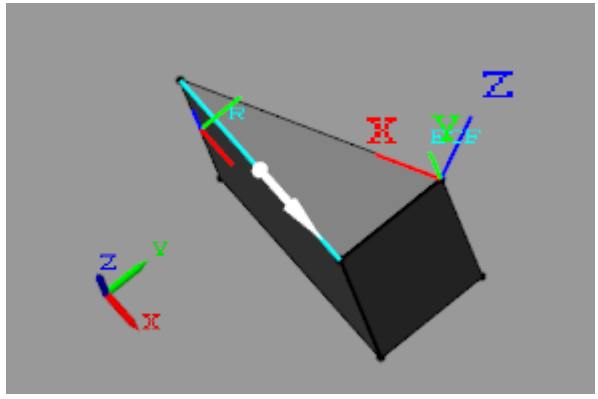
The secondary axis completes the definition of the new frame. In conjunction with the primary axis, the secondary axis fully constrains the direction of the third axis. The secondary axis is itself constrained to lie on the normal plane of the primary axis. To see the effects of this constraint, define the secondary axis based on a line not normal to the primary axis:

1. In the **Frame Axes** area, set the **Secondary Axis** parameter to `-X`. The direction you specify in the following steps is that of the `-X` axis.
2. In the **Frame Axes** area, under **Secondary Axis**, select **Based on Geometric Feature**.
3. In the visualization pane, rotate the solid and select the line shown. In the **Frame Axes** area, under **Secondary Axis**, ensure this line is named `Curve direction of top curve 1`.



4 Select the **Use Selected Feature** button. The visualization pane updates wit the x axis of the frame, shown in red, partially aligned with the selected line.

The two are not completely aligned as the selected line does not lie on the normal plane of the primary axis. The secondary axis is therefore the projection of the selected line onto the normal plane of the primary axis.

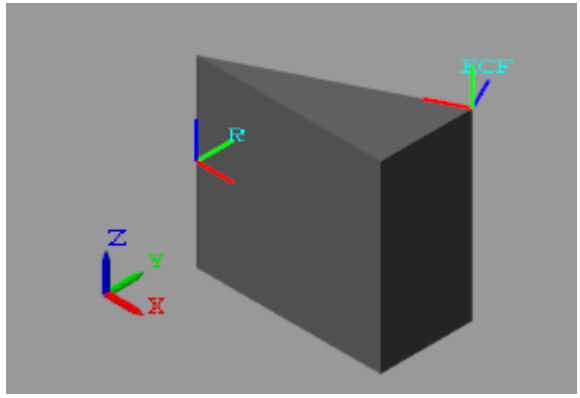


Save the New Frame

To save the frame you defined and commit it to the model:

1 Select the **Save** button. The visualization pane shows the solid with the final version of the frame you defined.

2 In the main interface of the Solid block dialog box, select **OK** or **Apply**. The Solid block commits the new frame to the model and exposes a new frame port labeled with the frame name you specified.



See Also

More About

- "Modeling Bodies" on page 1-5
- "Working with Frames" on page 1-25
- "Creating Connection Frames" on page 1-35
- "Visualize Simscape Multibody Frames" on page 5-32

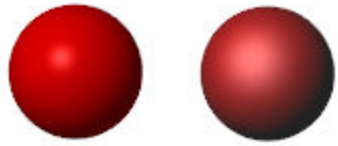
Manipulate the Color of a Solid

In this section...
“Visual Property Parameterizations” on page 1-102
“RGB and RGBA Vectors” on page 1-103
“Simple Visual Properties” on page 1-104
“Advanced Visual Properties” on page 1-104
“Adjust Solid Opacity” on page 1-105
“Adjust Highlight Color” on page 1-106
“Adjust Shadow Color” on page 1-106
“Adjust Self-Illumination Color” on page 1-107

Visual Property Parameterizations

The Solid block provides two parameterizations, `Simple` and `Advanced`, that you can use to specify solid visual properties. The `Simple` parameterization provides control over the solid color and opacity. The `Advanced` parameterization adds control over highlight, shadow, and self-illumination colors. You can use the `Advanced` visual properties to model emissive solids such as the sun and glossy solids such as polished metal parts.

You select a visual-property parameterization through the **Graphic > Visual Properties** block parameter. The figure contrasts the two parameterizations. On the left is a solid with `Simple` visual properties. On the right is the same solid with `Advanced` visual properties—including **Specular Color** and **Shininess** parameters, which impart to the solid a slight metallic sheen.



Try It

Set the visual-property parameterization of a solid to `Advanced`:

1. Add a Solid block to a new model canvas. The block provides its own visualization pane. You can use this pane to visualize the solid even if the model is not topologically valid.
2. Set the **Geometry > Shape** parameter to `Sphere`. In later examples, the curved spherical surface makes the specular highlights and ambient shadows easier to see.
3. In the Solid block dialog box, set the **Graphic > Visual Properties** parameter to `Advanced`. You set the visual-property parameterization individually for each solid.

RGB and RGBA Vectors

You can specify colors directly as [R,G,B] and [R,G,B,A] vectors on a normalized scale of 0–1. The R, G, and B elements provide the red, green, and blue contents of the specified color. The A element provides the color opacity—the degree to which the solid obstructs other components located behind it. Omitting the A element is equivalent to setting its value to `1`.

Try It

Identify the color and opacity given by the [R,G,B,A] vectors below:

- `[0,0,1,1]` — Denotes a solid color with no red or blue contents, maximum green content, and maximum opacity. The solid is bright green and fully opaque.

- `[1,0,0,0.5]` — Denotes a solid color with maximum red content, no green or blue contents, and partial opacity. The solid is bright red and transparent.

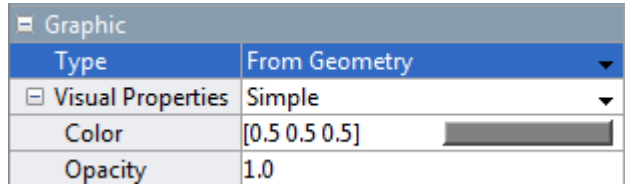
Simple Visual Properties

The `Simple` parameterization enables you to set the solid color and opacity. You can select a color using an interactive color picker or specify a color as an [R,G,B] vector. The **Color** parameter in the `Simple` parameterization is the same as the **Diffuse Color** parameter in the `Advanced` parameterization.

See It

See the parameters that comprise the `Simple` parameterization:

1. In the Solid block dialog box, set the **Visual Properties** parameter to `Simple`. This setting is the parameter default.
2. Expand the **Visual Properties** node. **Color** and **Opacity** appear as the active visual property parameters.



Advanced Visual Properties

The `Advanced` parameterization adds control over the highlight, shadow, and self-illumination colors as well as the size of the highlight areas. You must specify the colors directly as [R,G,B,A] vectors. The optional A element serves the same purpose as the **Opacity** parameter in the `Simple` parameterization.

See It

See the parameters that comprise the `Advanced` parameterization:

1. In the Solid block dialog box, set the **Visual Properties** parameter to `Advanced`.
2. Expand the **Visual Properties** node. The solid colors and shininess appear as the active visual property parameters.

Graphic	
Type	From Geometry
Visual Properties	Advanced
Diffuse Color	[0.5 0.5 0.5]
Specular Color	[0.5 0.5 0.5 1.0]
Ambient Color	[0.15 0.15 0.15 1.0]
Emissive Color	[0.0 0.0 0.0 1.0]
Shininess	75

Adjust Solid Opacity

You can make a solid transparent using either parameterization. If using the `Simple` parameterization, set the **Opacity** parameter to a value less than one. If using the `Advanced` parameterization, set the optional fourth element of the **Diffuse Color** [R,G,B,A] vector to a value less than one.

Try It

Model a transparent red solid using the `Advanced` visual-property parameterization:

1. Under the **Graphic** > **Visual Properties** node, change the **Diffuse Color** parameter to `[1,0,0,0.5]`.
2. In the visualization pane, click the button to refresh the solid visualization.

Delete the fourth vector element in the **Diffuse Color** parameter or set its value to `1` in order to make the solid opaque again.

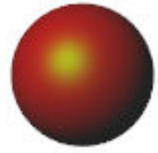
Adjust Highlight Color

You can control the size and color of specular highlights by adjusting the **Shininess** and **Specular Color** parameters in the `Advanced` visual-property parameterization. Lower the shininess value for large but soft highlights. Increase its value for small but sharp highlights.

Try It

Give the specular highlights a bright green hue. Set the **Diffuse Color** vector to `[1,0,0,1]` in order to make the solid opaque. Then:

1. In the **Graphic** > **Visual Properties** node, lower the **Shininess** parameter to `10`. This value increases the highlight size, making the specular color easier to see.
2. Change the **Specular Color** parameter to `[0,1,0,1]`. This vector sets the highlight color to bright green.
3. In the visualization pane, click the button to refresh the solid visualization. The specular color combines with the diffuse color to give highlight areas a green hue.



Adjust Shadow Color

You can control the color of shadow areas by adjusting the **Ambient Color** parameter in the `Advanced` visual-property parameterization.

Try It

Give the shadow areas a slight blue hue:

1. In the **Graphic** > **Visual Properties** node, set the **Ambient Color** parameter to `[0.15,0.15,0.3]`. This vector sets the shadow color to dark blue.
2. In the visualization pane, click the button to refresh the solid visualization. The ambient color combines with the diffuse color to give shadow areas a blue hue.

Adjust Self-Illumination Color

You can model self-illuminating solids such as the sun by adjusting the **Emissive Color** parameter in the `Advanced` visual-property parameterization.

Try It

Give the solid surface a red emissive color:

1. Under the **Graphic** > **Visual Properties** node, change the **Emissive Color** parameter to `[1,0,0,1]`.
2. In the visualization pane, click the button to refresh the solid visualization.

See Also

More About

- “Representing Solid Geometry” on page 1-42
- “Representing Solid Inertia” on page 1-65
- “Modeling Bodies” on page 1-5

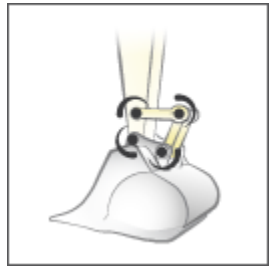
Multibody Systems

- "Multibody Assembly Workflow" on page 2-2
- "Modeling Joint Connections" on page 2-4
- "How Multibody Assembly Works" on page 2-9
- "Counting Degrees of Freedom" on page 2-15
- "Model an Open-Loop Kinematic Chain" on page 2-16
- "Model a Closed-Loop Kinematic Chain" on page 2-21
- "Troubleshoot an Assembly Error" on page 2-27
- "Modeling Gear Constraints" on page 2-36
- "Assemble a Gear Model" on page 2-44
- "Model a Compound Gear Train" on page 2-66
- "Constrain a Point to a Curve" on page 2-82

Multibody Assembly Workflow

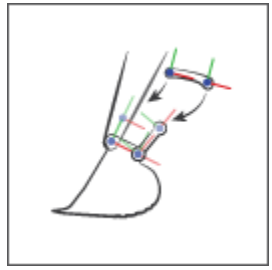
In this section...
“Study the Joints and Constraints to Model” on page 2-2
“Assemble Bodies Using Joints and Constraints” on page 2-2
“Guide Model Assembly” on page 2-3
“Verify Model Assembly” on page 2-3

Study the Joints and Constraints to Model



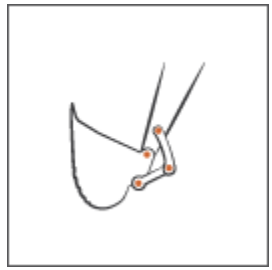
Identify the joints and constraints between the various bodies. Joints can be real, such as that between a piston and its case, or virtual, such as that between two planets.

Assemble Bodies Using Joints and Constraints



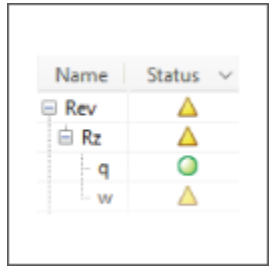
Model the degrees of freedom between bodies by connecting their frames through joints. You can further constrain these degrees of freedom through specialized constraints, such as those between gears. See “Assemble a Gear Model” on page 2-44 for an example.

Guide Model Assembly



Specify the state targets of the various joints. You can specify the desired position and velocity of a joint at time zero. If the state targets are valid and compatible, the joints assemble in the states specified. See the “Guide Assembly and Visualize Model” on page 2-24 section of “Model a Closed-Loop Kinematic Chain” on page 2-21 for an example.

Verify Model Assembly



Update the block diagram. Examine the model visualization for assembly issues. Open the Simscape Variable Viewer or the Simscape Multibody Model Report to see if all state targets have been satisfied. See the “Verify Model Assembly” on page 2-25 section of “Model a Closed-Loop Kinematic Chain” on page 2-21 for an example.

Modeling Joint Connections

In this section...
“Role of Joints in a Model” on page 2-4
“Joint Degrees of Freedom” on page 2-4
“Joint Primitives” on page 2-6
“Joint Inertia” on page 2-8

Role of Joints in a Model

Joints impose between bodies the primary kinematic constraints that determine how they can move relative to each other. A joint can be a physical connection, such as that between the case and shaft of a linear hydraulic actuator, or a virtual connection, such as that between the Earth and the moon. In Simscape Multibody, you model both connection types using Joint blocks.

Examples of physical and virtual connections between bodies

Gear and Constraint blocks too impose kinematic constraints between bodies. How are joint blocks different? While Gear and Constraint blocks are parameterized in terms of the DoFs they remove between bodies, Joint blocks are parameterized in terms of the DoFs they provide, through modules called joint primitives.

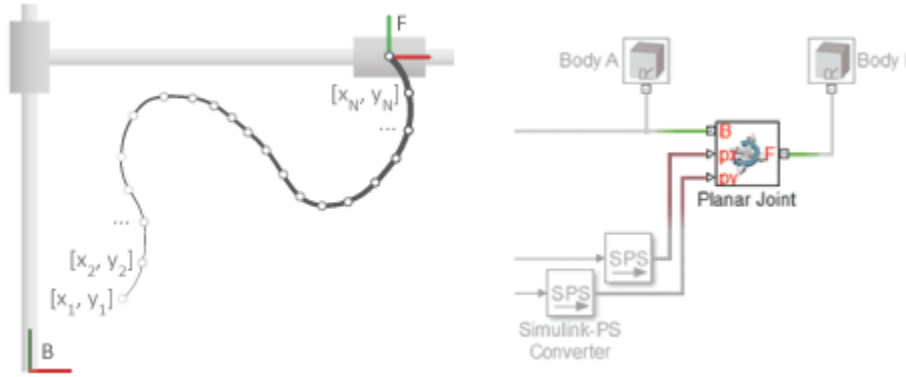
Joint Degrees of Freedom

Each Joint block connects exactly two bodies. Such a connection determines the maximum degrees of freedom, or DoFs, that the adjoining bodies can share. These DoFs range from zero in the Weld Joint block to six—three translational and three rotational—

in 6-DOF Joint and Bushing Joint blocks. Translation refers to a change in position and rotation to a change in orientation.

Joint DoFs are a measure of joint mobility. Precluding other constraints in a model, a joint with more DoFs allows greater freedom of motion between the adjoining bodies. Joint DoFs also have a mathematical interpretation. They are the minimum number of state variables needed to fully determine the configuration of a joint at each time step during simulation.

Consider a rectangular joint. This joint allows translation in a plane and it therefore has two translational DoFs—one for each spatial dimension. At each time step, the joint configuration is fully determined by two state variables, the position coordinates in the plane of motion [*x(t)*, *y(t)*]. This means, for example, that you can fully prescribe motion at this joint using two position input signals.



The table summarizes the DoFs that the various Joint blocks provide.

Joint Block	Translational DoFs	Rotational DoFs	Total DoFs
6-DOF Joint	3	3	6
Bearing Joint	1	3	4
Bushing Joint	3	3	6
Cartesian Joint	3	0	3
Constant Velocity Joint	0	2	2
Cylindrical Joint	1	1	2
Gimbal Joint	0	3	3
Leadscrew Joint	1 (coupled translational-rotational)		1
Pin Slot Joint	1	1	2
Planar Joint	2	1	3
Prismatic Joint	1	0	1
Rectangular Joint	2	0	2
Revolute Joint	0	1	1
Spherical Joint	0	3	3
Telescoping Joint	1	3	4
Universal Joint	0	2	2
Weld Joint	0	0	0

The actual DoFs at a joint are often fewer in number than the joint alone would allow. This happens when kinematic constraints elsewhere in the model limit the relative motion of the adjoining bodies. Such constraints can arise from gears in mesh, forbidden DoFs due to other joints in closed kinematic loops, and fixed distances and angles between bodies, among other factors.

Joint Primitives

Joint blocks are assortments of joint primitives, basic yet complete joints of various kinds you cannot decompose any further—at least without losing behavior such as the rotational-translational coupling of the lead screw joint. Joint primitives range in number from zero in the Weld Joint block to six in the Bushing Joint block. There are five joint primitives:

- Prismatic — Allows translation along a single standard axis (x, y, or z). Joint blocks can contain up to three prismatic joint primitives, one for each translational DoF. Prismatic primitives are labeled P*, where the asterisk denotes the axis of motion, e.g., Px, Py, or Pz.

- Revolute — Allows rotation about a single standard axis (x, y, or z). Joint blocks can contain up to three revolute joint primitives, one for each rotational DoF. Revolute primitives are labeled R*, where the asterisk denotes the axis of motion, e.g., Rx, Ry, or Rz.
- Spherical — Allows rotation about any 3-D axis, [*x*, *y*, *z*]. Joint blocks contain no more than one spherical primitive, and never in combination with revolute primitives. Spherical primitives are labeled S.
- Lead Screw Primitive — Allows coupled rotation and translation on a standard axis (e.g., z). This primitive converts between rotation at one end and translation at the other. Joint blocks contain no more than one lead screw primitive. Lead screw primitives are labeled LS*, where the asterisk denotes the axis of motion.
- Constant Velocity Joint — Allows rotation at constant velocity between intersecting though arbitrarily aligned shafts. Joint blocks contain no more than one constant velocity primitive. Constant velocity primitives are labeled CV.

The table summarizes the joint primitives and DoFs that the various Joint blocks provide.

Joint Block	Joint Primitives								
6-DOF Joint	Px	Py	Pz	.	.	.	S	.	.
Bearing Joint	.	.	Pz	Rx	Ry	Rz	.	.	.
Bushing Joint	Px	Py	Pz	Rx	Ry	Rz	.	.	.
Cartesian Joint	Px	Py	Pz	.	.	.	.	.	.
Constant Velocity Joint	.	.	.	.	.	.	.	CV	.
Cylindrical Joint	.	.	Pz	.	.	Rz	.	.	.
Gimbal Joint	.	.	.	Rx	Ry	Rz	.	.	.
Leadscrew Joint	.	.	.	.	.	.	.	.	LSz
Pin Slot Joint	Px	.	.	.	.	Rz	.	.	.
Planar Joint	Px	Py	.	.	.	Rz	.	.	.
Prismatic Joint	.	.	Pz	.	.	.	.	.	.
Rectangular Joint	Px	Py	.	.	.	.	.	.	.
Revolute Joint	.	.	.	.	.	Rz	.	.	.
Spherical Joint	.	.	.	.	.	.	S	.	.
Telescoping Joint	.	.	Pz	.	.	.	S	.	.
Universal Joint	.	.	.	Rx	Ry	.	.	.	.
Weld Joint	.	.	.	.	.	.	.	.	.

Why use Joint blocks with spherical primitives? Those with three revolute primitives are susceptible to gimbal lock—the natural but often undesired loss of one rotational DoF when any two rotation axes become aligned. Gimbal lock leads to simulation errors due to numerical singularities. Spherical primitives eliminate the risk of gimbal-lock errors by representing 3-D rotations using 4-D quantities known as quaternions.

Joint Inertia

Simscape Multibody joints are idealized. They differ from real joints in that they have no inertia—a suitable approximation in most models, where the impact of joint inertia on system dynamics is often negligible. This is the case, for example, in the constant-velocity joints of automobile driveline systems, where shaft inertia can dwarf joint inertia.

If joint inertia is important in your model, you can account for it using Solid or Inertia blocks. Connect the block reference frame ports to the appropriate joint frames and specify the joint inertial properties in the block dialog boxes. You can specify joint mass or density, products of inertia, moments of inertia, and center of mass. For more information on how to specify inertia, see “Representing Solid Inertia” on page 1-65.

How Multibody Assembly Works

In this section...
“Model Assembly” on page 2-9
“Connecting Joints” on page 2-9
“Orienting Joints” on page 2-10
“Guiding Assembly” on page 2-11
“Verifying Model Assembly” on page 2-12

Model Assembly

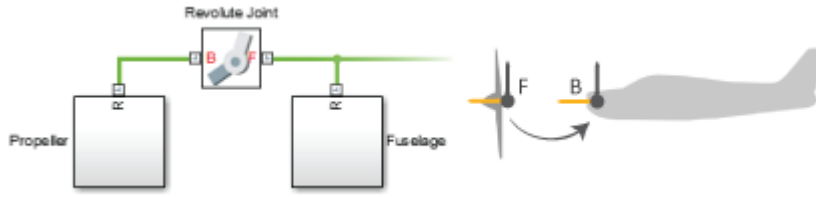
You model an articulated system by interconnecting bodies through joints and occasionally gears and other constraints. Bodies contribute their inertias to the model, while joints, gears, and constraints determine the relative degrees of freedom that exist between the bodies. You interconnect the two component types by linking frame ports on Joint, Gear, and Constraint blocks to frame ports on body subsystems.

Simscape Multibody automatically assembles your model when you update the block diagram—for example, by selecting **Simulation > Update Diagram** from the Simulink menu bar.

During model update, Simscape Multibody determines the initial states of joints—their positions and velocities—so that the resulting assembly satisfies all kinematic constraints in the model. This process occurs in two phases, with the assembly algorithm first computing the joint positions and then the joint velocities. The complete process is called model assembly.

Connecting Joints

Joints connect to bodies through frames. Each Joint block contains two frame ports, base (B) and follower (F), identifying the connection points in the adjoining bodies and the relative directions they can move in. When you connect these ports to frames in the body subsystems, you determine how the bodies themselves connect upon model assembly.



Joint Frames Identifying Connection Points and Rotation Axis of Aircraft Propeller

If a joint has no actuation and no sensing outputs, its frame ports are fully interchangeable. In this case, you can switch the bodies that the ports connect to without affecting model dynamics or joint sensing outputs. If the joint does have actuation inputs or sensing outputs, you may need to reverse the actuation or sensing signals to obtain the same dynamic behavior and simulation results.

To change the connection points of a joint, you must modify the connection frames in the adjoining body subsystems. You do this by specifying a translation transform using a Rigid Transform block. You can add new Rigid Transform blocks to the body subsystems or, if appropriate, change the translation transforms in existing Rigid Transform subsystems.

For more information on how Simscape Multibody software interprets frame ports, nodes, and lines, see “Working with Frames” on page 1-26.

Orienting Joints

To obtain the motion expected in a model, you must align its various joint motion axes properly. This means aligning the joints themselves as observed or anticipated in the real system. Misaligning the joint axes may lead to unexpected motion but it often leads to something more serious, such as a failure to assemble and simulate.

You can specify and change joint alignment by rotating the connection frames local to the adjoining body subsystems. For this purpose, you specify rotation transforms using Rigid Transform blocks, either by adding new blocks to the body subsystems or, if appropriate, by changing the rotation transforms in existing blocks within the subsystems.

Why change the orientation of joints through body subsystem frames? The primitives in a Joint block each have a predetermined motion axis, such as x or z. The axis definition is fixed and cannot be changed. Realigning the connection frames local to the adjoining

body subsystems provides a natural way to reorient joints while avoiding confusion over which axis a particular joint uses.

Guiding Assembly

Joints can start simulation from different states. For example, the crank joint of a crank-rocker linkage can start at any angle from 0° to 360°. As a result, during model assembly, Simscape Multibody must choose from many equally valid states. You can guide the states chosen by specifying state targets in the Joint block dialog boxes.

Crank-Slider Mechanism in Fully Extended and Fully Retracted Initial Configurations

State targets need not be exact values. If Simscape Multibody cannot achieve a state target exactly, it searches for the joint state nearest to the state target. For example, if you specify a position state target of 60° but the joint can only reach angles of 0° to 45°, Simscape Multibody attempts to assemble the joint at 45°.

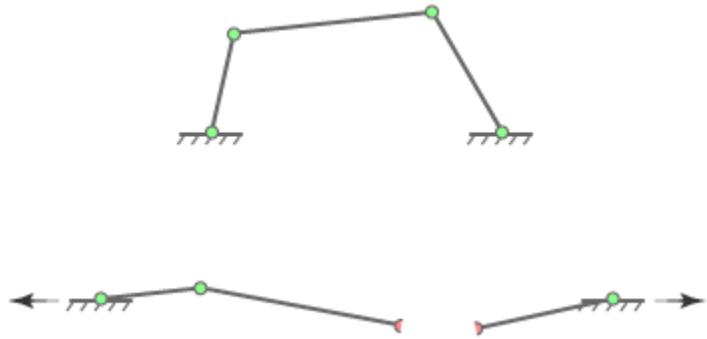
How close the actual joint state is to the state target depends on the kinematic constraints in your model, any conflicts with other state targets, and the state target priority level—a ranking that determines which of two state targets to satisfy if they prove to be mutually incompatible. You can set the priority level to `Low` or `High`.

Simscape Multibody first attempts to satisfy all state targets exactly. If a state target conflict arises, Simscape Multibody ignores the low-priority state targets and attempts to satisfy only the high-priority state targets. If a state target conflict still exists, Simscape Multibody ignores also the high-priority state targets and attempts to assemble the model in the nearest valid configuration.

You can specify state targets for all joints in an open kinematic chain. However, to avoid simulation errors, every closed chain must contain at least one joint without state targets.

Verifying Model Assembly

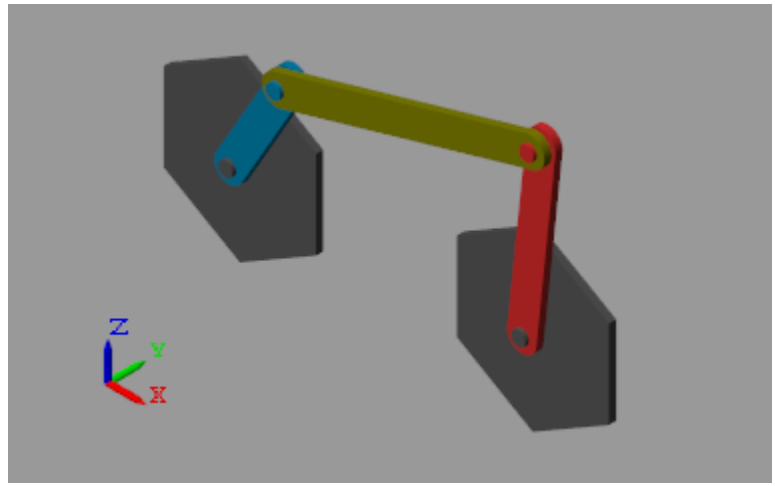
A model assembles successfully only if the connections between its bodies are congruous with each other. If in satisfying one kinematic constraint, Simscape Multibody must violate another kinematic constraint, the model is kinematically invalid and assembly fails. This happens, for example, when the ground link of a four-bar assembly exceeds the combined length of the remaining three links, preventing at least one joint from assembling.



Joint Assembly Failure in Four-Bar Linkage with Exceedingly Long Ground Link

To ensure that your model has assembled correctly, use these Simscape Multibody and Simscape utilities:

- Mechanics Explorer — Simscape Multibody visualization utility. Visually examine your model from different points of view to ensure that its bodies connect at the expected locations and with the proper orientations.



- Variable Viewer — Simscape state-reporting utility. Check the assembly status of individual joints and constraints and compare your state targets to the actual joint states achieved during assembly.

Name	Status	Priority	Target	Start	Unit
Base_Crank_Revolute					
Rz					
q		High	150.0	150.0	deg
w		High	-360.0	-360.0	deg/s
Base_Rocker_Revolute					
Rz					
q				3.0338	rad
w				-3.13757	rad/s
Connector_Rocker_Revolute					
Rz					
q				1.1814	rad
w				-4.35682	rad/s
Crank_Connector_Revolute					
Rz					
q		Low	-45.0	-43.86525496465046	deg
w				7.50244	rad/s

All high priority targets satisfied but some low priority targets not satisfied — Variables at start

- Statistics Viewer — Simscape metrics-reporting utility. Check, among other metrics, the degrees of freedom, number of joints, and number of constraints in your model.

Name	Value
3-D Multibody System	
Number of rigidly connected components (excluding ...	3
Number of joints (total)	4
Number of explicit tree joints	3
Number of implicit 6-DOF tree joints	0
Number of cut joints	1
Number of constraints	0
Number of tree degrees of freedom	3
Number of position constraint equations (total)	5
Number of position constraint equations (non-redund...	2
Number of mechanism degrees of freedom (minimum)	1
State vector size	8
Average kinematic loop length	4

Counting Degrees of Freedom

The number and types of joints, gears, and constraints in a mechanism partially determine its mobility—the total number of degrees of freedom, or DoFs, that the mechanism provides and therefore the minimum number of input variables needed to fully constrain its configuration. The mobility F of a mechanism with N bodies and j joints, each with f DoFs follows from expressions such as the Kutzbach criterion, which for a planar mechanism states:

$$F = 3(N-1) - \sum_{i=1}^{j}(3 - f_i)$$

Applying this criterion to a four-bar linkage, an assembly of four bodies ($n = 4$) and four joints ($j = 4$) with one rotational DoF each ($f_i = 1$), yields a mobility of one DoF—indicating that a single input variable suffices to fully control the linkage configuration. As mechanisms grow in complexity, manually calculating total DoFs becomes more time-consuming, so Simscape Multibody automatically computes them for you.

You can view the mechanism DoFs through the Simscape Statistics Viewer, shown below for the four-bar featured example. You open the Statistics Viewer from the Simulink Editor menu bar by selecting **Analysis > Simscape > Statistics Viewer**. Enter `sm_four_bar` at the MATLAB command prompt to open the four-bar model and view its DoFs through the Statistics Viewer.

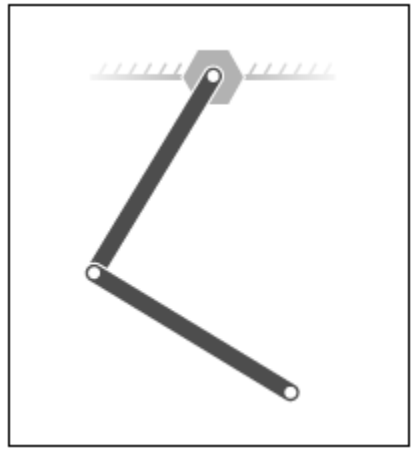
Name	Value
Number of implicit 6-DOF tree joints	0
Number of cut joints	1
Number of constraints	0
Number of tree degrees of freedom	3
Number of position constraint equations (total)	5
Number of position constraint equations (non-redundant)	2
Number of mechanism degrees of freedom (minimum)	1
State vector size	8
Average kinematic loop length	4

Model an Open-Loop Kinematic Chain

In this section...
“Model Overview” on page 2-16
“Build Model” on page 2-17
“Guide Model Assembly” on page 2-18
“Visualize Model and Check Assembly Status” on page 2-18
“Simulate Model” on page 2-20
“Open Reference Model” on page 2-20

Model Overview

This example shows how to model a double pendulum—a simple kinematic chain comprising two moving bodies connected in series via two revolute joints. A third body represents a mechanical ground and is rigidly connected to the inertial World frame. The custom `smdoc_compound_rigid_bodies` library provides the body subsystem blocks used in the example.



Revolute Joint blocks enable you to model the joints connecting adjacent bodies and help set their initial states. Simscape Multibody software satisfies a joint state target

precisely if it is kinematically valid and not in conflict with other state targets. A **Priority** parameter lets you specify which targets to attempt to satisfy first.

Build Model

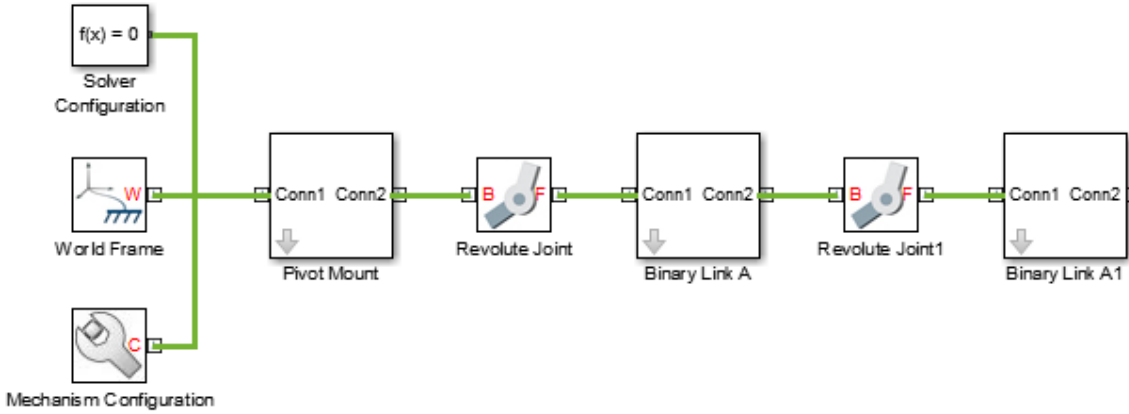
1. Start a new model.
2. Drag these blocks into the model. The two Revolute Joint blocks provide the double pendulum two rotational degrees of freedom.

Library	Block	Quantity
Simscape > Utilities	Solver Configuration	1
Simscape > Multibody > Utilities	Mechanism Configuration	1
Simscape > Multibody > Frames and Transforms	World Frame	1
Simscape > Multibody > Joints	Revolute Joint	2

3. At the MATLAB command prompt, enter `smdoc_compound_rigid_bodies`. A custom block library with the same name opens up.
4. Drag these custom blocks into the model. Each block represents a body in the double pendulum.

Block	Quantity
Pivot Mount	1
Binary Link A	2

5. Connect the blocks as shown in the figure.



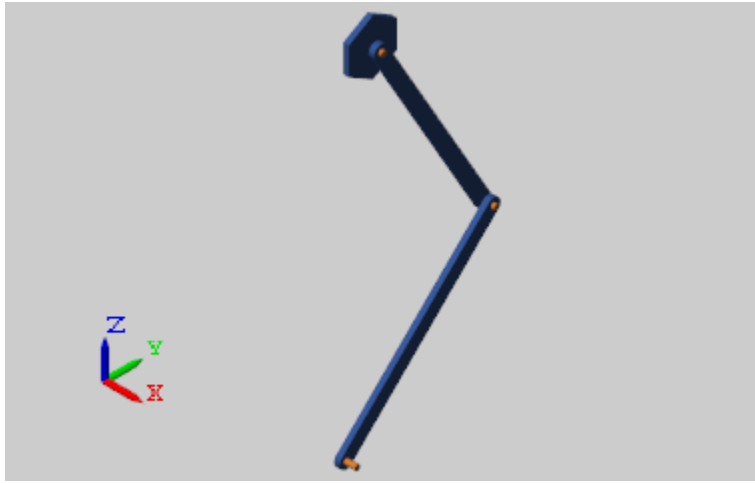
Guide Model Assembly

1. In the Revolute Joint block dialog boxes, select **State Targets** > **Specify Position Target**. You can now specify the desired starting positions of the two joints.
2. In **Value**, enter these joint angles.

Block Name	Value (degrees)
Revolute Joint	30
Revolute Joint1	-75

Visualize Model and Check Assembly Status

To visualize the model, update the block diagram. You can do this from the menu bar by selecting **Simulation** > **Update Diagram**. Mechanics Explorer opens with a 3-D view of the double pendulum assembly. Click the isometric view button to obtain the perspective in the figure.



To check the assembly status of the revolute joints, use the Model Report utility. You can open this utility from the Mechanics Explorer menu bar by selecting **Tools > Model Report**. The figure shows the assembly information for the double pendulum.

Model Report - double_pendulum_model

Assembly status:
Joints:
Constraints:

Joints | Constraints | Statistics

Joint	Asse...	Primit...	Position					Velocity				
			Actual	Specif...	Unit	Priority	Status	Actual	Specif...	Units	Priority	Status
Revolute_Joint		Rz	+30	+30	deg	High		+0		deg/s		
Revolute_Joint1		Rz	-75	-75	deg	High		+0		deg/s		

OK

Simulate Model

Run the simulation, e.g., by selecting **Simulation > Run**. Mechanics Explorer shows a 3-D animation of the double pendulum assembly. The assembly moves due to gravity, specified in the Mechanism Configuration block.

Open Reference Model

To see a complete model of the double pendulum assembly, at the MATLAB command prompt enter:

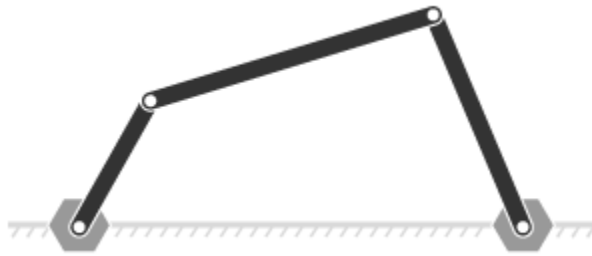
- `smdoc_double_pendulum`

Model a Closed-Loop Kinematic Chain

In this section...
“Model Overview” on page 2-21
“Build Model” on page 2-22
“Specify Block Parameters” on page 2-24
“Guide Assembly and Visualize Model” on page 2-24
“Verify Model Assembly” on page 2-25
“Simulate Model” on page 2-26
“Open Reference Model” on page 2-26

Model Overview

This example shows how to model a four bar—a closed kinematic chain comprising four bodies that connect through revolute joints. One of the links is fixed to the World frame and acts as a ground. This link is replaced here by two pivot mounts connected through a rigid translation transform. The custom `smdoc_compound_rigid_bodies` library provides the body subsystem blocks used in the example.



Revolute Joint blocks enable you to model the joints connecting adjacent bodies and help set their initial states. Simscape Multibody software satisfies a joint state target precisely if it is kinematically valid and not in conflict with other state targets. A **Priority** parameter lets you specify which targets to attempt to satisfy first.

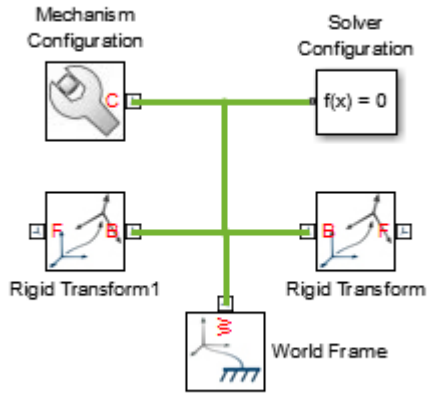
Build Model

To model the four-bar linkage:

1. Start a new model.
2. Drag these blocks to the model. The Rigid Transform blocks specify the distance between the two pivot mounts. This distance is the length of the implicit ground link.

Library	Block	Quantity
Simscape > Utilities	Solver Configuration	1
Simscape > Multibody > Utilities	Mechanism Configuration	1
Simscape > Multibody > Frames and Transforms	World Frame	1
Simscape > Multibody > Frames and Transforms	Rigid Transform	2

3. Connect and name the blocks as shown in the figure. The base frame ports of the Rigid Transform blocks must connect to the World Frame block.

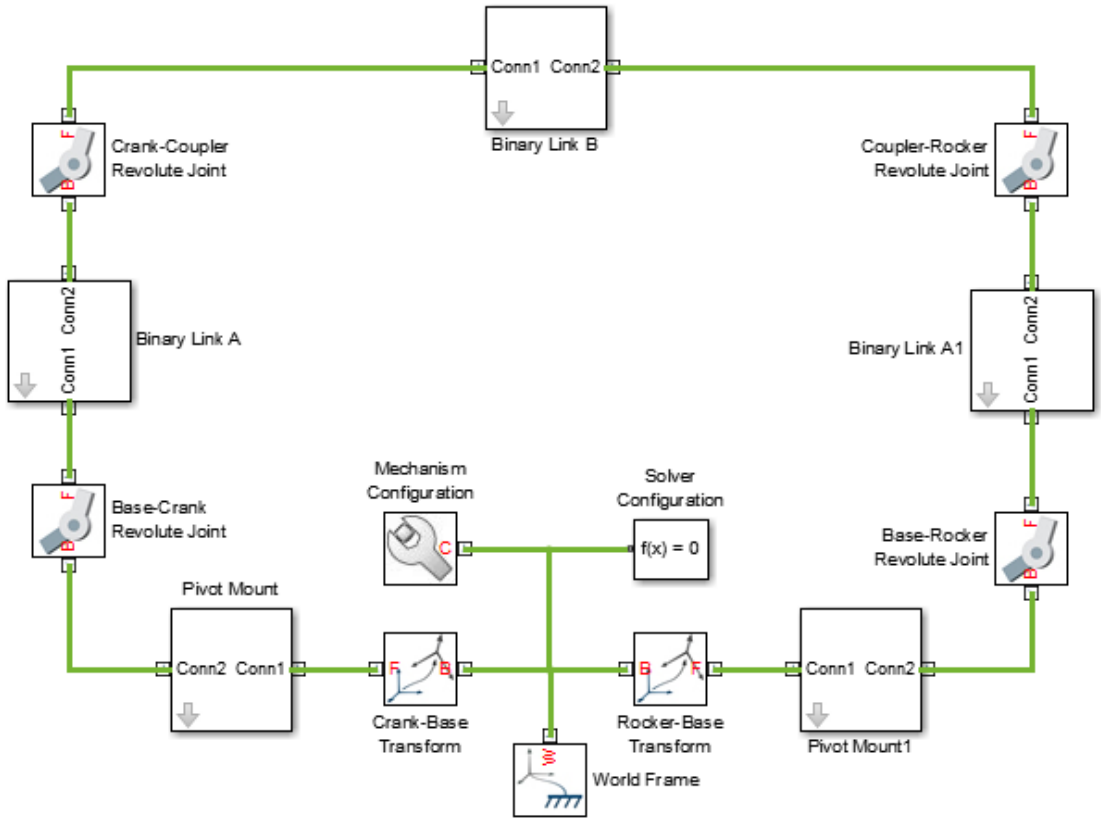


4. From the **Simscape > Multibody > Joints** library, drag four Revolute Joint blocks into the model.
5. At the MATLAB command prompt, enter `smdoc_compound_rigid_bodies`. A custom library with compound body subsystem blocks opens up.

6 From the `smdoc_compound_rigid_bodies` library, drag the following blocks.

Block	Quantity
Pivot Mount	2
Binary Link A	2
Binary Link B	1

7 Connect and name the blocks as shown in the figure. You must position the frame ports of the custom body subsystem blocks exactly as shown.



Specify Block Parameters

1 In the Rigid Transform block dialog boxes, specify the offset between the pivot mounts and the world frame. The pivot mounts are assumed to be symmetrically positioned about this frame.

Parameter	Crank-Base Transform	Rocker-Base Transform
Translation > Method	`Standard Axis`	`Standard Axis`
Translation > Axis	`-Y`	`+Y`
Translation > Offset	`15` in units of `cm`	`15` in units of `cm`

2 In each binary link block dialog box, specify the length parameter.

Block	Length (cm)
Binary Link A	`10`
Binary Link B	`35`
Binary Link A1	`20`

Guide Assembly and Visualize Model

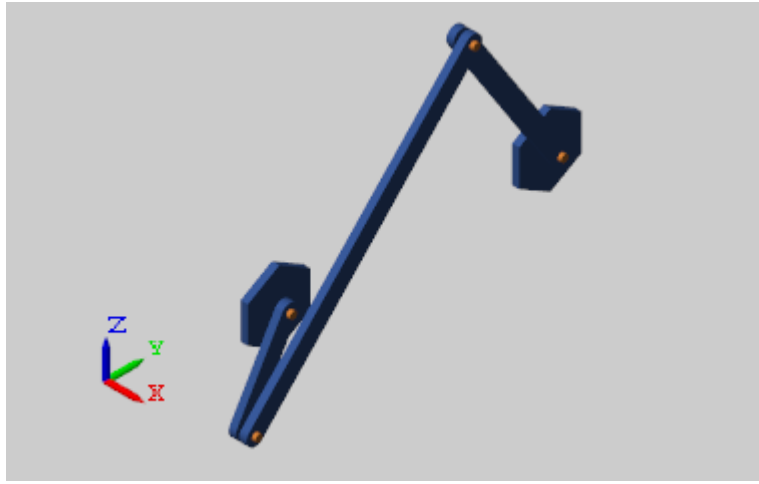
Guide model assembly by specifying the desired initial state for one or more joints in the model. To specify an initial angle of 30° for the Base-Crank joint:

1 In the Base-Crank Revolute Joint block dialog box, expand **State Targets** and select **Specify Position Target**.

2 In **Value**, enter `-30` and press **OK**.

3 In the menu bar, select **Simulation > Update Diagram**

Mechanics Explorer opens with a static display of the four-bar linkage in its initial configuration. If the joint state targets that you specified are valid and compatible, the initial configuration matches those state targets precisely.

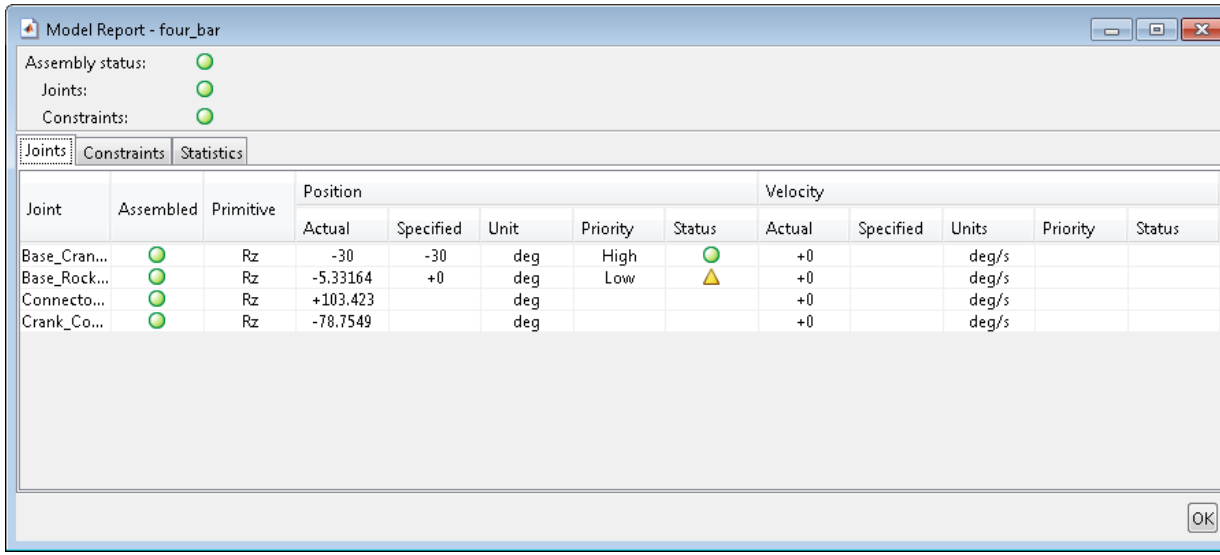


Verify Model Assembly

To check whether—and how precisely—your state targets were met, you can use the Simscape Variable Viewer or the Simscape Multibody Model Report.

To open the Simscape Variable Viewer, in the Simulink menu bar, select **Analysis > Simscape > Variable Viewer**. To open the Simscape Multibody Model Report, update the diagram and, in the Mechanics Explorer menu bar, select **Tools > Model Report**.

The figure shows a Model Report example. The yellow marker indicates that the Base-Rocker Revolute Joint state target was satisfied approximately only. The remaining green marker indicates that the remaining state target was satisfied precisely.



Simulate Model

Run the simulation, e.g., by selecting **Simulation > Run**. Mechanics Explorer shows a 3-D animation of the four bar assembly. The assembly moves due to gravity, specified in the Mechanism Configuration block.

Open Reference Model

To see a complete model of the four–bar assembly, at the MATLAB command prompt enter:

- `smdoc_four_bar`

Troubleshoot an Assembly Error

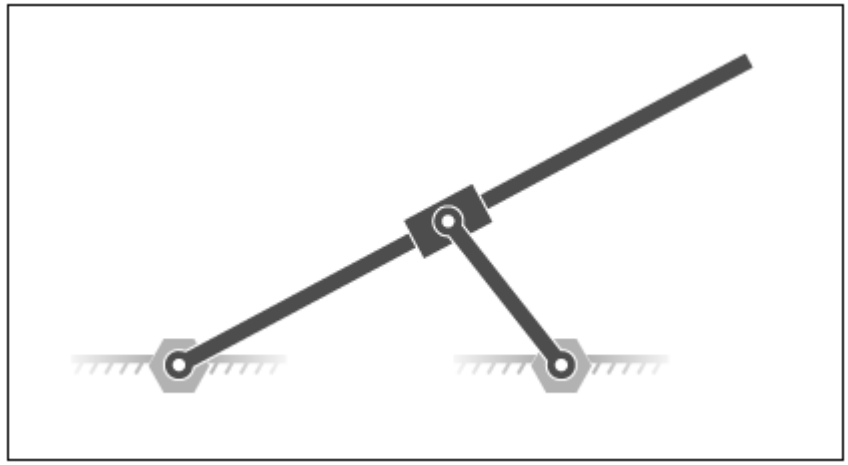
In this section...
“Model Overview” on page 2-27
“Explore Model” on page 2-28
“Update Model” on page 2-31
“Troubleshoot Assembly Error” on page 2-31
“Correct Assembly Error” on page 2-33
“Simulate Model” on page 2-34

Model Overview

In closed-loop systems, joints and constraints must be mutually compatible. For example, in a four-bar linkage, all revolute joints must spin about parallel axes. If one of the joints spins about a different axis, assembly fails and the model does not simulate.

To simplify the troubleshooting process, Simscape Multibody provides Model Report. This tool helps you pinpoint the joints and constraints that caused assembly to fail. Once you identify these joints and constraints, you can then determine which of their frames to correct—and how to correct them.

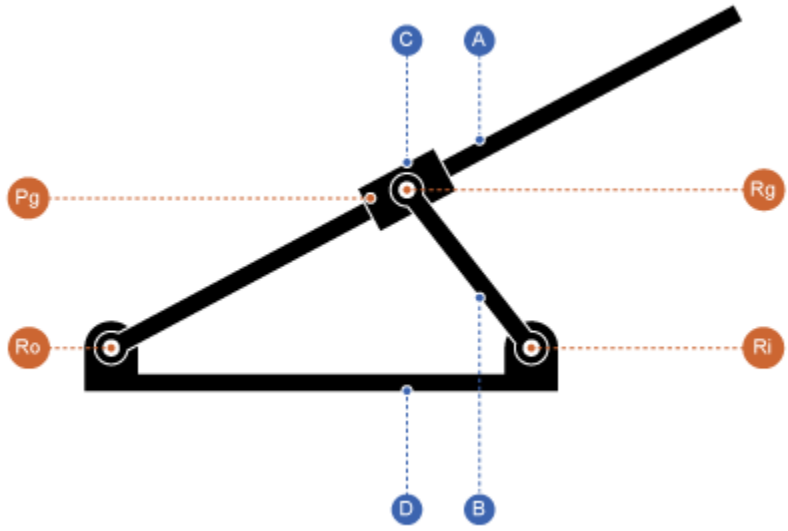
In this example, you identify the assembly error source in an aiming mechanism model using Model Report. Then, using Mechanics Explorer, you determine how to correct that error source. The `sm_dcrankaim_assembly_with_error` featured example provides the basis for this example.



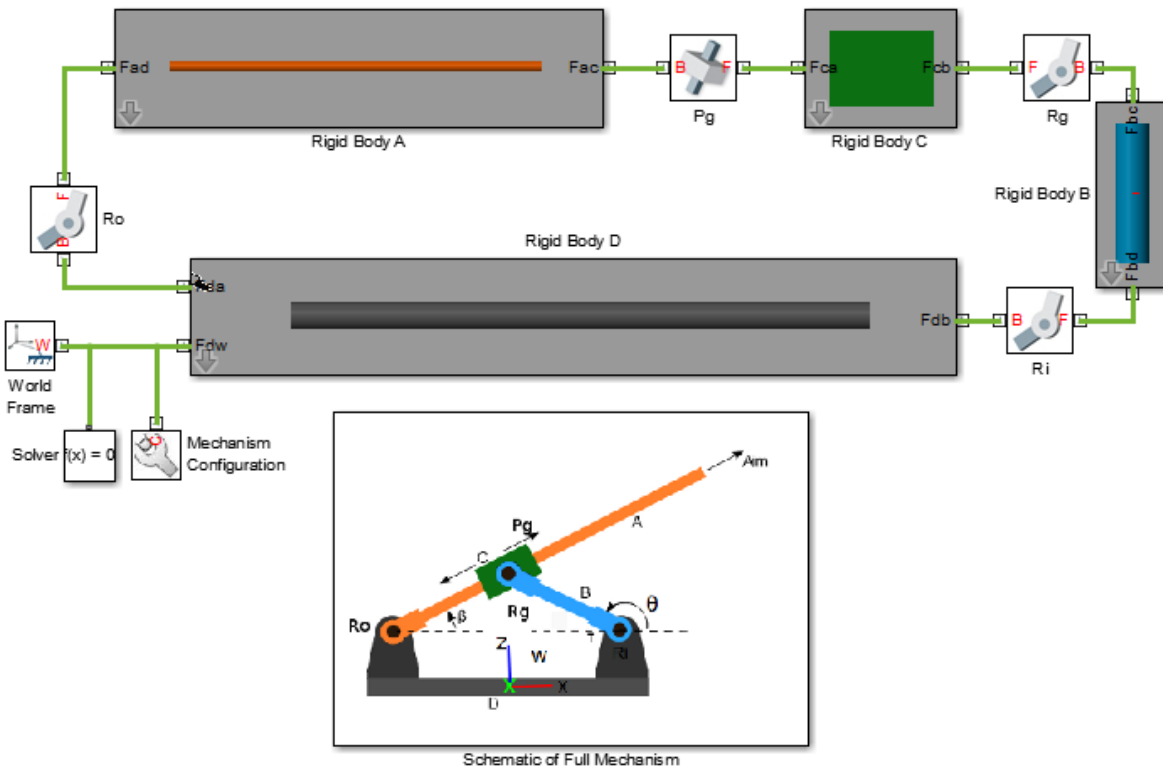
Explore Model

To open the model, at the MATLAB command line, enter `sm_dcrankaim_assembly_with_error`. The model opens in a new window.

The figure shows a schematic of the system that the model represents. This system contains four bodies, labeled A-D. These bodies connect in a closed loop via four joints, labeled Ri, Ro, Rg, and Pg. When connected to each other, these components form a system with one degree of freedom.

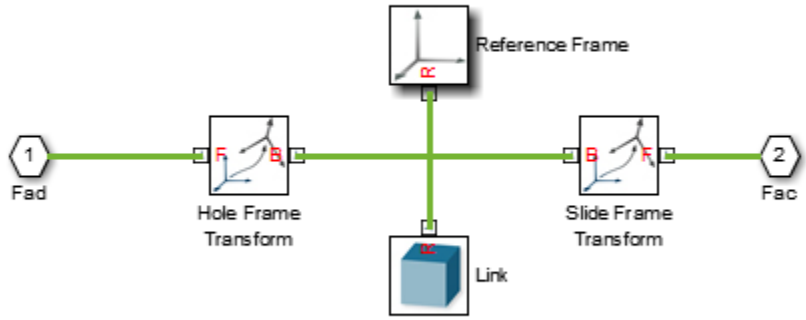


The model represents the components of this system using blocks. Each block represents a physical component. A World Frame block provides the ultimate reference frame in the model. The figure shows the block diagram that the model uses to represent the double-crank aiming mechanism.



To represent the bodies, the model contains four subsystem blocks, labeled Rigid Body A-D. Each subsystem contains one Solid block and multiple Rigid Transform blocks. The Solid block provides geometry, inertia, and color to the body subsystem. The Rigid Transform blocks provide the frames that you connect the joints to. A Reference Frame block identifies the ultimate reference frame in the subsystem block.

The model labels the body subsystem blocks Rigid Body A-D. To examine the block diagram for a body subsystem, right-click the subsystem block and select **Mask > Look Under Mask**. The figure shows the block diagram for Rigid Body A.



To represent the joints, the model contains four joint blocks. Three joints provide one rotational degree of freedom between a pair of bodies. You represent each of these joints with a Revolute Joint block. A fourth joint provides one translational degree of freedom between a pair of bodies. You represent this joint with a Prismatic Joint block. The model labels the revolute joint blocks Ro, Rg, and Ri, and the prismatic joint block Pg.

Update Model

As the model name suggests, this model contains an error. The error prevents the model from assembling successfully, which causes simulation to fail. To update the model and investigate the assembly error:

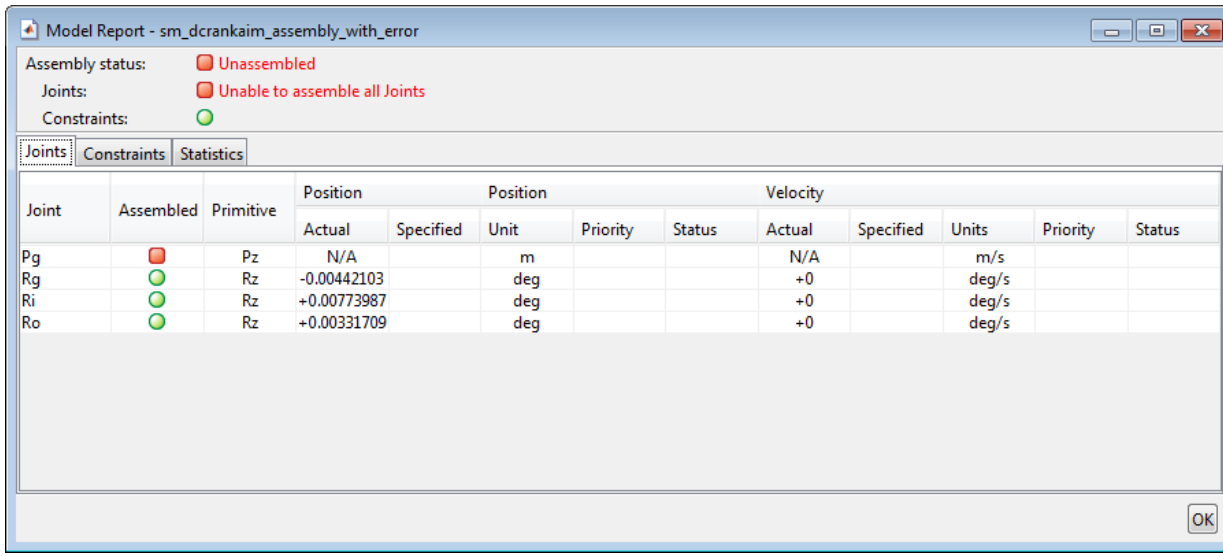
- On the Simulink menu bar, select **Simulation** > **Update Diagram**.

  Mechanics Explorer opens with a static display of your model in its initial state. Because the model contains an assembly error, Simscape Multibody issues an error message. Ignore that message for now.

Troubleshoot Assembly Error

Mechanics Explorer provides access to Model Report, a Simscape Multibody utility that summarizes the assembly status of each joint and constraint in a model. Open this utility to determine which joint has failed to assemble. To do this, in the Mechanics Explorer menu bar, select **Tools** > **Model Report**.

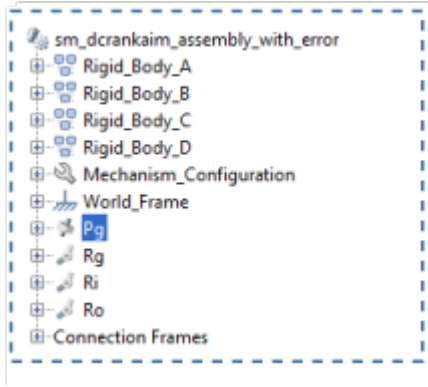
Model Report opens in a new window. A red square indicates that the model, as expected, has failed to assemble. A second red square indicates that an unassembled joint, Pg, is the only contributing factor in the model assembly error. This information enables you to concentrate your troubleshooting efforts on a small block diagram region—that surrounding the Pg joint block.



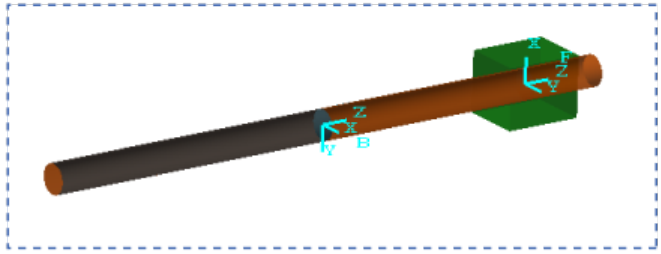
Identifying Error Root Cause

The error message that Simscape Multibody issued during model update identifies position violation as the root cause of assembly failure. This suggests that the frames connected by joint Pg are improperly aligned. To confirm this hypothesis, check the orientation of these frames in Mechanics Explorer.

1 In the Mechanics Explorer tree pane, select **Pg**.



2 In the Mechanics Explorer visualization pane, examine the position and orientation of the highlighted frames. These are the frames that appear in a light turquoise blue color.



The two frames are offset along the Z axis. This offset is valid, since joint Pg contains a prismatic primitive aligned with the Z axis, providing the frames with one translational degree of freedom along that axis. However, the two frames are also rotated with respect to each other about the common Z axis. This offset is invalid, since joint Pg contains no Revolute or Spherical primitives, and hence no rotational degrees of freedom about any axis. To correct the model assembly error, you must rotate either of the two frames so that all of their axes are parallel to each other.

Correct Assembly Error

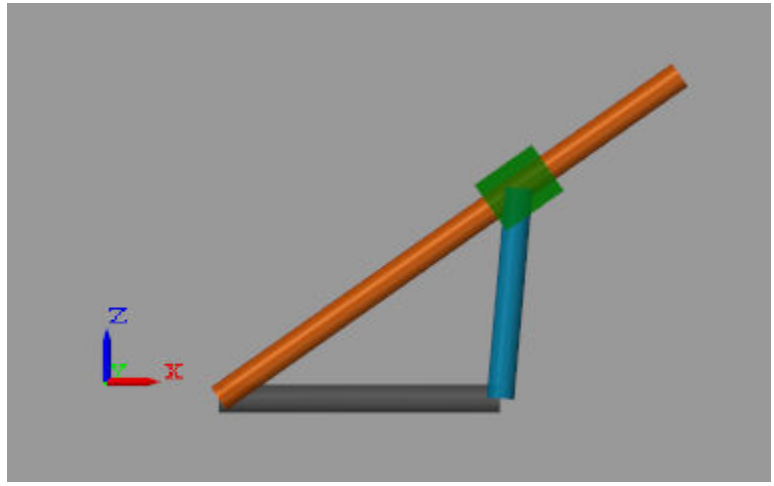
In this example, you apply a rotation transform to the follower frame so that its axes lie parallel to the base frame axes. Alternatively, you could apply an equivalent rotation transform to the base frame. This step enables joint Pg, and hence the model itself, to assemble successfully.

1. In the tree pane of Mechanics Explorer, right-click the Pg node and select `Go To Block`. Simscape Multibody brings the block diagram to the front and highlights the Pg block.
2. Right-click the Rigid Body C subsystem block and select **Mask > Look Under Mask**.
3. Double-click the **Slide Frame Transform** block and select the new parameter values that the table provides. Select **OK**.

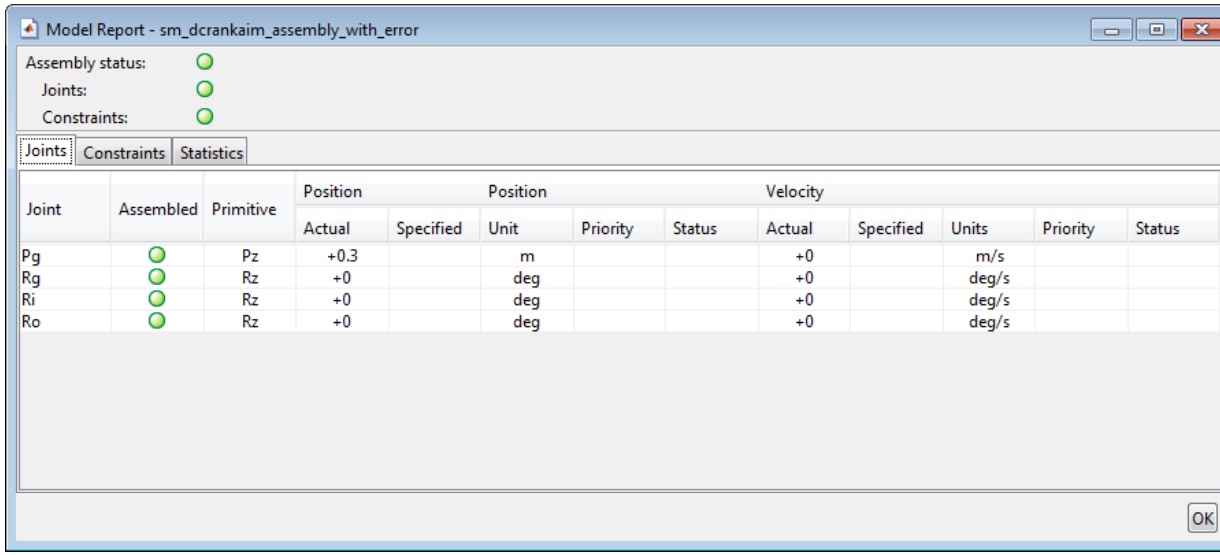
Parameter	New Value
Rotation > Pair 2 > Follower	`+X`
Rotation > Pair 2 > Base	`+Y`

Simulate Model

You can now simulate the model. On the Simulink menu bar, select **Simulation > Run**. Mechanics Explorer opens with a 3-D animation of your model. The figure shows a snapshot of the animation. Rotate, roll, pan, and zoom to explore.



You can use the Model Report tool to verify the assembly status. To do this, in the Mechanics Explorer menu bar, select **Tools > Model Report**. In Model Report, check that the assembly status icons for the model and its joints are green circles. The green circles indicate that the model has assembled correctly.



See Also

Related Examples

- “Model an Open-Loop Kinematic Chain” on page 2-16
- “Model a Closed-Loop Kinematic Chain” on page 2-21

More About

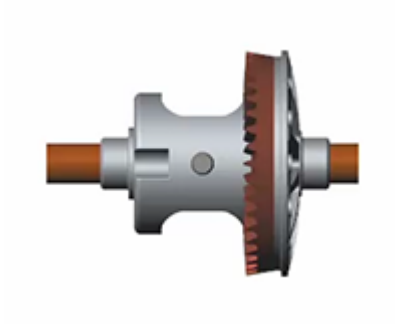
- “Modeling Joint Connections” on page 2-4

Modeling Gear Constraints

In this section...
“Gear Constraints and Applications” on page 2-36
“Gear Assemblies as Kinematic Loops” on page 2-37
“Gear Assembly Restrictions” on page 2-39
“Gear Pitch Circles” on page 2-40
“Modeling Gear Geometries” on page 2-41
“Limitations of Gear Constraints” on page 2-42

Gear Constraints and Applications

Gear assemblies are ubiquitous in rotating machinery. They appear in couplings and drives, often as gear trains, where they transmit torque at a ratio or at an angle between moving bodies. Some, like rack-and-pinion assemblies, serve special purposes, such as converting between rotational and translational motions.



Gears in an Automotive Differential

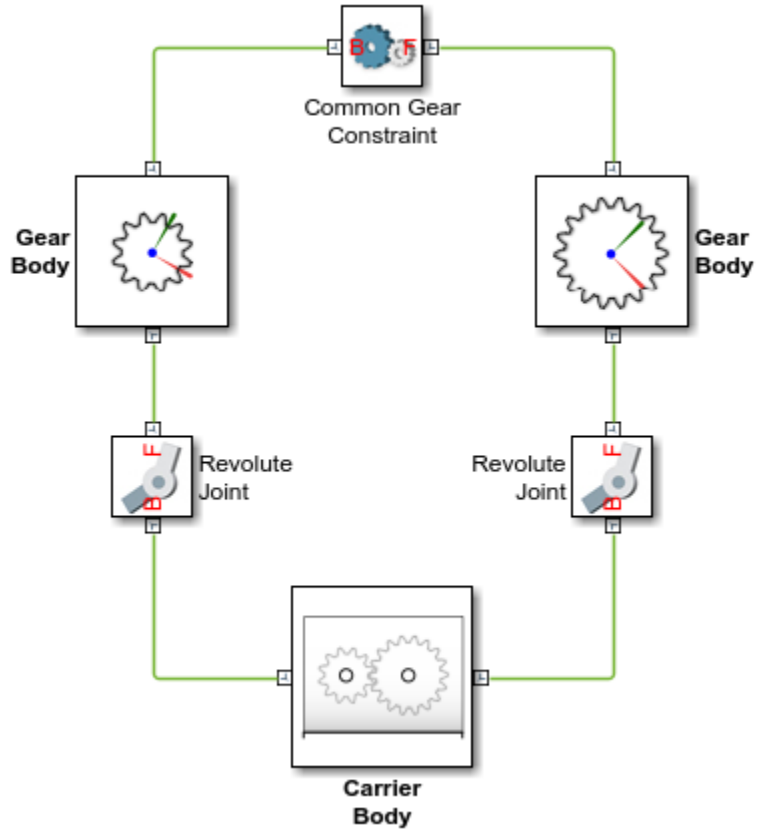
The kinematics of gears in mesh arise from what are, in computational terms, algebraic constraints between the gear rotations. Gear teeth cannot physically overlap and the gears must, at a contact point known as the pitch point, move with the same instantaneous linear velocity.

Gear constraint blocks capture the effects of these constraints in a model. The blocks, found in the **Gears and Couplings** > **Gears** library, include:

- Bevel Gear Constraint — Couple two gears, generally conical in cross-section, with intersecting rotation axes meeting at a right or general angle. Bevel gear assemblies are common in the drivetrains of rotorcraft, where they transmit torque between rotor shafts mounted at an angle.
- Common Gear Constraint — Couple two gears, generally cylindrical in cross-section, with internal or external meshing and parallel rotation axes. Common gear assemblies appear in automotive transmissions, often as planetary gear trains, that transmit power from engine to wheels at preset torque ratios.
- Rack and Pinion Constraint — Couple a rotating pinion to a translating rack with the respective motion axes facing at a right angle. Rack-and-pinion assemblies are common in power steering systems, where they transform a rotation of the steering wheel into a translation of the tie rods, causing the steering arms and wheels to turn.
- Worm and Gear Constraint — Couple a worm and a gear with nonintersecting rotation axes facing at a right angle. Worm-and-gear assemblies form the foundation of slew drives built into solar trackers that are designed to follow the sun and maximize the intensity of sunlight striking a solar panel array.

Gear Assemblies as Kinematic Loops

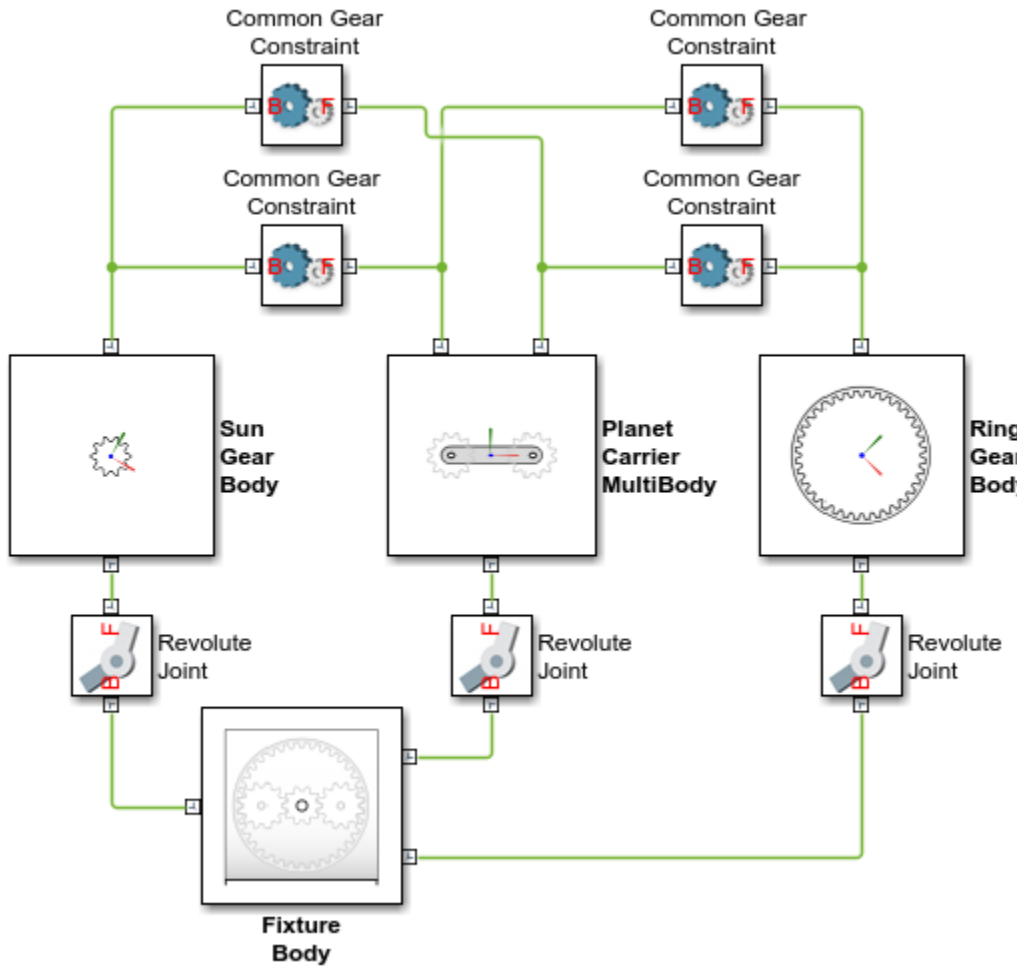
From a topological point of view, gear assemblies form closed kinematic chains, or *loops*. A simple loop comprises two or more gears—the term used loosely here to include worms, pinions, and racks—and a fixture, to hold the gears. The gears connect on one end to the fixture through joints, and on the other end to each other through a gear constraint.



Simple Gear Kinematic Chain

The joints define the degrees of freedom available to the gears before they are brought into mesh. The degrees of freedom encode the types of motion the gears are capable of and the respective motion axes. The gear constraint couples the gears so that they move as though in mesh at a speed ratio determined from the gear (pitch) radii or tooth counts.

More complex model topologies are possible. In a planetary gear train, a ring gear adds a second kinematic loop to the model. Planet gears attached to a carrier add still more kinematic loops. Still, no matter how unique the gear assembly, the model must by its nature comprise at least one kinematic loop.



Planetary Gear Kinematic Loops

Gear Assembly Restrictions

Gear constraints impose special restrictions on the positions and orientations of the gear connection frames. These restrictions are in addition to the meshing constraint, which couples the motions of the gears about the respective rotation axes, and serve to ensure

that the gears are always arranged in mesh. For example, the Common Gear Constraint block requires that:

- The distance between the *z*-axes be equal to the distance between the gear centers.
- The follower frame origin lie on the *xy* plane of the base frame.
- The *z*-axes of the base and follower frames point in the same direction.

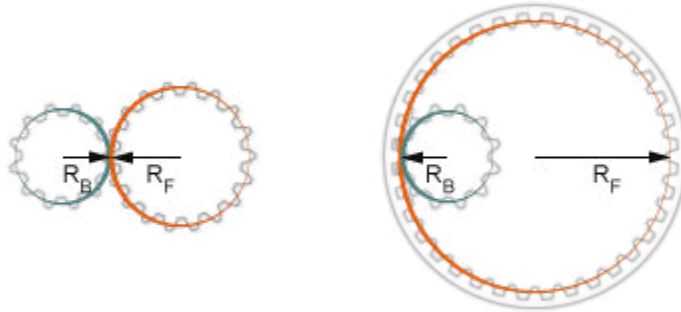
The gear constraint blocks enforce the assembly restrictions, but during model assembly only, when the gears are first placed in mesh. Once simulation starts, it is the task of the model to ensure that the gear placement still satisfies the assembly requirements. The gear constraint blocks then enforce the meshing constraint but merely monitor the assembly constraints, to ensure that the gears remain in a valid configuration.

For examples showing how to properly place the gear connection frames using Rigid Transform blocks, see:

- “Bevel Gear” on page 2-46
- “External Spur Gear” on page 2-50
- “Internal Spur Gear” on page 2-54
- “Rack and Pinion” on page 2-57
- “Worm and Gear” on page 2-61

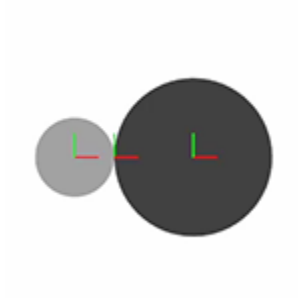
Gear Pitch Circles

Gear constraints are parameterized in terms of pitch circle dimensions. A pitch circle is an imaginary circle concentric with a gear or worm and tangent to the tooth contact point. Every gear and worm has a pitch circle. The figure shows the pitch circles of spur gears with external and internal meshing. The parameters R_B and R_F denote the gear pitch radii.

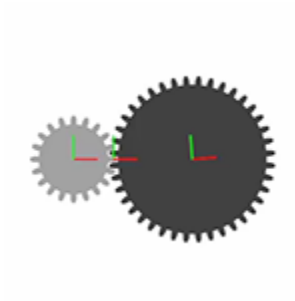


Modeling Gear Geometries

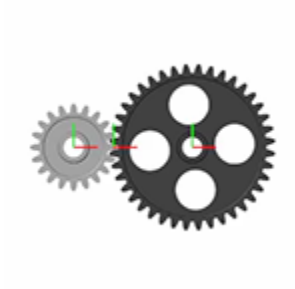
You can approximate gears, worms, and racks using standard solid shapes. Use cylinders with radii equal to the pitch radii for gears and worms. You can use cones for bevel gears and bricks for rack shapes. The figure shows an example with spur gear geometries reduced to cylinders. If you are new to modeling bodies using standard solid shapes, see "Model a Simple Link".



For more detailed geometries, use the `General Extrusion` solid shape. This shape enables you to specify the toothed cross-sectional shapes of gears and racks. The Solid block generates 3-D extrusions by sweeping the cross-sections along their normal axes. The figure shows an example with spur gear geometries modeled as general extrusions. For an example showing how to model a simple body with a `General Extrusion` solid shape, see "Modeling Extrusions and Revolutions" on page 1-50.



For precise geometries, you can load 3-D shapes into Solid blocks using STEP or STL files. You must obtain the STEP or STL files from external sources. If you have CAD models of gears, worms, and racks, you may be able to export them in STEP or STL format for use in Simscape Multibody software. The figure shows an example with spur gear geometries imported from CAD models via STEP files.



Limitations of Gear Constraints

The physical models provided by the gear constraint blocks are idealized. Gear friction, inertia, and backlash are ignored. You add viscous damping to the gear shafts by specifying damping coefficients in the joint blocks that represent the shaft joints. The shaft joint blocks are typically located between the gear shaft bodies and the gear carrier body. You add inertia to the gears by modeling the gear bodies using Solid, Inertia, or General Variable Mass blocks.

See Also

Related Examples

- “Assemble a Gear Model” on page 2-44

Assemble a Gear Model

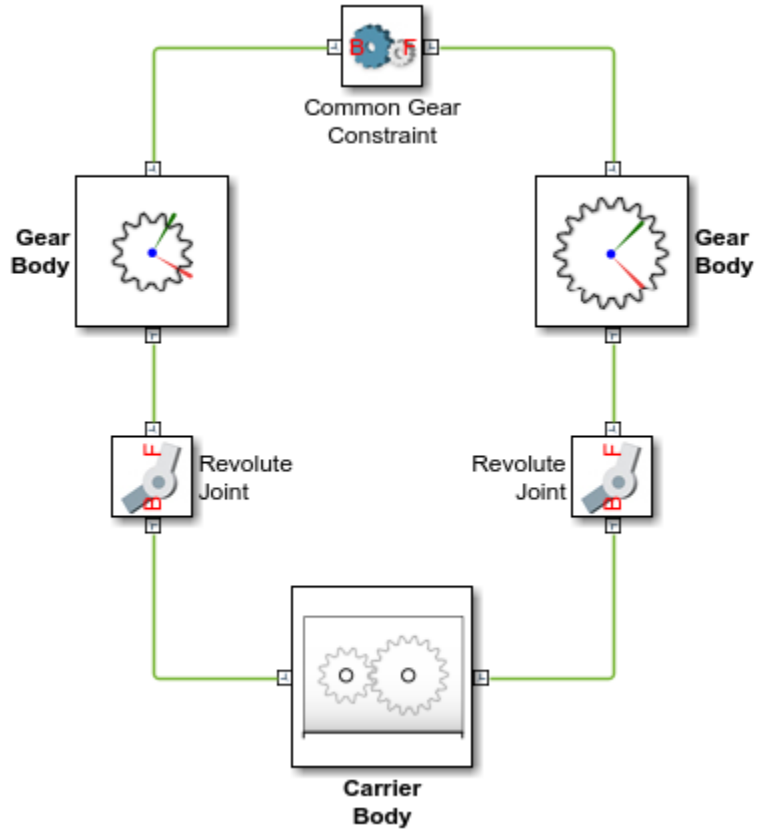
In this section...

"Gear Examples" on page 2-44
"Bevel Gear" on page 2-46
"External Spur Gear" on page 2-50
"Internal Spur Gear" on page 2-54
"Rack and Pinion" on page 2-57
"Worm and Gear" on page 2-61

Gear Examples

The examples that follow show how to position and orient gear bodies so that they satisfy the assembly requirements of the various gear constraint blocks. Each example starts with an overview of relevant gear dimensions and frame placements. These attributes guide the selection of rigid transforms needed to ensure that the gears assemble in mesh.

The models share the same block diagram topology, with the model components—bodies, joints, and gear constraint—arranged in a kinematic loop in each case. The figure shows a simple loop. The carrier body is in the examples considered to be fixed to the world frame, with its inertia consequently reduced to a superfluous detail and the body altogether ignored.



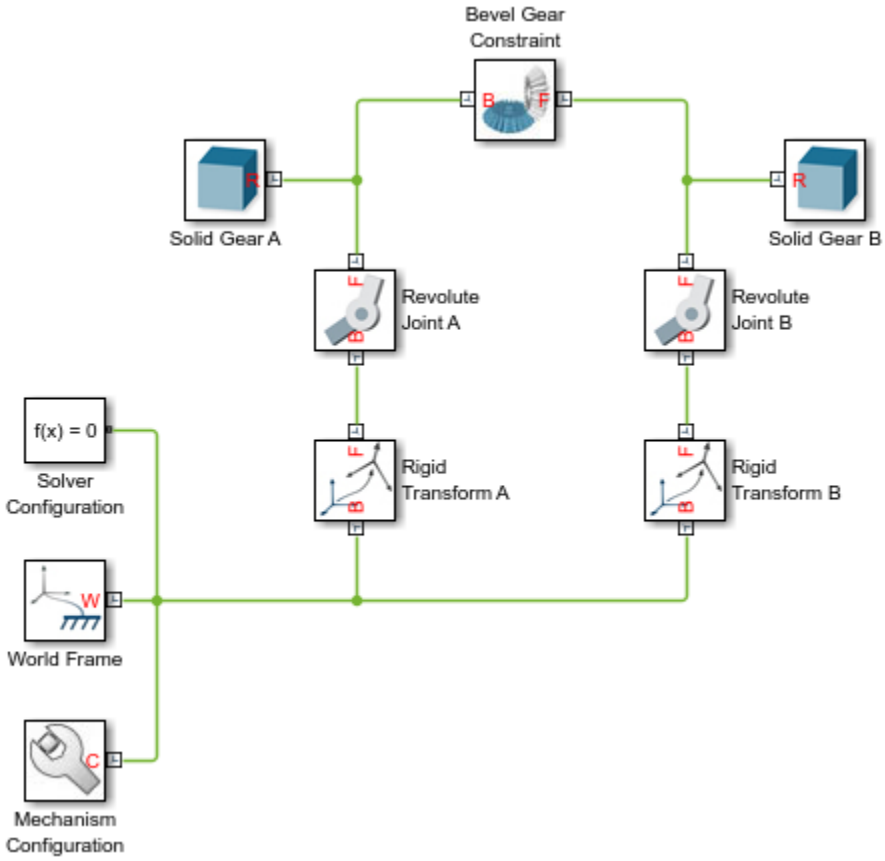
The models comprise four types of Simscape Multibody blocks:

- Solid — Provides the gear geometries, inertias, and colors. The gear geometries, complete with teeth or threads to more clearly show the gears in mesh, are imported from STEP files. The poses of the gear reference frames relative to the gear geometries are obtained from the same files.
- Joint — Provides the gear bodies with the requisite degrees of freedom. Revolute Joint blocks enable rotation about a single axis. Prismatic Joint blocks enable translation along a single axis. Velocity state targets specified in the joint blocks set the gears in motion.

- Rigid Transform — Rotates and translates the joints and the attached gear bodies so that they are properly placed for meshing. Rigid Transform blocks provide the means to change the gear placements and therefore to satisfy the gear assembly requirements.
- Gear constraint — Couples the motions of the gear bodies. Gear constraint blocks eliminate one degree of freedom between the gears, causing them to move as though in mesh. The examples showcase, one by one, the various gear constraint blocks.

Bevel Gear

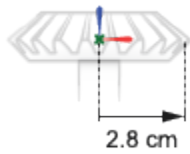
The `smdoc_bevel_gear_start` model, shown in the figure, provides an example of a bevel gear assembly. The model, based on the Bevel Gear Constraint block, is complete in every sense but one—all rigid transforms are zero and the gear reference frames are therefore coincident in space.



This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

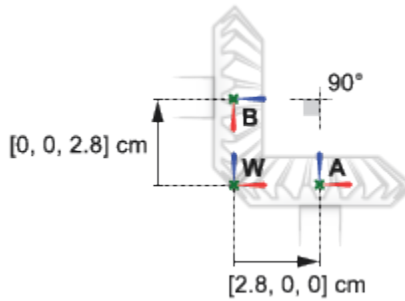
Gear Geometry

The bevel gears, **A** and **B**, are identical in size, with a pitch radius of `2.8 cm` in each case. The gear reference frames are placed with origins at the gear centers and *z*-axes aligned with the gear rotation axes so as to face away from the gear shafts. This alignment is consistent with the Revolute Joint blocks, which allow rotation about the *z*-axis only.



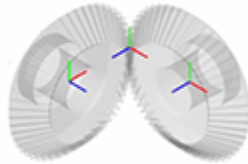
Gear Assembly

The gear rotation axes meet at a right angle. The reference frame of bevel gear **A** sits at an offset of `[2.8, 0, 0] cm`, in Cartesian coordinates, relative to the world frame. The reference frame of bevel gear **B** sits at an offset of `[0, 0, 2.8] cm` relative to the world frame and at an angle of `90 deg` about the *y*-axis also of the world frame.



Complete the Model

Complete the bevel gear model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_bevel_gear_start` at the MATLAB command prompt.

1 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of bevel gear **A** relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	`Cartesian`
Offset	`[2.8, 0, 0] cm`

2 In the Rigid Transform B block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of bevel gear **B** relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	`Cartesian`
Offset	`[0, 0, 2.8] cm`

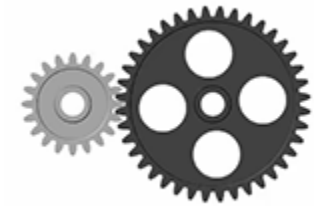
3 In the Rigid Transform B block dialog box, specify the **Rotation** parameters shown in the table. These parameters set the orientation of bevel gear **B** relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	`Standard Axis`
Axis	`+Y`
Angle	`90 deg`

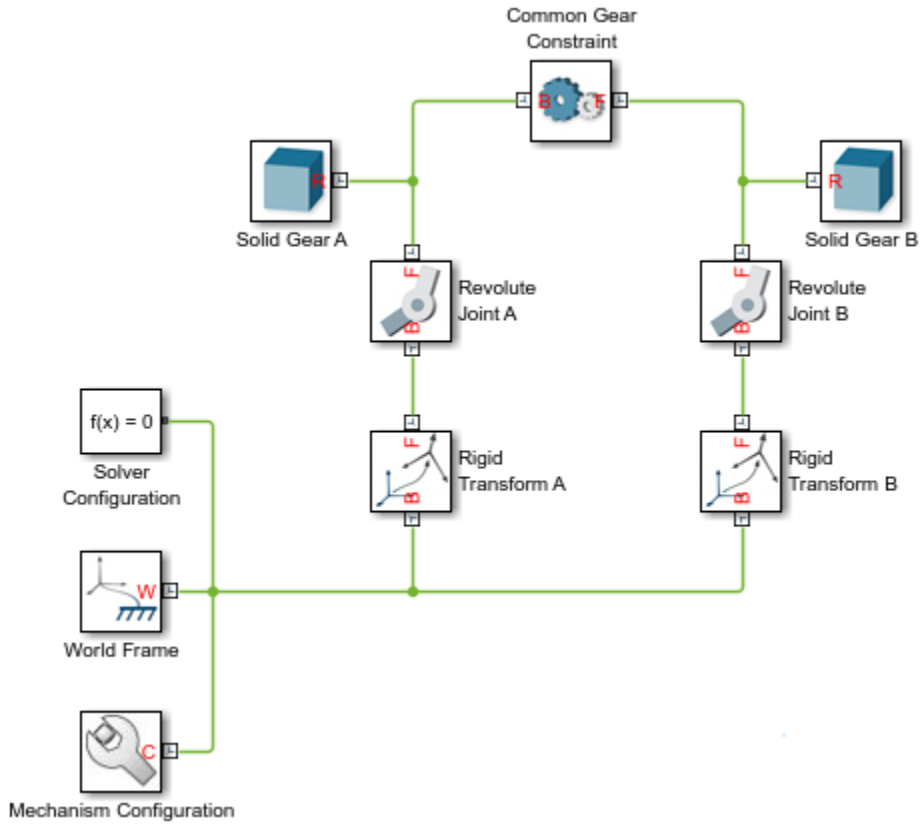
4 Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

To see a complete bevel gear model, at the MATLAB command prompt enter `smdoc_bevel_gear`. Simscape Multibody opens a bevel gear model with the rigid transforms described in this example.

External Spur Gear



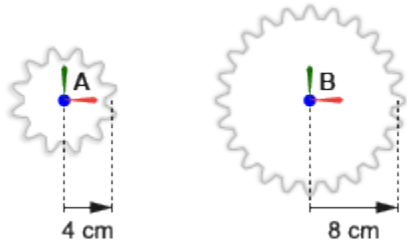
The `smdoc_common_gear_external_start` model, shown in the figure, provides an example of an external spur gear assembly. The model, based on the Common Gear Constraint block, is complete in every sense but one—all rigid transforms are zero and the gear reference frames are therefore coincident in space.



This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

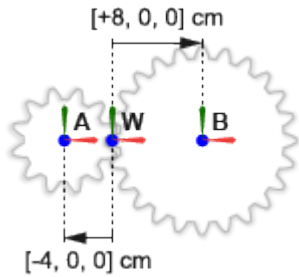
Gear Geometry

The small spur gear, **A**, has a pitch radius of `4 cm`. The large spur gear, **B**, has a pitch radius of `8 cm`. The gear reference frames are placed with origins at the gear centers and *z*-axes aligned with the gear rotation axes so as to face away from the gear shafts. This alignment is consistent with the Revolute Joint block, which allows rotation about the *z*-axis only.



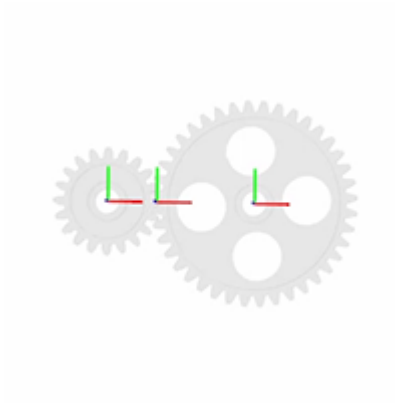
Gear Assembly

The spur gear rotation axes are parallel to each other. The reference frame of the small spur gear sits at an offset of `[-4, 0, 0] cm`, in Cartesian coordinates, relative to the world frame. The reference frame of the large spur gear sits at an offset of `[-8, 0, 0] cm`, also relative to the world frame.



Complete the Model

Complete the external spur gear model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_common_gear_external_start` at the MATLAB command prompt.

1 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the small spur gear, **A**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	`Cartesian`
Offset	`[-4, 0, 0] cm`

2 In the Rigid Transform B block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the large spur gear, **B**, relative to the world frame as described in the Gear Assembly schematic.

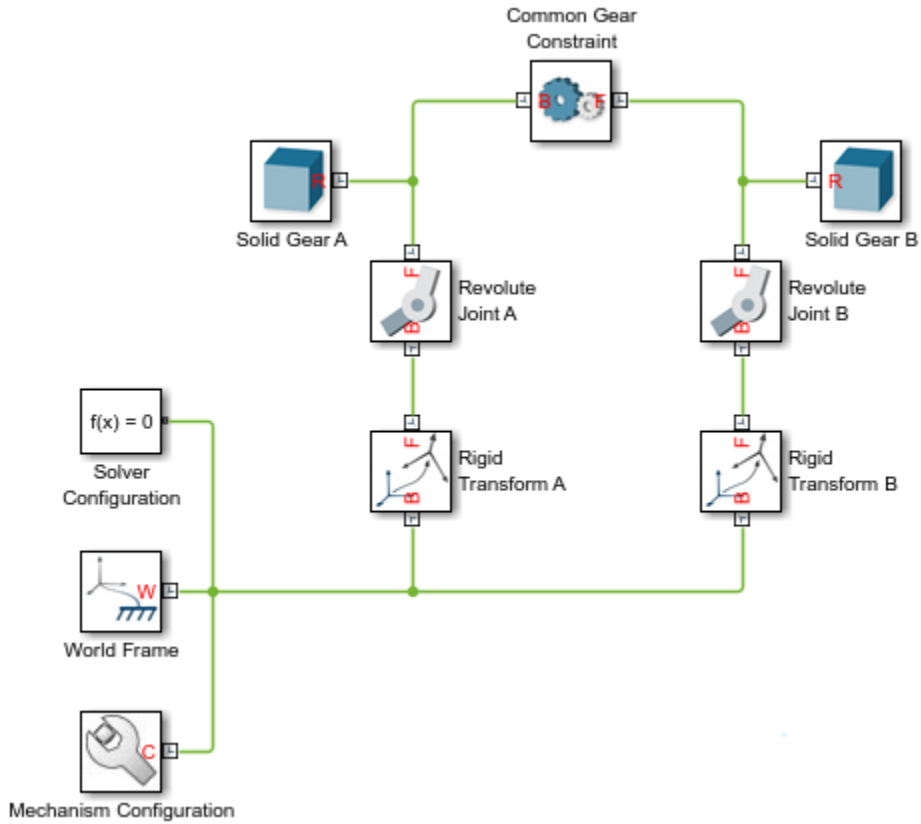
Parameter	Setting
Method	`Cartesian`
Offset	`[8, 0, 0] cm`

3 Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

To see a complete external spur gear model, at the MATLAB command prompt enter `smdoc_common_gear_external`.

Internal Spur Gear

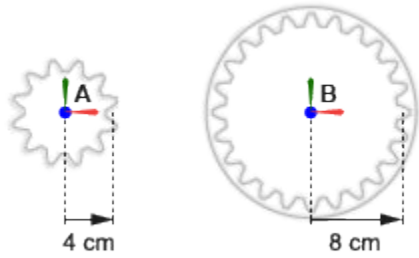
The `smdoc_common_gear_internal_start` model, shown in the figure, provides an example of an internal spur gear assembly. The model, based on the Common Gear Constraint block, is complete in every sense but one—all rigid transforms are zero and the gear reference frames are therefore coincident in space.



This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

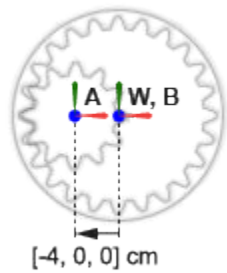
Gear Geometry

The spur gear, **A**, has a pitch radius of `4 cm`. The ring gear, **B**, has a pitch radius of `8 cm`. The gear reference frames are placed with origins at the gear centers and *z*-axes aligned with the gear rotation axes so as to face away from the gear shafts. This alignment is consistent with the Revolute Joint block, which allows rotation about the *z*-axis only.



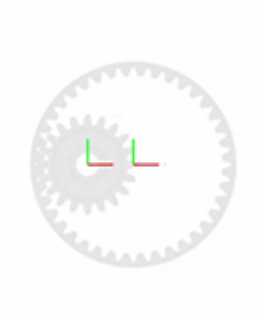
Gear Assembly

The gear rotation axes are parallel to each other. The spur gear reference frame sits at an offset of `[-4, 0, 0] cm`, in Cartesian notation, relative to the world frame. The ring gear reference frame sits left with its origin and *z*-axis coincident with those of the world frame.



Complete the Model

Complete the internal spur gear model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_common_gear_internal_start` at the MATLAB command prompt.

1 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the spur gear, **A**, relative to the world frame as described in the Gear Assembly schematic.

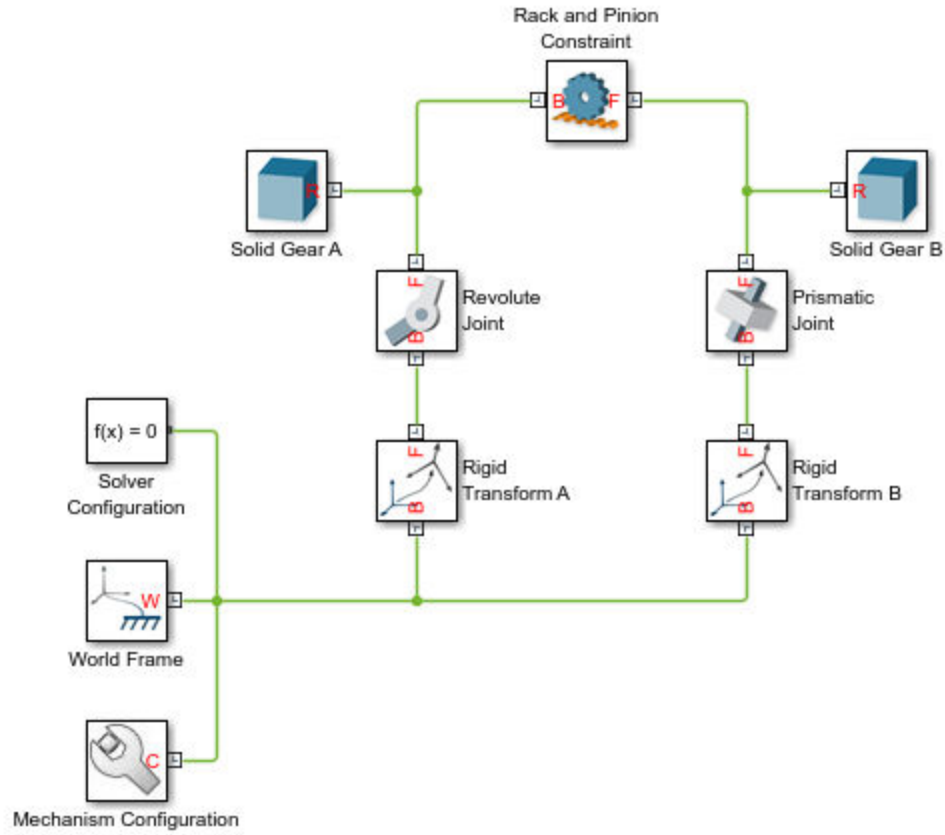
Parameter	Setting
Method	`Cartesian`
Offset	`[-4, 0, 0] cm`

2 Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

To see a complete internal spur gear model, at the MATLAB command prompt enter `smdoc_common_gear_internal`.

Rack and Pinion

The `smdoc_rack_and_pinion_start` model, shown in the figure, provides an example of a rack-and-pinion assembly. The model, based on the Rack and Pinion Constraint block, is complete in every sense but one—all rigid transforms are zero and the gear reference frames are therefore coincident in space.

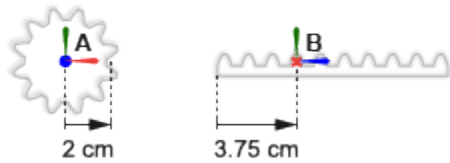


This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

Gear Geometry

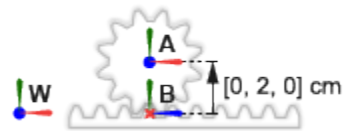
The pinion, **A**, has a pitch radius of `2 cm`. The pinion reference frame is placed with origin at the pinion center and *z*-axis along the pinion axis. The rack reference frame is

placed with origin `3.75 cm` from the rack edge and *z*-axis along the rack length. The frame alignments are consistent with the Revolute Joint and Prismatic Joint blocks, which allow motion about or along the *z*-axis only.



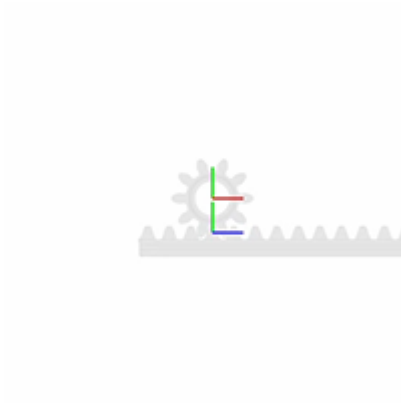
Gear Assembly

The rack translation axis is at a right angle to the pinion rotation axis. The pinion reference frame sits at an offset of `[0, 2, 0] cm`, in Cartesian notation, relative to the world frame. The rack reference frame sits at an angle of `90 deg` relative to the positive *y*-axis of the world frame.



Complete the Model

Complete the rack-and-pinion model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_rack_and_pinion_start` at the MATLAB command prompt.

1. In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the pinion, **A**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	`Cartesian`
Offset	`[0, 2, 0] cm`

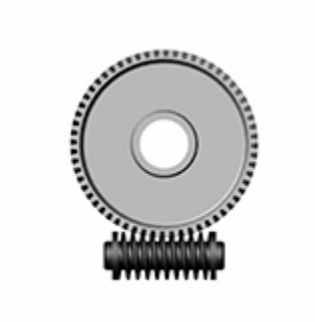
2. In the Rigid Transform B block dialog box, specify the **Rotation** parameters shown in the table. These parameters set the orientation of the rack, **B**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	`Standard Axis`
Axis	`+Y`
Angle	`90 deg`

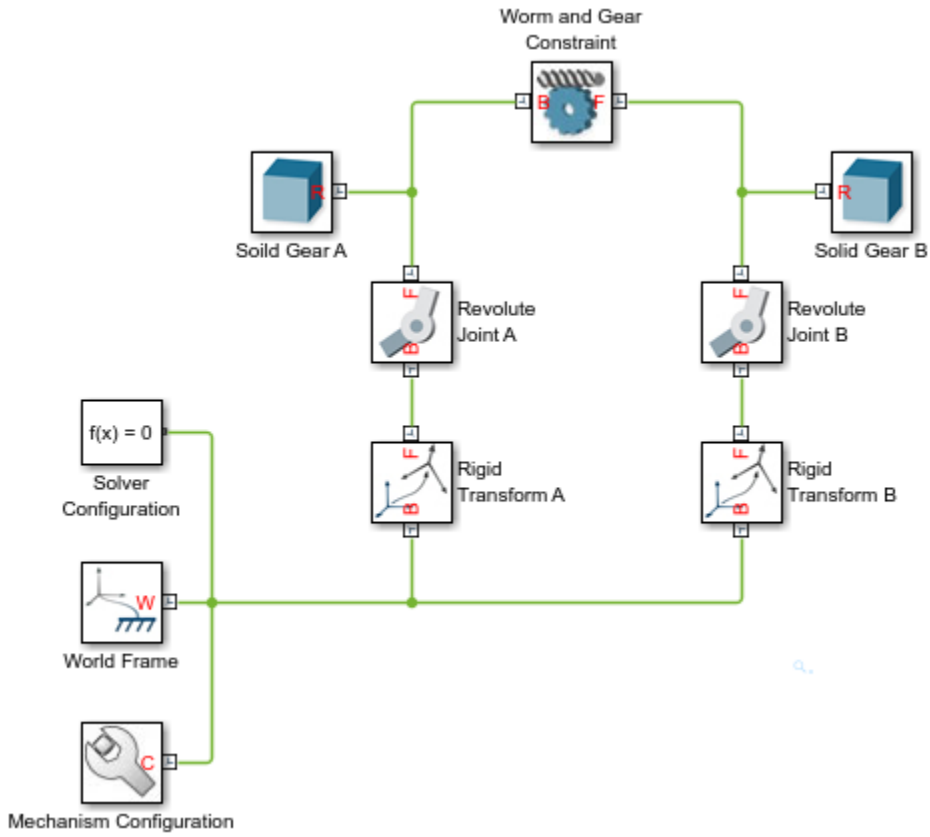
3. Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

   To see a complete rack-and-pinion model, at the MATLAB command prompt enter `smdoc_rack_and_pinion`.

Worm and Gear



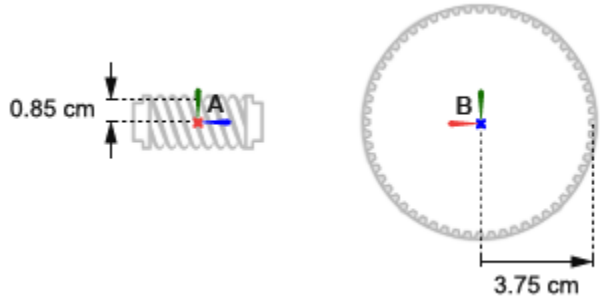
The `smdoc_worm_and_gear_start` model, shown in the figure, provides an example of a worm-and-gear assembly. The model, based on the Worm and Gear Constraint block, is complete in every sense but one—all rigid transforms are zero and the gear reference frames are therefore coincident in space.



This short tutorial shows how suitable transforms follow readily from the gear dimensions and assembly constraints—and how, once specified in the Rigid Transform blocks, they enable the gear model to assemble as though in mesh without error.

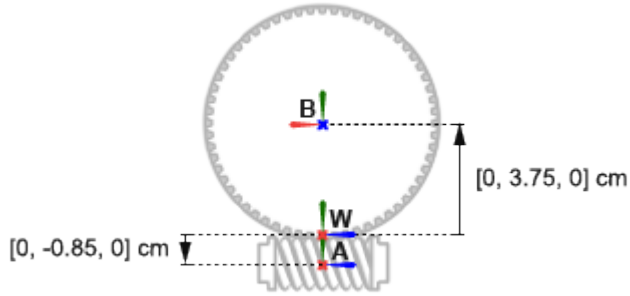
Gear Geometry

The worm, **A**, has a pitch radius of `0.85 cm`. The gear, **B**, has a pitch radius of `3.75 cm`. The worm and gear reference frames are placed with origins at the geometry centers and *z*-axes aligned with the respective rotation axes. This alignment is consistent with the Revolute Joint block, which allows rotation about the *z*-axis only.



Gear Assembly

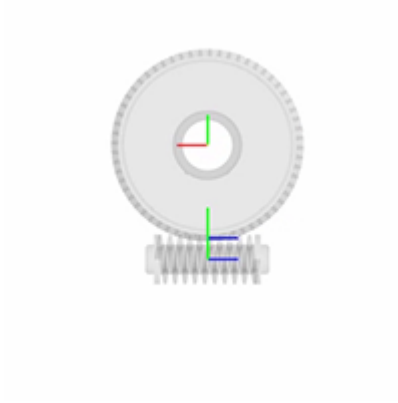
The worm rotation axis is at a right angle to the gear rotation axis. The worm reference frame sits at an offset of `[0, -0.85, 0] cm`, in Cartesian notation, relative to the world frame. The gear reference frame sits at an offset of `[0, +3.75, 0] cm` and at an angle of `90 deg` about the positive *y*-axis relative to the world frame.



Complete the Model

Complete the worm-and-gear model by specifying the rigid transforms described in the gear assembly schematic. The conceptual animation that follows shows the incremental

effects that the rigid transforms would have were they to apply in sequence during model update.



If you have not yet done so, open the incomplete bevel gear model by entering the model name, `smdoc_worm_and_gear_start` at the MATLAB command prompt.

1 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the worm, **A**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	`Cartesian`
Offset	`[0, -0.85, 0] cm`

2 In the Rigid Transform A block dialog box, specify the **Translation** parameters shown in the table. These parameters set the position of the gear, **B**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	`Cartesian`
Offset	`[0, 3.75, 0] cm`

3 In the Rigid Transform B block dialog box, specify the **Rotation** parameters shown in the table. These parameters set the orientation of the gear, **B**, relative to the world frame as described in the Gear Assembly schematic.

Parameter	Setting
Method	`Standard Axis`

Parameter	Setting
Axis	`+Y`
Angle	`90 deg`

4 Simulate the model. Mechanics Explorer opens with the dynamic gear visualization shown at the beginning of this example.

To see a complete worm-and-gear gear model, at the MATLAB command prompt enter `smdoc_worm_and_gear`.

Model a Compound Gear Train

In this section...
“Model Overview” on page 2-66
“Model Sun-Planet Gear Set” on page 2-67
“Constrain Sun-Planet Gear Motion” on page 2-70
“Add Ring Gear” on page 2-72
“Add Gear Carrier” on page 2-75
“Add More Planet Gears” on page 2-80

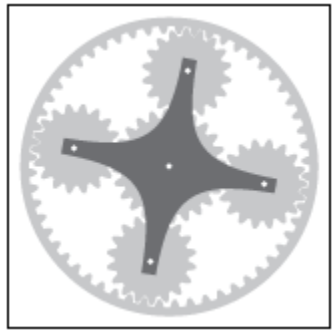
Model Overview

Planetary gear trains are common in industrial, automotive, and aerospace systems. A typical application is the automatic transmission system of car. From a kinematic point of view, what sets this mechanism apart is the kinematic constraint set between gear pairs. These constraints fix the angular velocity ratios of the gear pairs, causing the gears in each pair to move in sync.

In Simscape Multibody, you represent the kinematic constraint between meshed gears using blocks from the Gears sublibrary. This tutorial shows you how to use these blocks to model a planetary gear train. The gear train contains four bodies:

- Sun gear
- Planet gear
- Ring gear
- Planet carrier

Each body, including the planet carrier, can spin about its central axis. In addition, each planet gear can revolve about the sun gear. Joint blocks provide the required degrees of freedom, while gear constraint blocks ensure the gears move as if they were meshed.



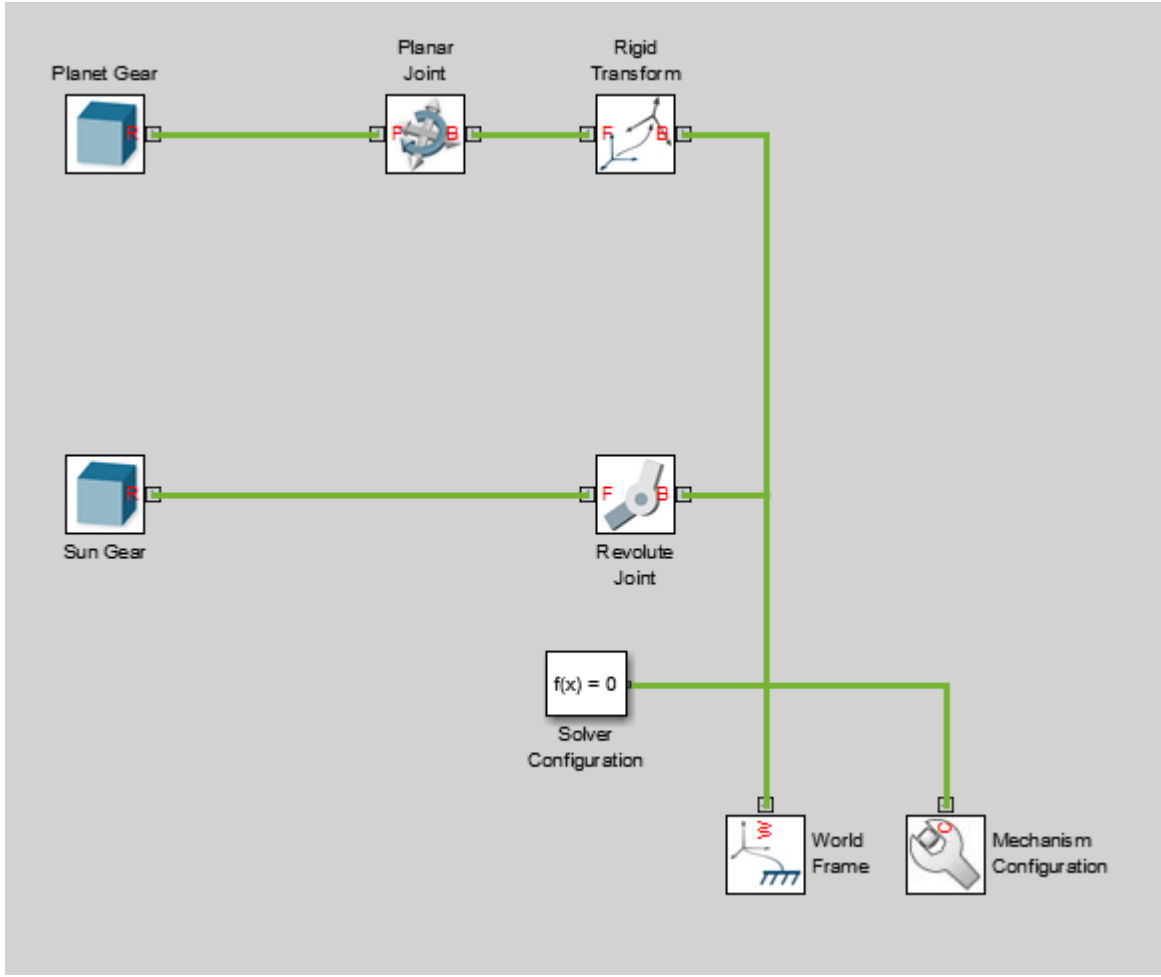
Model Sun-Planet Gear Set

Model the gear bodies and connect them with the proper degrees of freedom. In a later step, you add gear constraints to this model.

1 Drag these blocks to a new model.

Library	Block	Quantity
Body Elements	Solid	2
Joints	Revolute Joint	1
Joints	Planar Joint	1
Frames and Transforms	Rigid Transform	1
Frames and Transforms	World Frame	1
Utilities	Mechanism Configuration	1
Simscape > Utilities	Solver Configuration	1

2 Connect and name the blocks as shown.



3 In the Sun Gear block dialog box, specify these parameters.

Parameter	Setting
Geometry > Shape	Select `General Extrusion`.
Geometry > Cross-Section	Enter `simmechanics.demohelpers.gear_profile(2*Sun.R,Sun.N,A)`. Select units of `cm`.

Parameter	Setting
Geometry > Length	Enter `T`. Select units of `cm`.
Inertia > Density	Enter `Rho`.
Graphic > Visual Properties > Color	Enter `Sun.RGB`.

The `simmechanics.demohelpers.gear_profile` function generates the cross-section matrix for an external gear with an involute tooth profile. The cross-section is approximate. Use the function as an example only.

4 In the Planet Gear block dialog box, specify these parameters.

Parameter	Setting
Geometry > Shape	Select `General Extrusion`.
Geometry > Cross-Section	Enter `simmechanics.demohelpers.gear_profile(2*Planet.R,Planet.N,A)`. Select units of `cm`.
Geometry > Length	Enter `T`. Select units of `cm`.
Inertia > Density	Enter `Rho`.
Graphic > Visual Properties > Color	Enter `Planet.RGB`.

5 In the Rigid Transform block dialog box, specify these parameters.

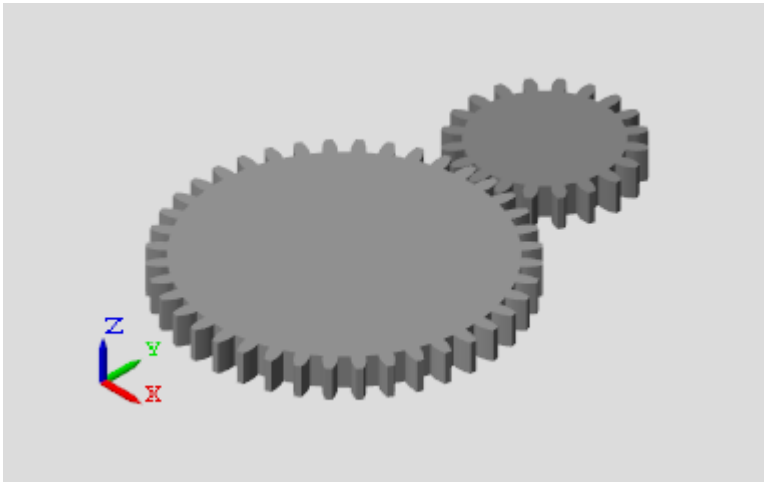
Parameter	Setting
Translation > Method	Select `Standard Axis`.
Translation > Axis	Select `+Y`.
Translation > Offset	Enter `Sun.R + Planet.R`. Select units of `cm`.

6 In the model workspace, define the block parameters using MATLAB code:

```
% Common Parameters
Rho = 2700;
T = 3;
A = 0.8; % Gear Addendum

% Sun Gear Parameters
Sun.RGB = [0.75 0.75 0.75];
Sun.R = 15;
Sun.N = 40;
```

```
% Planet Gear Parameters
Planet.RGB = [0.65 0.65 0.65];
Planet.R = 7.5;
Planet.N = Planet.R/Sun.R*Sun.N;
```

7 Simulate the model. To induce motion, try adjusting the velocity state targets in the joint block dialog boxes. Notice that the sun and planet gears move independently of each other. To constrain gear motion, you must add a gear constraint block between the gear solid blocks.



You can open a copy of the resulting model. At the MATLAB command line, enter `smdoc_planetary_gear_a`.

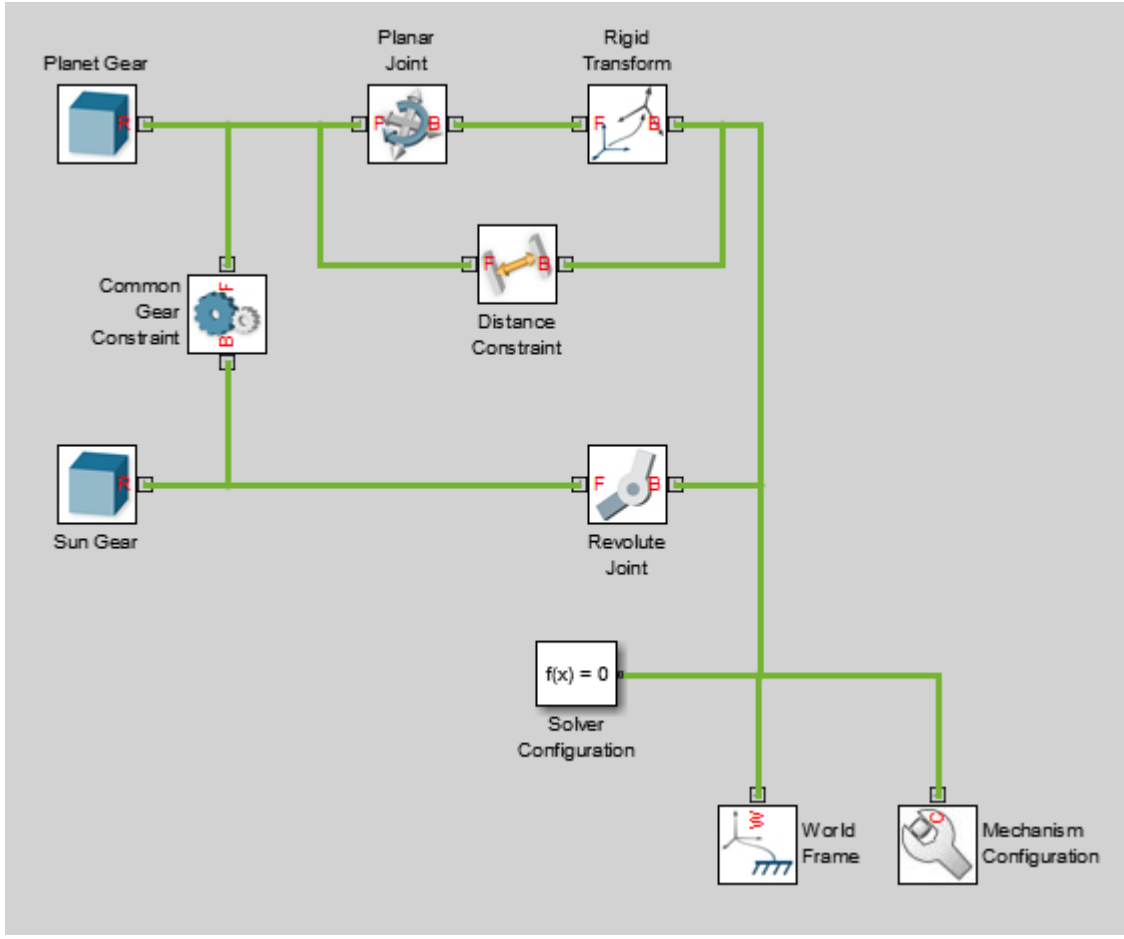
Constrain Sun-Planet Gear Motion

Specify the kinematic constraints acting between the sun and planet gears. These constraints ensure that the gears move in a meshed fashion.

1 Drag these blocks to the sun-planet gear model.

Library	Block
Constraints	Distance Constraint
Gears and Couplings > Gears	Common Gear Constraint

2 Connect the blocks as shown. The new blocks are highlighted.



3 In the Common Gear Constraint block dialog box, specify these parameters.

Parameter	Setting
Specification Method	Select `Pitch Circle Radii`.
Specification Method > Base Gear Radius	Enter `Sun.R`. Select units of `cm`.
Specification Method > Follower Gear Radius	Enter `Planet.R`. Select units of `cm`.

4 In the Distance Constraint block dialog box, specify this parameter:

- **Distance** — Enter `Sun.R + Planet.R`. Select units of `cm`.

5 Simulate the model. To induce motion, try adjusting the velocity state targets in the joint block dialog boxes. Notice that the sun and planet gears now move in sync.

You can open a copy of the resulting model. At the MATLAB command line, enter `smdoc_planetary_gear_b`.

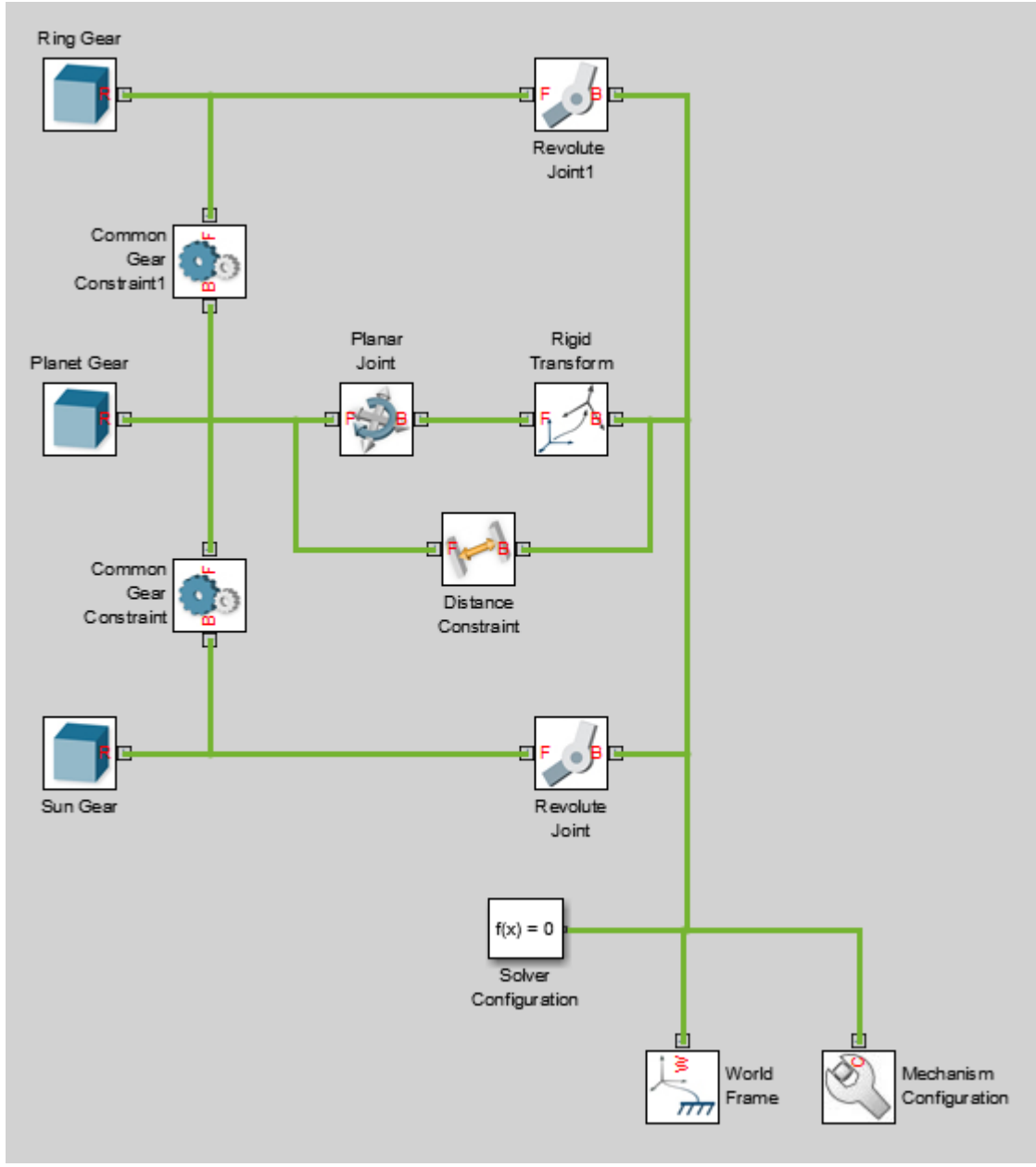
Add Ring Gear

Model the ring gear body, connect it with the proper degrees of freedom, and constrain its motion with respect to the planet gear.

1 Add these blocks to the sun-planet gear model.

Library	Block
Body Elements	Solid
Joints	Revolute Joint
Gears and Couplings > Gears	Common Gear Constraint

2 Connect and name the blocks as shown. The new blocks are highlighted.


Ring Gear
Revolute
Joint1
Common
Gear
Constraint1
Planet Gear
Planar
Joint
Rigid
Transform
Common
Gear
Constraint
Distance
Constraint
Sun Gear
Revolute
Joint
f(x) = 0
Solver
Configuration
World
Frame
Mechanism
Configuration

3 In the Ring Gear block dialog box, specify these parameters.

Parameter	Setting
Geometry > Shape	Select `General Extrusion`.
Geometry > Cross-Section	Enter `Ring.CS`. Select units of `cm`.
Geometry > Length	Enter `T`.
Inertia > Density	Enter `Rho`.
Graphic > Visual Properties > Color	Enter `Ring.RGB`.

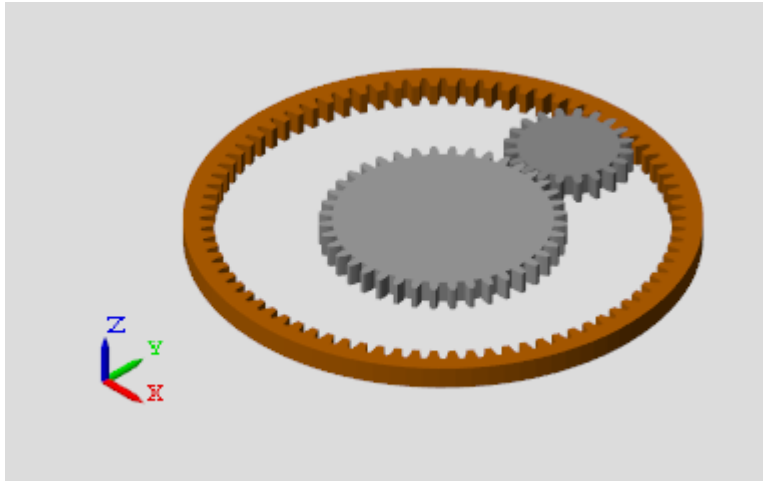
4 In the Common Gear Constraint1 block dialog box, specify these parameters.

Parameter	Setting
Type	Select `Internal`.
Specification Method	Select `Pitch Circle Radii`.
Specification Method > Base Gear Radius	Enter `Planet.R`. Select units of `cm`.
Specification Method > Follower Gear Radius	Enter `Ring.R`. Select units of `cm`.

5 In the model workspace, define the Ring Gear block parameters using MATLAB code:

```
% Ring Gear Parameters
Ring.RGB = [0.85 0.45 0];
Ring.R = Sun.R + 2*Planet.R;
Ring.N = Ring.R/Planet.R*Planet.N;

Ring.Theta = linspace(-pi/Ring.N,2*pi-pi/Ring.N,100)';
Ring.RO = 1.1*Ring.R;
Ring.CSO = [Ring.RO*cos(Ring.Theta) Ring.RO*sin(Ring.Theta)];
Ring.CSI = simmechanics.demohelpers.gear_profile(2*Ring.R,Ring.N,A);
Ring.CSI = [Ring.CSI; Ring.CSI(1,:)];
Ring.CS = [Ring.CSO; flipud(Ring.CSI)];
```

6 Simulate the model. To induce motion, try adjusting the velocity state targets in the joint block dialog boxes. Notice that the sun, planet, and ring gears move in a meshed fashion.



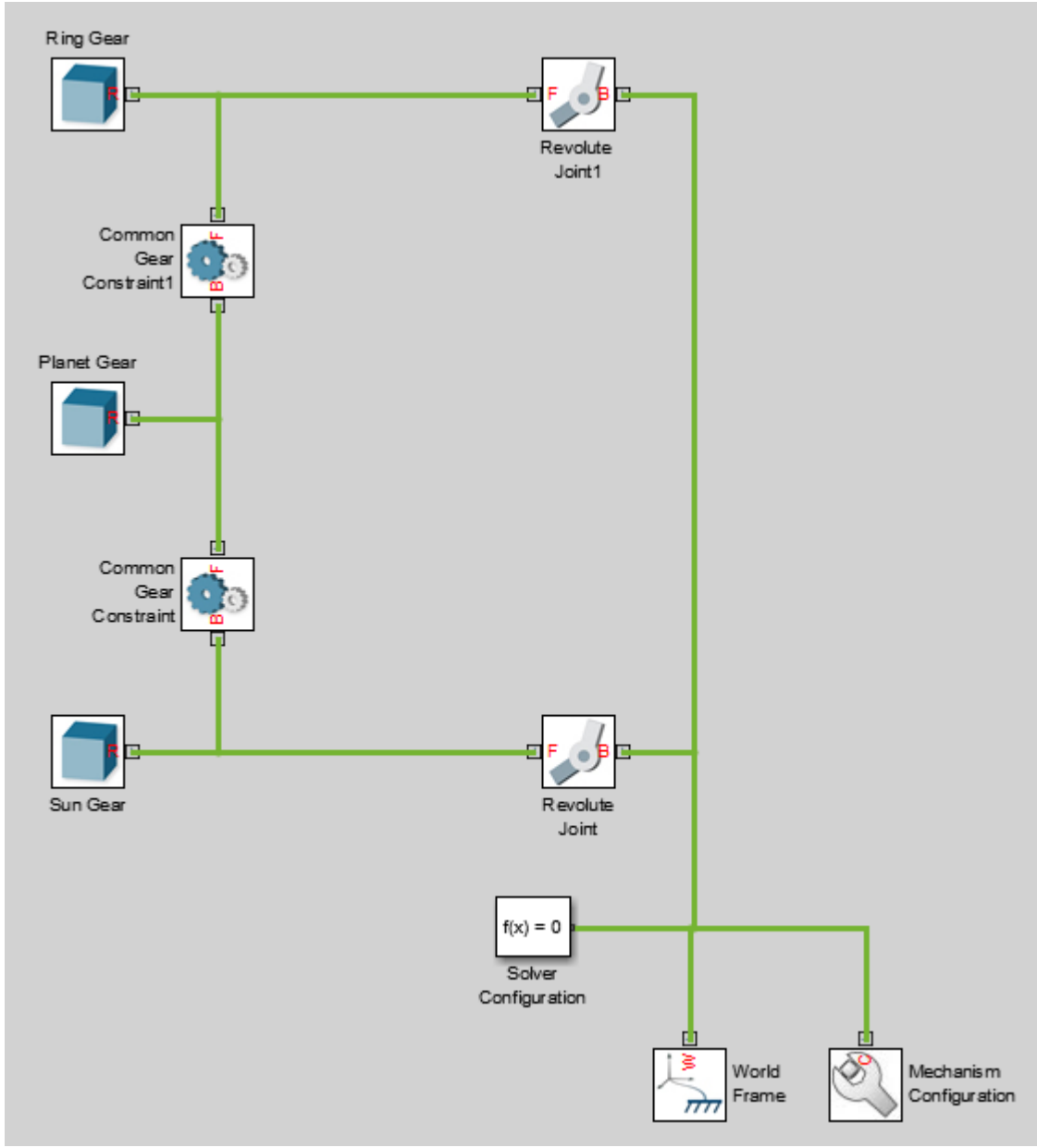
You can open a copy of the resulting model. At the MATLAB command line, enter `smdoc_planetary_gear_c`.

Add Gear Carrier

Up to now, you have kept the sun and planet gears at a fixed distance using a Distance Constraint block. In an actual planetary gear, a gear carrier enforces this constraint. Model the gear carrier and connect it between the sun and planet gears.

1 Remove these blocks from the planetary gear model:

- Planar Joint
- Rigid Transform
- Distance Constraint

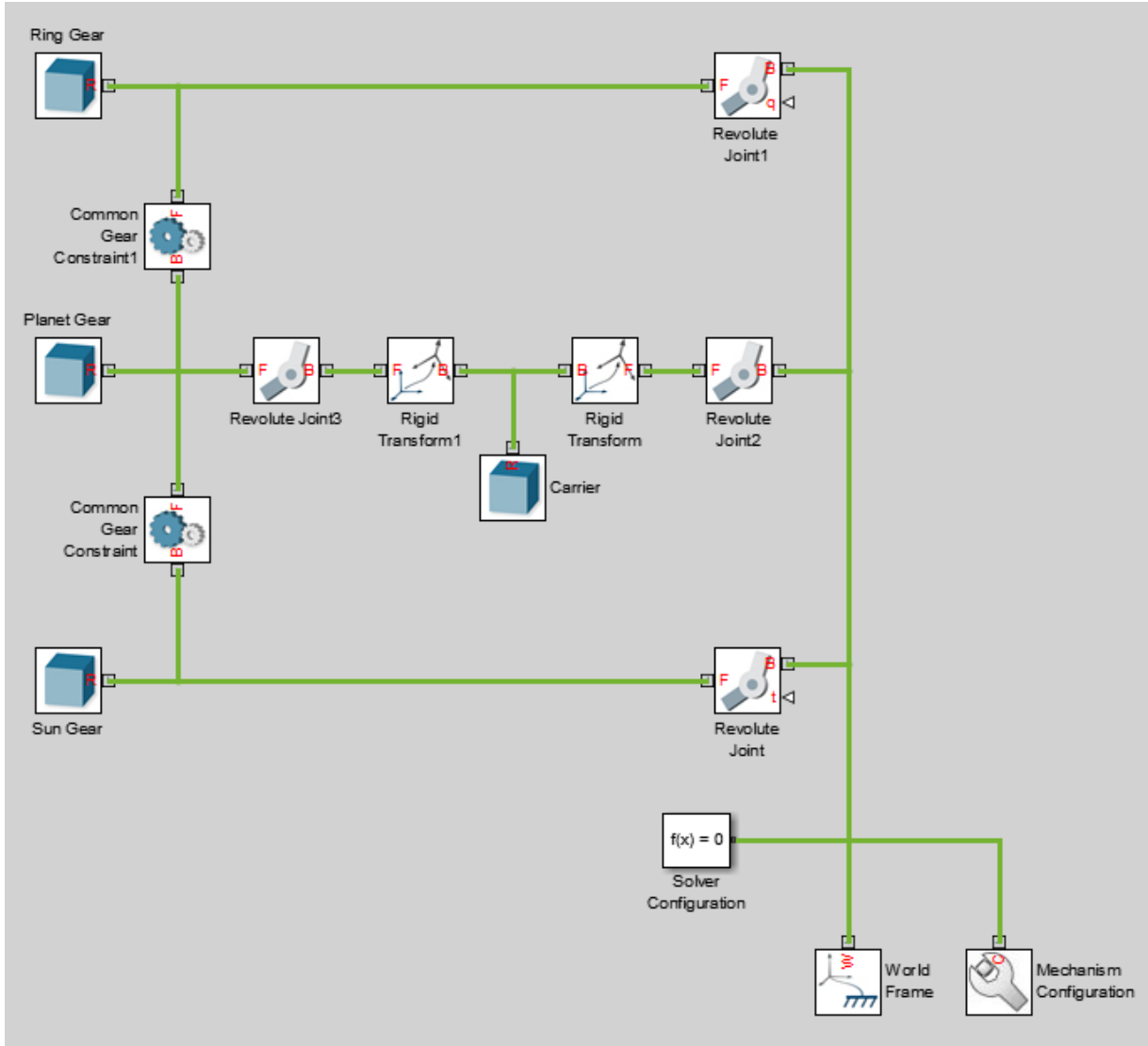

Ring Gear
R
F
B
Revolute
Joint1
Common
Gear
Constraint1
B
F
Planet Gear
R
Common
Gear
Constraint
B
F
R
Sun Gear
F
B
Revolute
Joint
f(x) = 0
Solver
Configuration
W
World
Frame
C
Mechanism
Configuration

2 Add these blocks to the planetary gear model.

Library	Block	Quantity
Body Elements	Solid	1
Joints	Revolute Joint	2
Frames and Transforms	Rigid Transform	2

3 Connect and name the blocks as shown.

Pay close attention to the Rigid Transform block orientation: the B frame ports should face the Solid block. The new blocks are highlighted.



4 In the Carrier block dialog box, specify these parameters.

Parameter	Setting
Geometry > Shape	Select `General Extrusion`.

Parameter	Setting
Geometry > Cross-Section	Enter `Carrier.CS`. Select units of `cm`.
Geometry > Length	Enter `Carrier.T`.
Inertia > Density	Enter `Rho`.
Graphic > Visual Properties > Color	Enter `Carrier.RGB`.

5 In the Rigid Transform block dialog box, specify these parameters.

Parameter	Setting
Translation > Method	Select `Cartesian`.
Translation > Offset	Enter `[Carrier.L/2 0 -(Carrier.T +T)/2]`. Select units of `cm`.

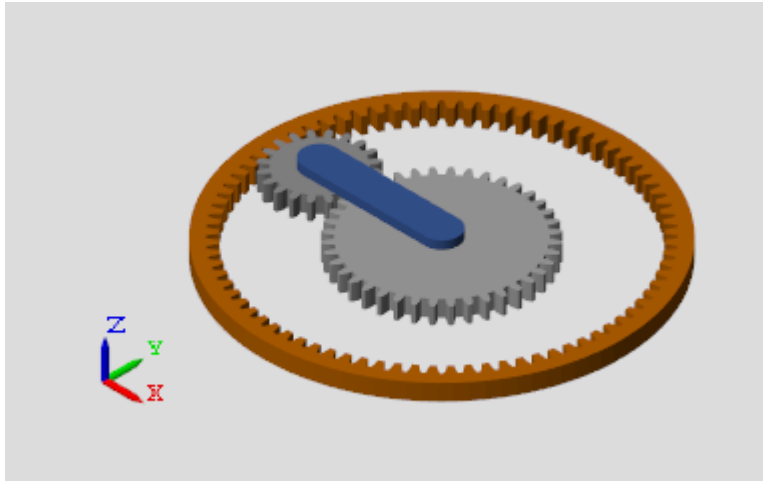
6 In the Rigid Transform1 block dialog box, specify these parameters.

Parameter	Setting
Translation > Method	Select `Cartesian`.
Translation > Offset	Enter `[-Carrier.L/2 0 -(Carrier.T +T)/2]`. Select units of `cm`.

7 In the model workspace, define the Carrier block parameters using MATLAB code:

```
% Gear Carrier Parameters
Carrier.RGB = [0.25 0.4 0.7];
Carrier.L = Sun.R + Planet.R;
Carrier.W = 2*T;
Carrier.T = T/2;

Theta = (90:1:270)'*pi/180;
Beta = (-90:1:90)'*pi/180;

Carrier.CS = [-Carrier.L/2 + Carrier.W/2*cos(Theta) ...
Carrier.W/2*sin(Theta); Carrier.L/2 + Carrier.W/2*cos(Beta), ...
Carrier.W/2*sin(Beta)];
```

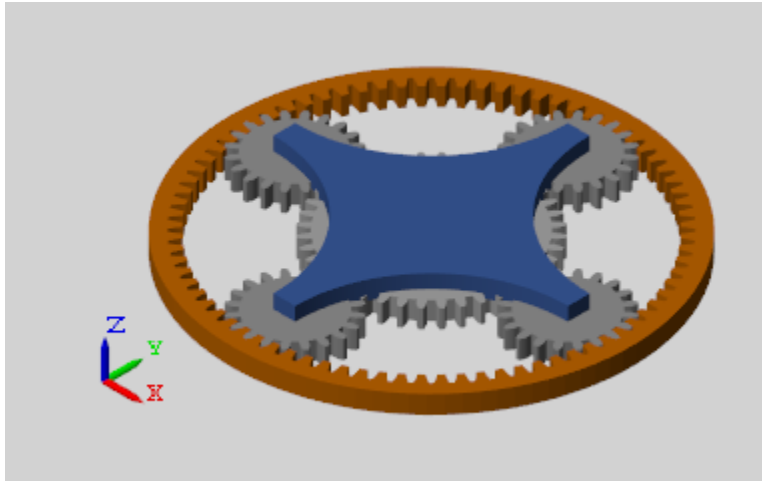
8 Simulate the model. To induce motion, try adjusting the velocity state targets in the joint block dialog boxes. Notice that the gear carrier now performs the task of the Distance Constraint block.



You can open a copy of the resulting model. At the MATLAB command line, enter `smdoc_planetary_gear_d`.

Add More Planet Gears

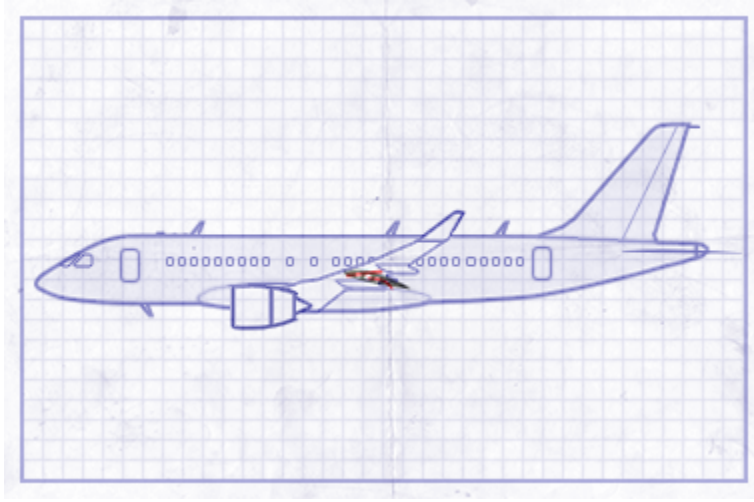
Experiment with the model by adding more planet gears. Remember that you must change the Carrier body to accommodate any additional planet gears. To see an example with four planet gears, at the MATLAB command line enter `smdoc_planetary_gear_e`.


Z
Y
X

Constrain a Point to a Curve

In this section...
“Open the Flap Assembly Model” on page 2-82
“Effect of Constraints on the Model” on page 2-83
“Create the Connection Frames” on page 2-84
“Connect the Constraint Blocks” on page 2-86
“Specify the Flap Constraint Curves” on page 2-87
“Simulate the Model” on page 2-88

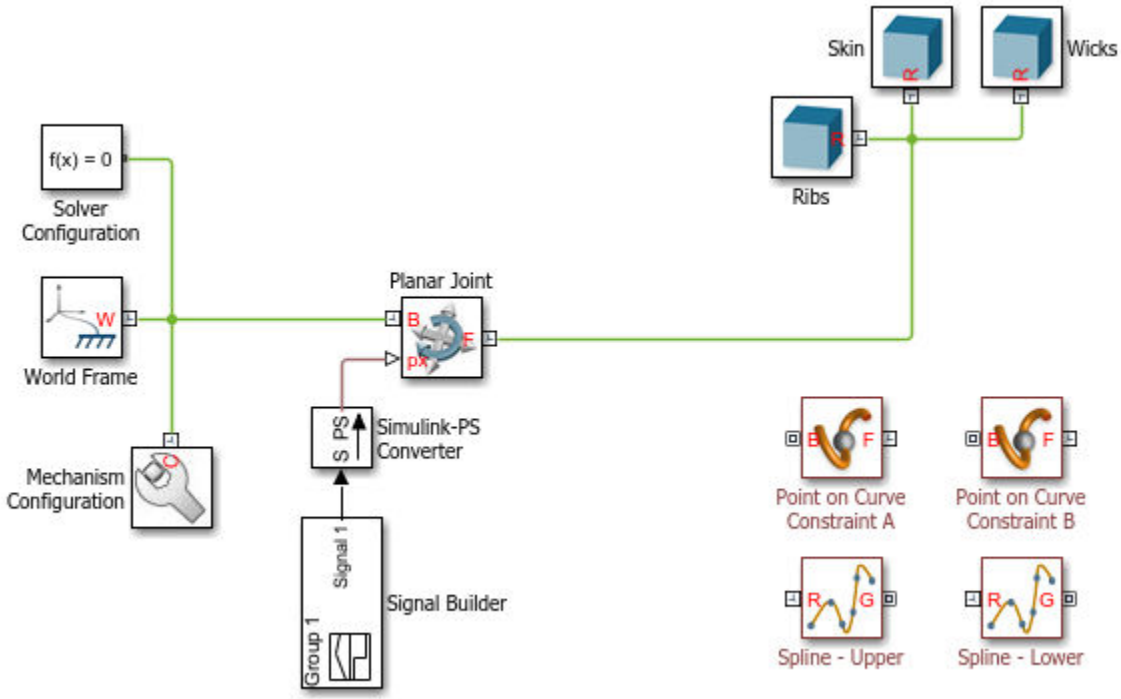
This example shows how to apply multiple point-on-curve constraints to a single body. The example is based on a partial model of an aircraft flap that extends and retracts by riding on curved tracks. You complete the model by adding point-on-curve constraints between two points on the flap and the flap track curves. The figure shows the visualization results of the flap model, once it is complete.



Open the Flap Assembly Model

At the MATLAB command prompt, enter `smdoc_poc_flap_start`. Simscape Multibody software opens the flap assembly model. Save the model with a different name in a convenient folder so that you do not inadvertently overwrite the model.

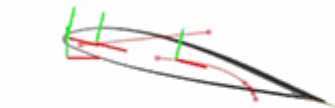
The model is missing key connection lines and block parameters and does yet not simulate. Three rigidly connected blocks, named Ribs, Skin, and Wicks, represent the flap body. A Planar Joint block connects this body to the world frame with three degrees of freedom—one rotational and two translational.



Effect of Constraints on the Model

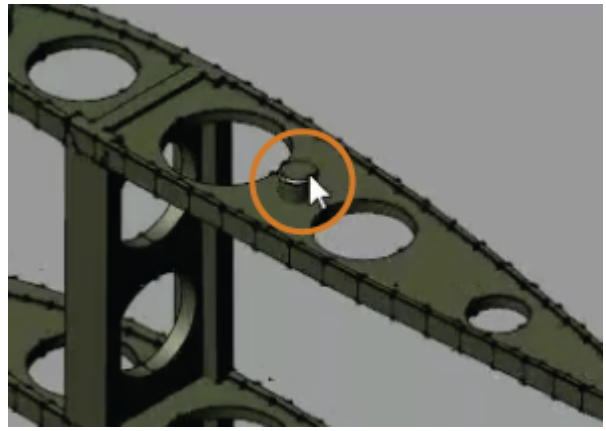
The flap is at this point unconstrained. In this state, the degrees of freedom of the flap are those provided by the Planar Joint block. The flap is free to rotate about the *z*-axis, translate along the *x*-axis, and translate along the *y*-axis. The animated figure shows these motions. The frame axes are color-coded, with red denoting *x*, green *y*, and blue *z*. The *z*-axis points out of the screen is not visible in the figure.

The point-on-curve constraints couple the initially independent motions of the flap so that any one of these motions suffices to completely determine the flap trajectory. For example, in the fully constrained flap, a translation along the *x*-axis determines also the translation along the *y*-axis and the rotation about the *z*-axis. The animated figure shows the constrained motion that you obtain at the end of this example.



Create the Connection Frames

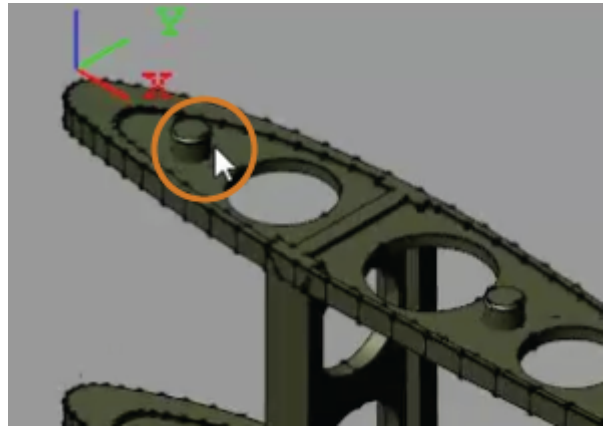
1. In the Ribs Solid block dialog box, expand the **Frames** node and click the button. The frame creation interface opens. You use this interface to create a new frame and set its position and orientation with respect to the solid geometry.
2. In the visualization pane, zoom in on the top portion of the flap and select the top surface of the rightmost cylindrical protrusion, as shown in the figure. The visualization pane highlights the selected surface and reveals its normal vector.



3 In the **Frame Origin** area of the frame creation interface, click the **Based on Geometric Feature** radio button and then the **Use Selected Feature** button. The frame moves to the center of the selected surface.

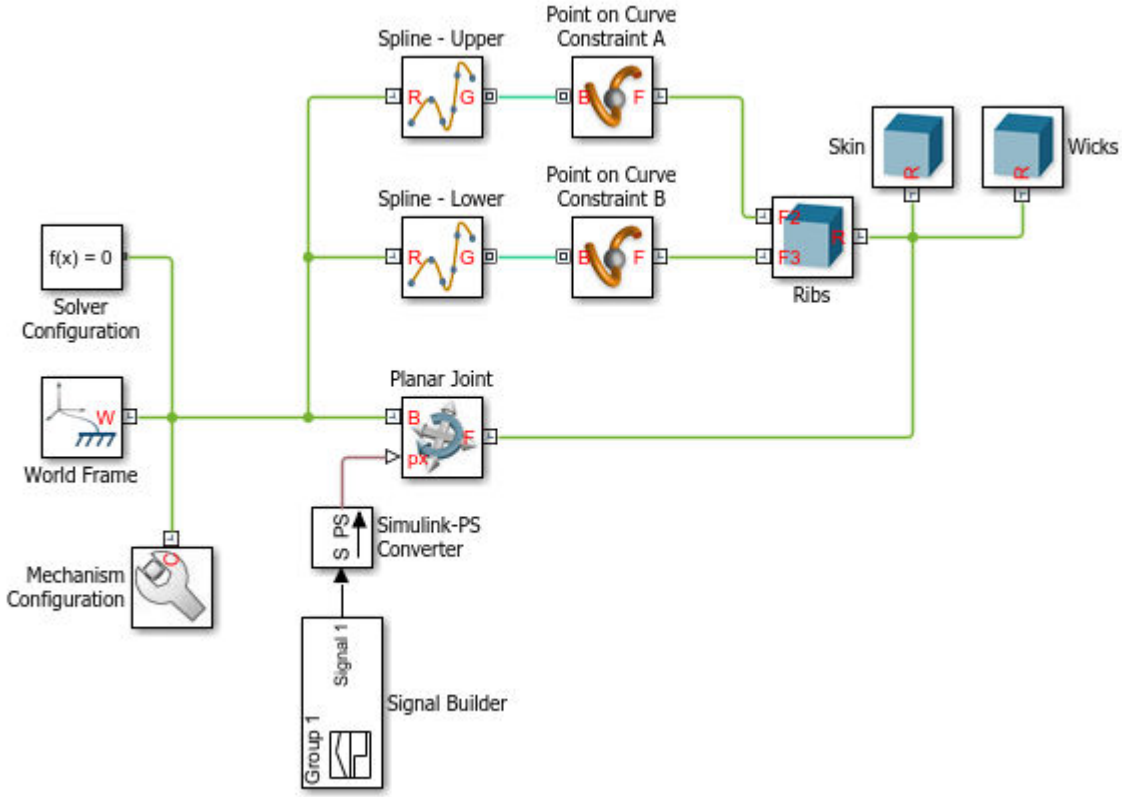
4 Click the **Save** button to add the new frame to the solid. The block exposes a new port, F1, corresponding to the new frame. You can rename the frame anything you want but, in this example, the default name suffices.

5 Repeat steps 1–4 to create a second frame with origin at the top surface of the leftmost cylindrical protrusion, as shown in the figure. The block exposes a new port, F2, corresponding to the new frame.



Connect the Constraint Blocks

1. Connect the frame port of each Spline block to the World Frame block. This connection ensures that the constraint curves specified in the Spline blocks are resolved in the World frame. The curve definitions are currently the block defaults. You later change these defaults to obtain a reasonable flap trajectory.
2. Connect the frame ports of the Point on Curve Constraint blocks to the frame ports that you added to the Ribs Solid block (F1 and F2). The origins of these frames are the points that you constrain in this example.
3. Connect the geometry ports of the Spline blocks to the geometry ports of the Point on Curve Constraint blocks as shown in the figure. The geometry connection lines identify the spline curve definitions as the curves to constrain.



Specify the Flap Constraint Curves

1. Open Model Explorer and, in the **Model Hierarchy** pane, expand the **smdoc_poc_cam_start** > **Model Workspace** node. Model Explorer enables you to define workspace variables so that you can reference them in block dialog box parameters.
2. In the **Model Workspace** pane of Model Explorer, set the **Data source** parameter to `MATLAB Code`. An editable field appears with the still-incomplete variable definitions:

   ```
   theta = linspace(4*pi/12,pi/8,4)';
   lowerTrack = [];
   upperTrack = [];
   ```

   The variables `lowerTrack` and `upperTrack` are the constraint curves of the lower and upper flap tracks. The variable `theta` is an angle range used in the definition of the lower track curve.
3. Complete the `lowerTrack` and `upperTrack` definitions by specifying a few points on the curves as shown in the following code snippet. You must click the **Reinitialize from Source** button to apply the changes to the model. The new code portions are shown in blue.

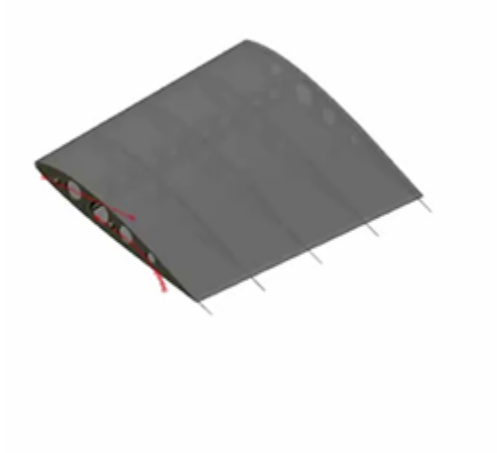
   ```
   theta = linspace(4*pi/12,pi/8,4)';
   lowerTrack = [350 0; (640+100*cos(theta)) (130*sin(theta)-210)];
   upperTrack = [50 50; 550 100];
   ```
4. In the dialog box of each Spline block, specify the parameters listed in the table. The **Interpolation Points** parameter is defined in terms of the MATLAB variables from the model workspace. Ensure that this parameter is in units of `mm`. The **Color** parameter is defined as a normalized RGB vector corresponding to a light shade of red.

Block	Interpolation Points	Graphic > Visual Properties + Color
Spline – Upper	`upperTrack`	`[0.8, 0, 0]`
Spline – Lower	`lowerTrack`	`[0.8, 0, 0]`

5. Update the block diagram, e.g., by selecting **Simulation** > **Update Diagram** from the Simulink menu bar. Mechanics Explorer opens with a static visualization of the model in its initial configuration. The figure shows the spline curves as they appear in Mechanics Explorer with the bodies hidden. You can hide a body by selecting its node in the tree view pane and selecting **Hide This**.

Simulate the Model

Simulate the model. Mechanics Explorer shows a dynamic visualization of the flap assembly. The motion of the flap is now constrained so that the origins of the frames you created always lie on the curves you defined. Switch between the standard views of Mechanics Explorer or rotate, pan, and zoom to better explore the flap and its motion.



To open a complete version of the flap assembly model, at the MATLAB command prompt, enter `smdoc_poc_flap`.

Internal Mechanics, Actuation and Sensing

- "Modeling and Sensing System Dynamics" on page 3-2
- "Modeling Gravity" on page 3-4
- "Model Gravity in a Planetary System" on page 3-9
- "Specifying Joint Actuation Inputs" on page 3-25
- "Joint Actuation Limitations" on page 3-34
- "Actuating and Sensing with Physical Signals" on page 3-36
- "Sensing" on page 3-39
- "Force and Torque Sensing" on page 3-42
- "Motion Sensing" on page 3-46
- "Rotational Measurements" on page 3-51
- "Translational Measurements" on page 3-56
- "Measurement Frames" on page 3-64
- "Sense Motion Using a Transform Sensor Block" on page 3-67
- "Specify Joint Actuation Torque" on page 3-73
- "Analyze Motion at Various Parameter Values" on page 3-85
- "Sense Forces and Torques Acting at Joints" on page 3-91
- "Sense Constraint Forces" on page 3-99
- "Specify Joint Motion Profile" on page 3-105
- "Prescribe Joint Motion in Planar Manipulator Model" on page 3-110

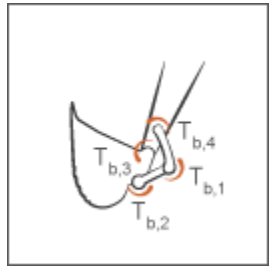
Modeling and Sensing System Dynamics

In this section...
“Provide Joint Actuation Inputs” on page 3-2
“Specify Joint Internal Mechanics” on page 3-2
“Model Body Interactions and External Loads” on page 3-3
“Sense Dynamical Variables” on page 3-3

Provide Joint Actuation Inputs

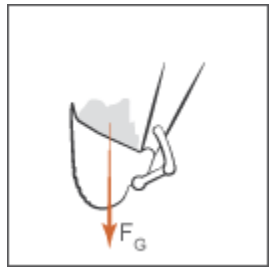
Identify the joints to actuate and the actuation type to use. Then, model the actuation inputs as time-varying physical signals and connect them to the various joints. See “Specify Joint Actuation Torque” on page 3-73 for an example.

Specify Joint Internal Mechanics



Model damping and spring behavior at joints. Specify joint damping coefficients to model energy dissipation and joint spring constants to model energy storage.

Model Body Interactions and External Loads



Identify the forces and torques acting at or between bodies not connected by joints. Model these forces and torques explicitly using Forces and Torques blocks. See "Model Gravity in a Planetary System" on page 3-9 for an example.

Sense Dynamical Variables

Identify the forces, torques, and motion variables to sense. You can sense these variables at joints through Joint blocks. You can also sense motion variables using the Transform Sensor block. See "Sense Motion Using a Transform Sensor Block" on page 3-67 for an example.

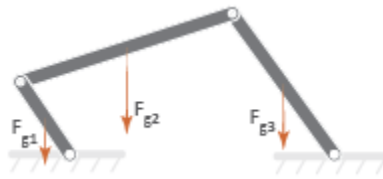
Modeling Gravity

In this section...
“Gravity Models” on page 3-4
“Gravitational Force Magnitude” on page 3-5
“Force Position and Direction” on page 3-6
“Gravitational Torques” on page 3-7

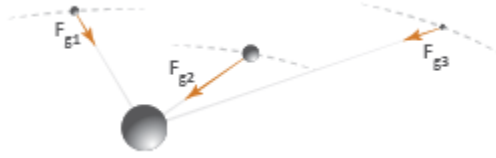
Gravity Models

Gravity influences motion in many natural and engineered systems. These range in scale from the very large, such as the planets orbiting the sun, to the relatively small, such as the shock absorbers damping gravity-driven oscillations in a car. In Simscape Multibody, you can add gravity to systems like these using three gravity models:

- Uniform gravity, as experienced by most earthbound systems. The force on each body due to uniform gravity depends only on its mass. This force is the same everywhere in space for a given body, though it can vary in time. You model uniform gravity using the Mechanism Configuration block.



- Gravitational field, as experienced by the planets in the solar system. The force on each body due to a gravitational field depends not only on its mass but also on its inverse square distance to the field origin. You model a gravitational field using the Gravitational Field block.



- Inverse-square law force pair, similar in nature to a gravitational field, but acting exclusively between one pair of bodies. You model an inverse-square law force pair

using the Inverse Square Law Force block. You must specify the body masses and force constants explicitly.

Gravitational Force Magnitude

The force of gravity is an inverse-square law force—that is, one that decays with the square distance from the field origin to the target body. The magnitude of this force, F_g, follow from Newton's law of universal gravitation which, for two bodies of mass M and m a distance R apart, states

$$F_g = -G\frac{Mm}{R^2}$$

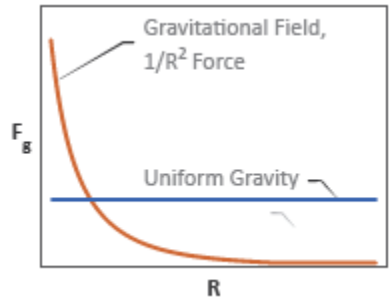
with G being the gravitational constant. This is the force that you model when you represent gravity through Gravitational Field or Inverse Square Law Force blocks. If the distance between source and target masses is constant, the gravitational force reduces to a simpler form,

$$F_g = -mg$$

with g being the nominal gravitational acceleration. Near the surface of the Earth, at a distance equal to Earth's radius from the gravitational field origin, the nominal acceleration equals

$$g = \frac{GM}{R^2} \approx 9.80665\, m/s^2$$

This is the gravitational force that you model when you represent gravity through the Mechanism Configuration block. The figure shows how the magnitude of the gravitational force (F_g) varies with distance (R) for a given body under uniform gravity, a gravitational field, and an inverse-square law force pair.



Force Position and Direction

In a physical system, the force due to a gravitational field acts at a body's center of mass —automatically computed during simulation—along the imaginary line connecting the field origin to the center of mass. These are also the application point and direction of gravity that the Gravitational Field block provides. See "Modeling Bodies" on page 1-5 for more information on how Simscape Multibody defines a body subsystem.

Far from the field origin, the field origin-center of mass line remains approximately constant at small-to-moderate displacements, and the force of gravity behaves as if its direction were fixed. This is the approximation used in the Mechanism Configuration block. Gravity still acts at each body's center of mass, but its direction is now fixed along the gravity vector that you specify.

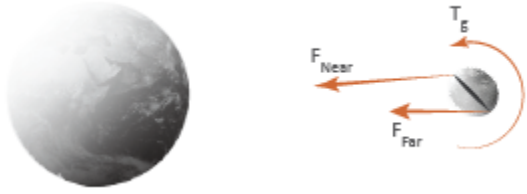
If you want to model the effects of gravity on a point other than a body's center of mass, you can add a frame at the desired location and apply a gravitational force directly at that frame. You model the force using the Inverse Square Law Force block. This force points along the imaginary line between the two body frames that the Inverse Square Law Force block connects.

The table summarizes the application point and direction of gravity provided by the different blocks.

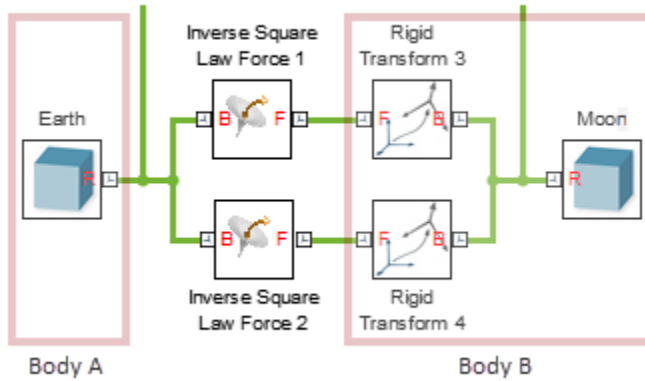
Block	Position	Direction
Mechanism Configuration	Center of Mass	Specified gravity vector
Gravitational Field	Center of Mass	Field origin-center of mass line
Inverse Square Law Force	Connection frames	Base-follower frame line

Gravitational Torques

A gravitational torque can arise in a large body immersed in a nonuniform gravitational field. The lemon-shaped moon, with its near end perpetually facing Earth, is one example. Being placed at different distances from Earth, the near and far elongated ends experience dissimilar gravitational forces, resulting in a net gravitational torque if the line between the two ends ever falls out of alignment with the center of the Earth.



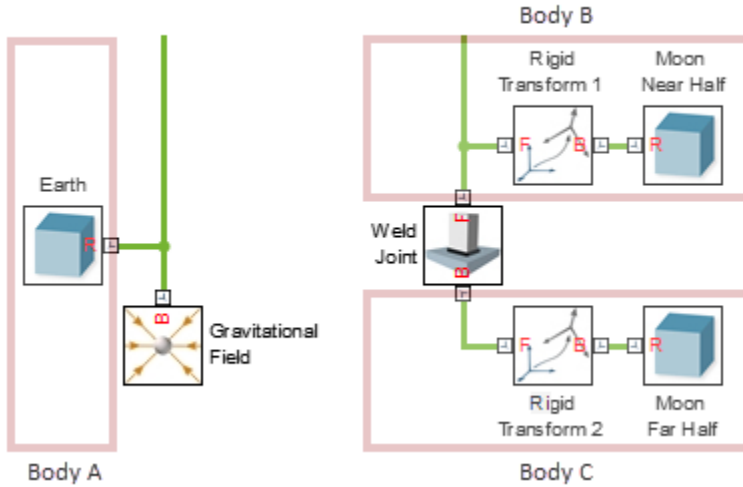
You can model such torques in Simscape Multibody by modeling the different gravitational forces acting on a body. You do this using the Inverse Square Law Force or Gravitational Field block. If you use the Inverse Square Law Force block, you must create additional frames in each body whose response to gravitational torque you want to model. You must then apply a gravitational force to each frame explicitly. The figure shows an example.



Torque on the moon due to dissimilar gravitational forces at the elongated ends

If you use the Gravitational Field block, you must split each body into discrete sections and connect them through Weld Joint blocks. The Gravitational Field block

automatically applies a force at the center of mass of each section, approximating the compound effect of the different gravitational forces on the body—which in this case is treated as a rigid multibody system. The figure shows an example.



Torque on the moon due to dissimilar gravitational forces at the elongated ends

Model Gravity in a Planetary System

In this section...
“Model Overview” on page 3-9
“Step 1: Start a New Model” on page 3-10
“Step 2: Add the Solar System Bodies” on page 3-11
“Step 3: Add the Degrees of Freedom” on page 3-15
“Step 4: Add the Initial State Targets” on page 3-17
“Step 5: Add the Gravitational Fields” on page 3-21
“Step 6: Configure and Run the Simulation” on page 3-23
“Open an Example Model” on page 3-24

Model Overview

This tutorial shows how to simulate the gravity-driven orbits of the major solar system bodies. The model treats the sun and planets as perfect spheres each with three translational degrees of freedom. Planet spin is ignored. Gravitational fields generate the forces that keep the planets in orbit.

Solid blocks represent the solar system bodies and provide their geometries, inertias, and colors. Cartesian Joint blocks define the bodies’ degrees of freedom relative to the world frame, located at the solar system barycenter. Gravitational Field blocks add the long-range forces responsible for bending the initial planet trajectories into closed elliptical orbits.

The Cartesian Joint blocks provide the initial states—positions and velocities—of the sun and planets relative to the world frame. The initial states correspond to the solar system

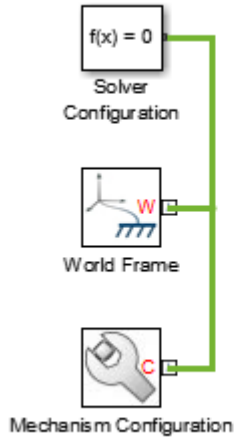
configuration on Jun 20th, 2016. They are sourced from ephemeris databases maintained by the Jet Propulsion Laboratory (JPL).

You can query the databases through the JPL Horizons system using a web or telnet interface. Aerospace Toolbox users can alternatively obtain the ephemeris data at the MATLAB command prompt using the `planetEphemeris` function after installing the Aerospace Ephemeris Data support package.

Step 1: Start a New Model

Open the Simscape Multibody model template and remove all unnecessary blocks. Modify the gravity settings so that you can add gravitational fields to the model. The result provides a starting point for the solar system model.

1. At the MATLAB command prompt, enter `smnew`. MATLAB opens a model template with commonly used blocks and suitable solver settings for Simscape Multibody models.
2. Cut all but the Mechanism Configuration, Solver Configuration, and World Frame blocks. These three blocks provide the model with gravity settings, solver settings, and a global inertia reference frame.



3 In the Mechanism Configuration block dialog box, set **Uniform Gravity** to `None`. This setting enables you to model gravity as an inverse-square law force using Gravitational Field blocks instead.

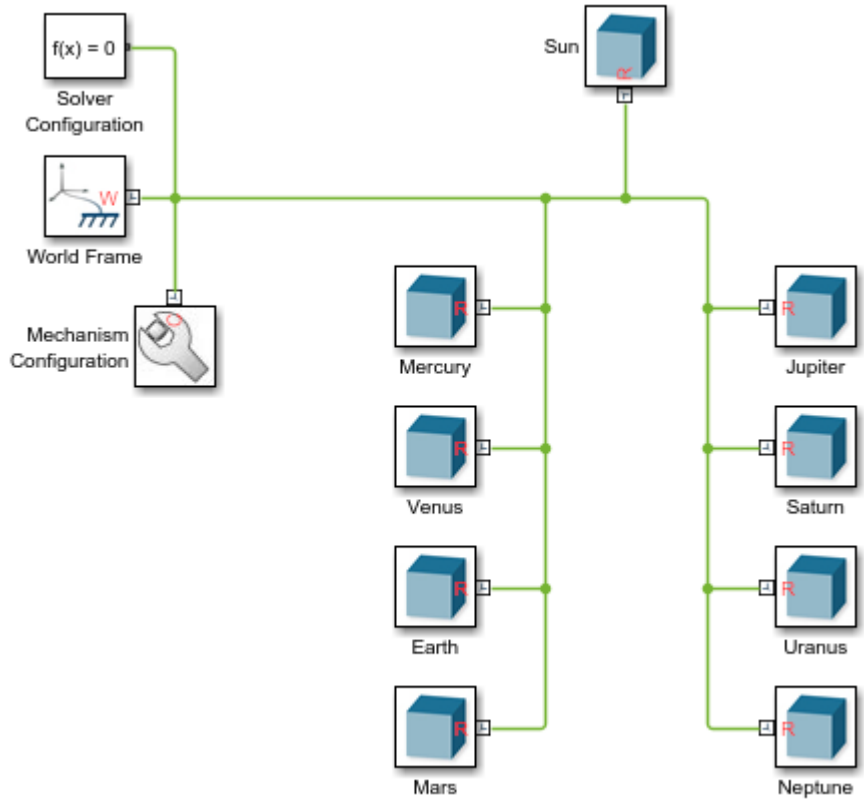
Step 2: Add the Solar System Bodies

Represent the solar system bodies using Solid blocks. Specify the geometry and inertia parameters in terms of MATLAB variables and initialize these variables in the model workspace using Model Explorer. The variables are data structures named after the solar system bodies using proper-noun capitalization.

Connect and Configure the Solid Blocks

1 Add to the model nine Solid blocks from the Body Elements library. The blocks represent the sun and eight known planets.

2 Connect and name the blocks as shown in the figure. The branched frame connection line between the world frame and the planets makes them rigidly connected and coincident in space. You later change this condition using Cartesian Joint blocks.



3 In the Solid block dialog boxes, set the **Geometry > Shape** parameter to `Sphere` and the **Inertia > Based on** parameter to `Mass`. The inertia parameter setting enables you to specify the solid mass directly so that you can scale the planet shapes without affecting the model dynamics.

4 Specify the following Solid block parameters in terms of MATLAB data structure fields. Enter the field names in the format *Structure.Field*, where *Structure* is the title-case name of the solar system body and *Field* is the string shown in the table—e.g., `Sun.R` or `Earth.RGB`.

Block Parameter	Field String
Geometry > Radius	`R`
Inertia > Mass	`M`

See Also

Related Examples

- “Sense Motion Using a Transform Sensor Block” on page 3-67
- “Specify Joint Actuation Torque” on page 3-73

More About

- “Motion Sensing” on page 3-46
- “Rotational Measurements” on page 3-51
- “Measurement Frames” on page 3-64

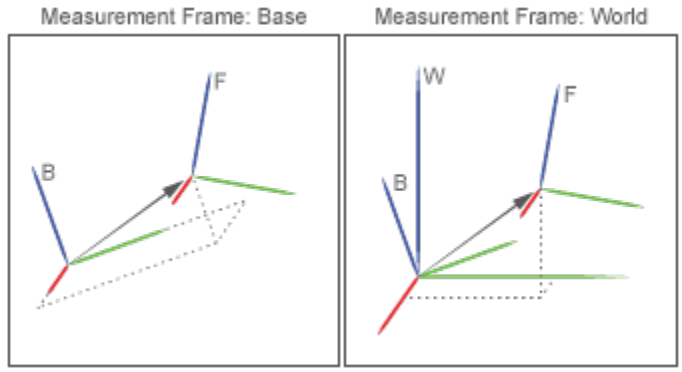
Measurement Frames

In this section...
“Measurement Frame Purpose” on page 3-64
“Measurement Frame Types” on page 3-65

You can sense the spatial relationship between two frames. When you do so, Simscape Multibody resolves the measurement in a measurement frame. For most joint blocks, the measurement frame is the base frame. However, if you use either Transform Sensor or a joint block with a spherical primitive, you can select a different measurement frame. Measurement frames that you can select include `Base`, `Follower`, and `World`. The Transform Sensor block adds the choice between rotating and non-rotating versions of the base and follower frames.

Measurement Frame Purpose

The measurement frame defines the axes that Simscape Multibody uses to resolve a measurement. The measurement still describes the relationship between base and follower frames. However, the X, Y, and Z components of that measurement refer to the axes of the measurement frame. Simscape Multibody takes the measurement and projects it onto the axes of the measurement frame that you select. The figure illustrates the measurement frame concept.



The arrow connecting base and follower frame origins is the translation vector. If you select the base frame as the measurement frame, Simscape Multibody resolves that translation vector along the axes of the base frame. If you select the World frame as the

measurement frame, Simscape Multibody instead resolves the translation vector along the axes of the World frame. The translation vector remains the same, but the frame Simscape Multibody expresses that measurement in changes.

Note that you can select the measurement frame only with certain blocks. Among joint blocks, only those with a spherical primitive offer a selection of measurement frames. All other joint blocks resolve their measurements in the base frame. The Transform Sensor block offers the most extensive selection of measurement frames.

Measurement Frame Types

Simscape Multibody offers five different measurement frames. These include World as well as rotating and non-rotating versions of the base and follower frames. The table describes these measurement frames.

Measurement Frame	Description
World	Inertial frame at absolute rest. World is the ultimate reference frame in a model. The World Frame block identifies this frame in a model.
Base	Frame that connects to the B port of the sensing block. Unless you rigidly connect it to World, Base is generally non-inertial.
Follower	Frame that connects to the F port of the sensing block. Unless you rigidly connect it to World, Follower is generally non-inertial.
Non-Rotating Base/Follower	Non-rotating versions of the Base and follower frames. A non-rotating frame is a virtual frame which, at every point in time, Simscape Multibody holds coincident with the rotating frame, but which has zero angular velocity with respect to the World frame. Measurements that can differ between rotating and non-rotating frames are the linear velocity and linear acceleration.

See Also

Related Examples

- “Sense Motion Using a Transform Sensor Block” on page 3-67
- “Specify Joint Actuation Torque” on page 3-73

More About

- “Motion Sensing” on page 3-46
- “Rotational Measurements” on page 3-51
- “Translational Measurements” on page 3-56

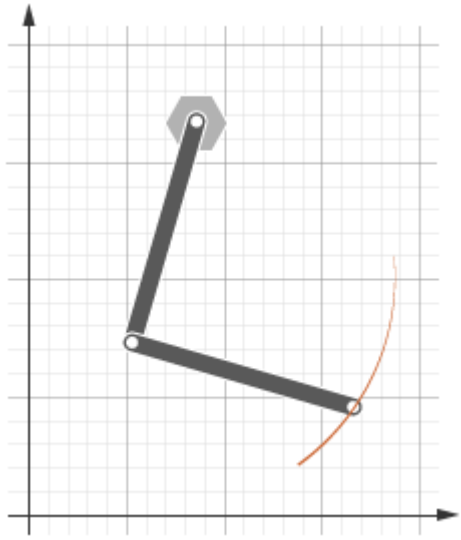
Sense Motion Using a Transform Sensor Block

In this section...
“Model Overview” on page 3-67
“Modeling Approach” on page 3-68
“Build Model” on page 3-68
“Guide Model Assembly” on page 3-70
“Simulate Model” on page 3-70
“Save Model” on page 3-72

Model Overview

The Transform Sensor block provides the broadest motion-sensing capability in Simscape Multibody models. Using this block, you can sense motion variables between any two frames in a model. These variables can include translational and rotational position, velocity, and acceleration.

In this example, you use a Transform Sensor block to sense the lower link translational position with respect to the World frame. You output the position coordinates directly to the model workspace, and then plot these coordinates using MATLAB commands. By varying the joint state targets, you can analyze the lower-link motion under quasi-periodic and chaotic conditions.



Modeling Approach

In this example, you rely on gravity to cause the double pendulum to move. You displace the links from equilibrium and then let gravity act on them. To displace the links at time zero, you use the **State Targets** section of the Revolute Joint block dialog box. You can specify position or velocity. When you are ready, you simulate the model to analyze its motion.

To sense motion, you use the Transform Sensor block. First, you connect the base and follower frame ports to the World Frame and lower link subsystem blocks. By connecting the ports to these blocks, you can sense motion in the lower link with respect to the World frame. Then, you select the translation parameters to sense. By selecting **Y** and **Z**, you can sense translation along the Y and Z axes, respectively. You can plot these coordinates with respect to each other and analyze the motion that they reveal.

Build Model

1 At the MATLAB command prompt, enter `smdoc_double_pendulum`. A double pendulum model opens up. For instructions on how to create this model, see “Model an Open-Loop Kinematic Chain” on page 2-16.

2 Drag these blocks into the model to sense motion.

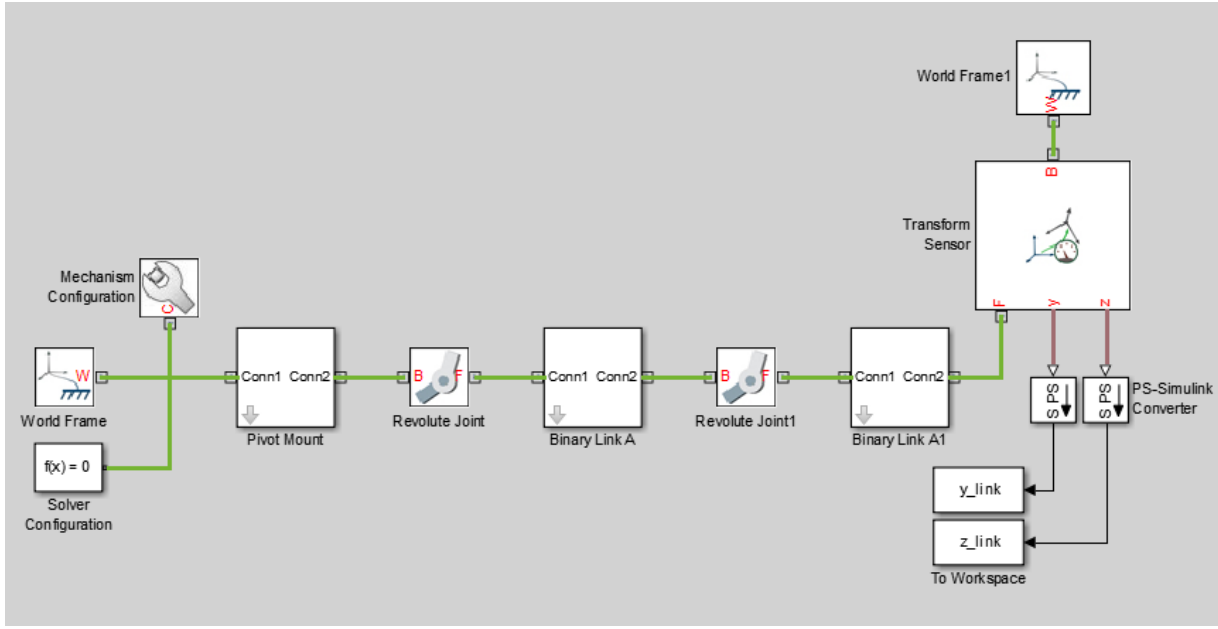
Library	Block	Quantity
Simscape > Multibody > Frames and Transforms	Transform Sensor	1
Simscape > Multibody > Frames and Transforms	World Frame	1
Simscape > Utilities	PS-Simulink Converter	2
Simulink > Sinks	To Workspace	2

3 In the Transform Sensor block dialog box, select **Translation > Y** and **Translation > Z**. The block exposes two physical signal output ports, labeled y and z.

4 In the PS-Simulink Converter blocks, specify units of `cm`.

5 In the To Workspace blocks, enter the variable names `y_link` and `z_link`.

6 Connect the blocks to the model as shown in the figure. You must connect the base frame port of the Transform Sensor block to the World Frame block. The new blocks are shaded gray.



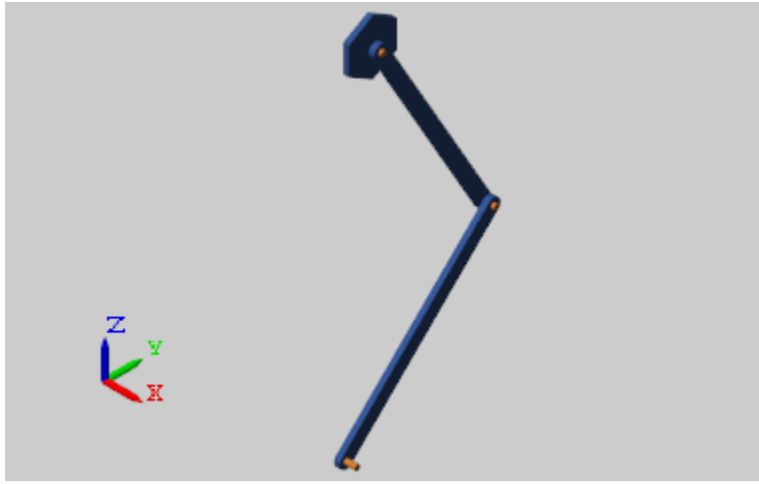
Guide Model Assembly

Specify the initial state of each joint. Later, you can modify this state to explore different motion types. For the first iteration, rotate only the top link by a small angle.

1. In the Revolute Joint block dialog box, select **State Targets** > **Specify Position Target**.
2. Set **Value** to `10 deg`.
3. In the Revolute Joint1 block dialog box, check that **State Targets** > **Specify Position Target** is cleared.

Simulate Model

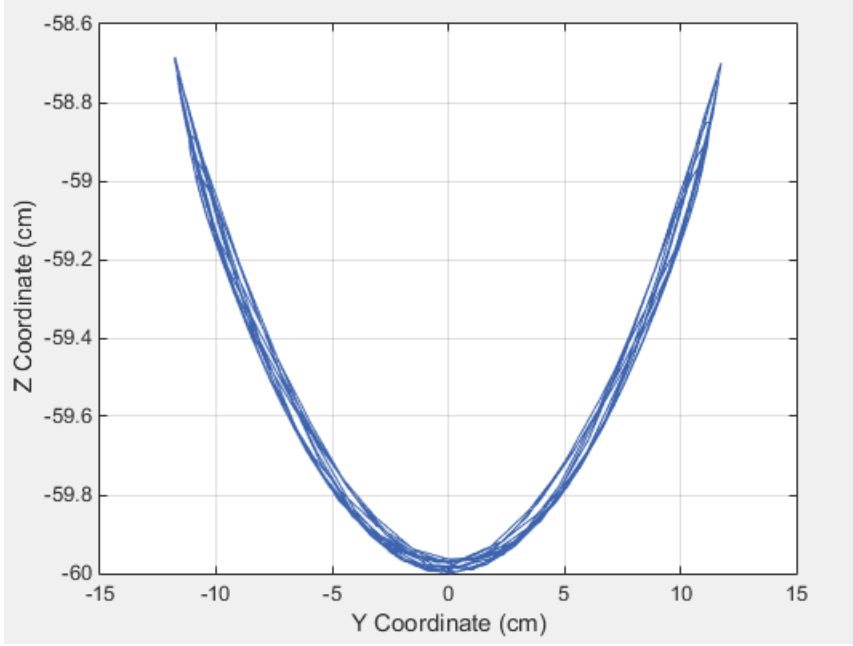
Run the simulation. Mechanics Explorer plays a physics-based animation of the double pendulum assembly.



You can now plot the position coordinates of the lower link. To do this, at the MATLAB command line, enter:

```
figure;
plot(y_link.data, z_link.data, 'color', [60 100 175]/255);
xlabel('Y Coordinate (cm)');
ylabel('Z Coordinate (cm)');
grid on;
```

The figure shows the plot that opens. This plot shows that the lower link path varies only slightly with each oscillation. This behavior is characteristic of quasi-periodic systems.



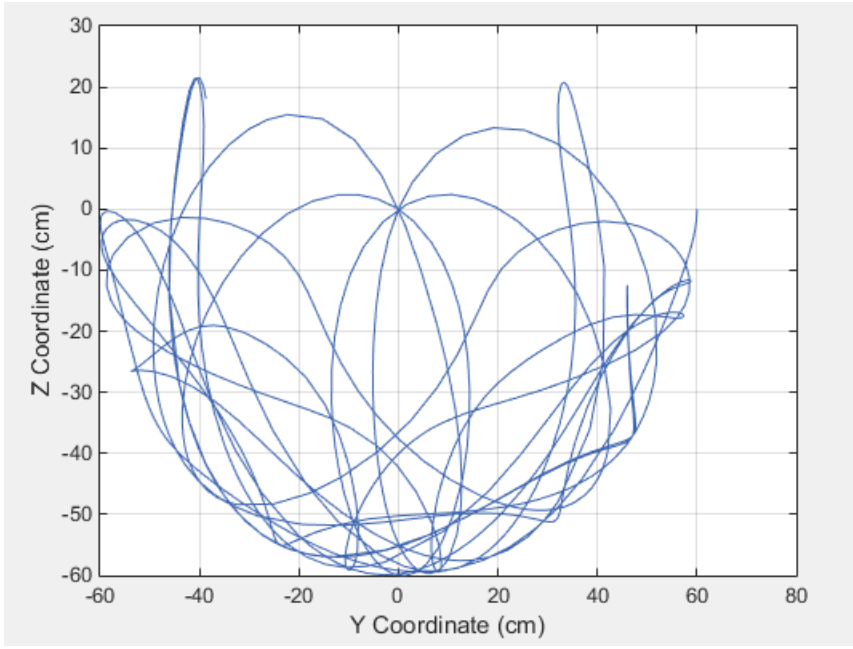
Simulate Chaotic Motion

By adjusting the revolute joint state targets, you can simulate the model under chaotic conditions. One way to obtain chaotic motion is to rotate the top revolute joint by a large angle. To do this, in the Revolute Joint dialog box, change **State Targets** > **Position** > **Value** to `90` and click **OK**.

Simulate the model with the new joint state target. To plot the position coordinates of the lower pendulum link with respect to the world frame, at the MATLAB command prompt, enter this code:

```
figure;
plot(y_link.data, z_link.data, 'color', [60 100 175]/255);
xlabel('Y Coordinate (cm)');
ylabel('Z Coordinate (cm)');
grid on;
```

The figure shows the plot that opens.



Save Model

Save the model in a convenient folder under the name `double_pendulum_sensing`. You reuse this model in a subsequent tutorial, “Prescribe Joint Motion in Planar Manipulator Model” on page 3-110.

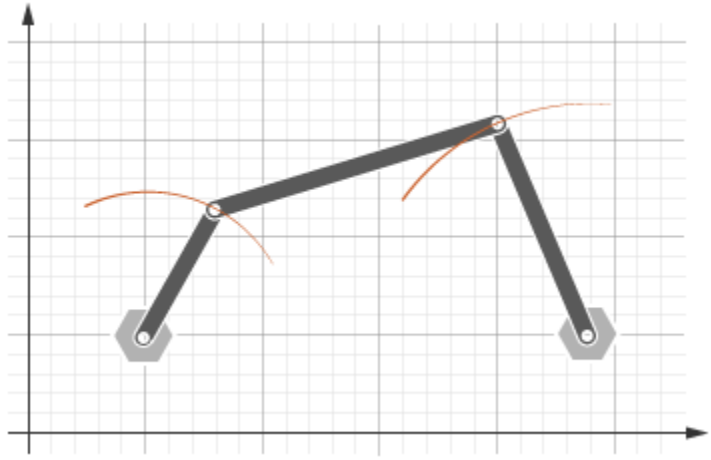
Specify Joint Actuation Torque

In this section...
“Model Overview” on page 3-73
“Four-Bar Linkages” on page 3-74
“Modeling Approach” on page 3-76
“Build Model” on page 3-77
“Simulate Model” on page 3-80

Model Overview

In Simscape Multibody, you actuate a joint directly using the joint block. Depending on the application, the joint actuation inputs can include force/torque or motion variables. In this example, you prescribe the actuation torque for a revolute joint in a four-bar linkage model.

Transform Sensor blocks add motion sensing to the model. You can plot the sensed variables and use the plots for kinematic analysis. In this example, you plot the coupler curves of three four-bar linkage types: crank-rocker, double-crank, and double-rocker.



Four-Bar Linkages

The four-bar linkage contains four links that interconnect with four revolute joints to form a planar closed loop. This linkage converts the motion of an input link into the motion of an output link. Depending on the relative lengths of the four links, a four-bar linkage can convert rotation into rotation, rotation into oscillation, or oscillation into oscillation.

Links

Links go by different names according to their functions in the four-bar linkage. For example, coupler links transmit motion between crank and rocker links. The table summarizes the different link types that you may find in a four-bar linkage.

Link	Motion
Crank	Revolves with respect to the ground link
Rocker	Oscillates with respect to the ground link
Coupler	Transmits motion between crank and rocker links
Ground	Rigidly connects the four-bar linkage to the world or another subsystem

It is common for links to have complex shapes. This is especially true of the ground link, which may be simply the fixture holding the two pivot mounts that connect to the crank or rocker links. You can identify links with complex shapes as the rigid span between two adjacent revolute joints. In example “Model a Closed-Loop Kinematic Chain” on page 2-21, the rigid span between the two pivot mounts represents the ground link.

Linkages

The type of motion conversion that a four-bar linkage provides depends on the types of links that it contains. For example, a four-bar linkage that contains two crank links converts rotation at the input link into rotation at the output link. This type of linkage is known as a double-crank linkage. Other link combinations provide different types of motion conversion. The table describes the different types of four-bar linkages that you can model.

Linkage	Input-Output Motion
Crank-rocker	Continuous rotation-oscillation (and vice-versa)

Linkage	Input-Output Motion
Double-Crank	Continuous rotation-continuous rotation
Double-rocker	Oscillation-oscillation

Grashof Condition

The Grashof theorem provides the basic condition that the four-bar linkage must satisfy so that at least one link completes a full revolution. According to this theorem, a four-bar linkage contains one or more crank links if the combined length of the shortest and longest links does not exceed the combined length of the two remaining links. Mathematically, the Grashof condition is:

$s+l \leq p+q$

where:

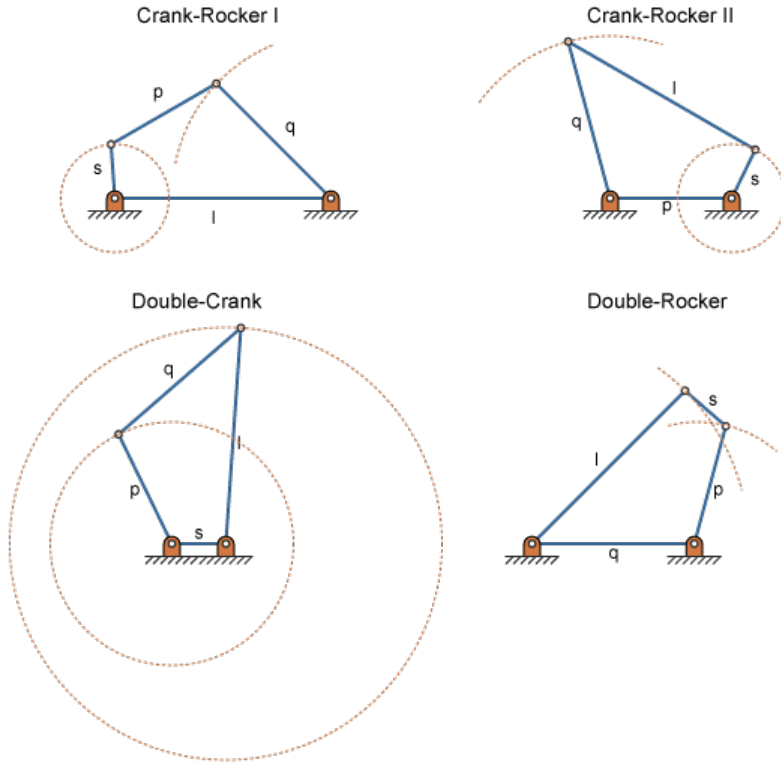
- s is the shortest link
- l is the longest link
- p and q are the two remaining links

Grashof Linkages

A Grashof linkage can be of three different types:

- Crank-rocker
- Double-crank
- Double-rocker

By changing the ground link, you can change the Grashof linkage type. For example, by assigning the crank link of a crank-rocker linkage as the ground link, you obtain a double-crank linkage. The figure shows the four linkages that you obtain by changing the ground link.



Modeling Approach

In this example, you perform two tasks. First you add a torque actuation input to the model. Then, you sense the motion of the crank and rocker links with respect to the World frame. The actuation input is a torque that you apply to the joint connecting the base to the crank link. Because you apply the torque at the joint, you can add this torque directly through the joint block. The block that you add the actuation input to is called Base-Crank Revolute Joint.

You add the actuation input to the joint block through a physical signal input port. This port is hidden by default. To display it, you must select `Provided by Input` from the **Actuation > Torque** drop-down list.

You can then specify the torque value using either Simscape or Simulink blocks. If you use Simulink blocks, you must use the Simulink-PS Converter block. This block converts the Simulink signal into a physical signal that Simscape Multibody can use. For more information, see “Actuating and Sensing with Physical Signals” on page 3-36.

To sense crank and rocker link motion, you use the Transform Sensor block. With this block, you can sense motion between any two frames in a model. In this example, you use it to sense the [Y Z] coordinates of the crank and rocker links with respect to the World frame.

The physical signal output ports of the Transform Sensor blocks are hidden by default. To display them, you must select the appropriate motion outputs. Using the PS-Simulink Converter, you can convert the physical signal outputs into Simulink signals. You can then connect the resulting Simulink signals to other Simulink blocks.

In this example, you output the crank and rocker link coordinates to the workspace using Simulink To Workspace blocks. The output from these blocks provide the basis for phase plots showing the different link paths.

Build Model

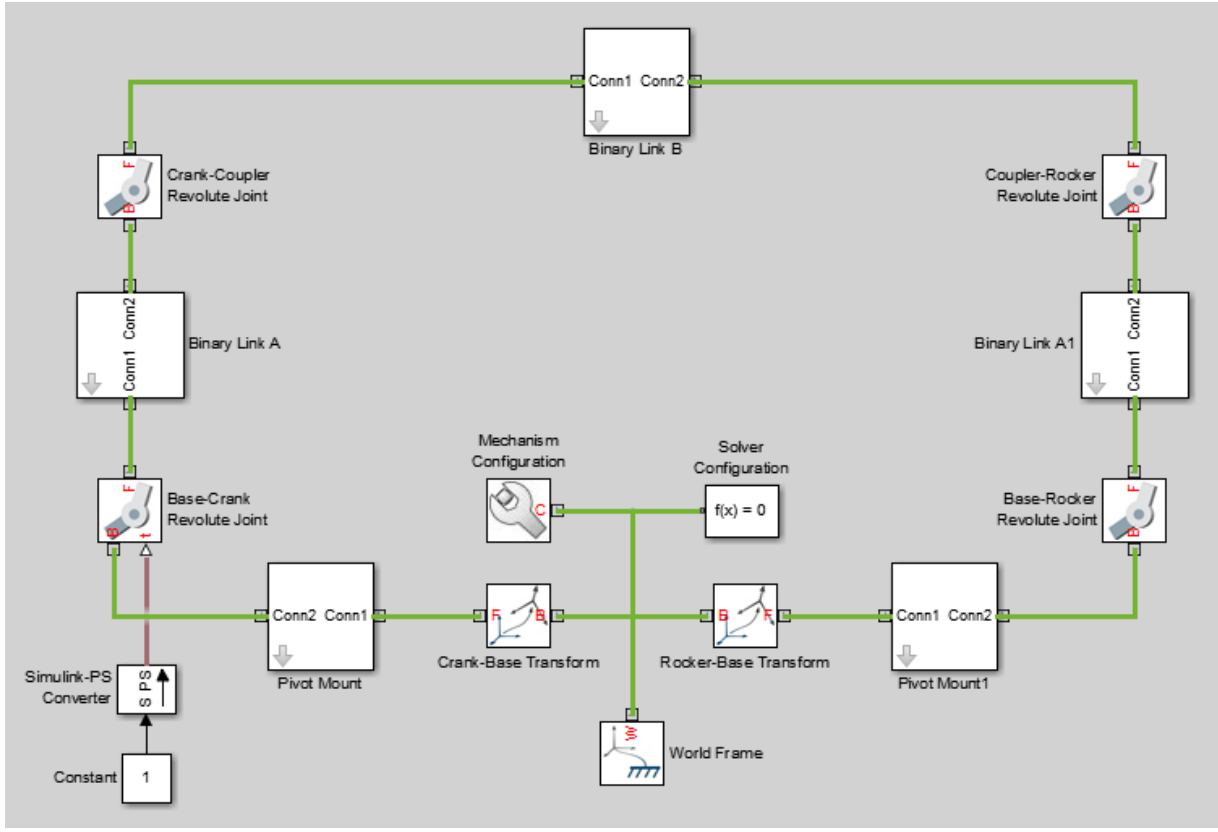
Provide the joint actuation input, specify the joint internal mechanics, and sense the position coordinates of the coupler link end frames.

Provide Joint Actuation Input

1. At the MATLAB command prompt, enter `smdoc_four_bar`. A four bar model opens up. For instructions on how to create this model, see “Model a Closed-Loop Kinematic Chain” on page 2-21.
2. In the Base-Crank Revolute Joint block dialog box, in the **Actuation** > **Torque** drop-down list, select `Provided by Input`. The block exposes a physical signal input port, labeled t.
3. Drag these blocks into the model. The blocks enable you to specify the actuation torque signal.

Library	Block
Simulink > **Sources**	Constant
Simscape > **Utilities**	Simulink-PS Converter

4. Connect the blocks as shown in the figure. The new blocks are shaded gray.



Specify Joint Internal Mechanics

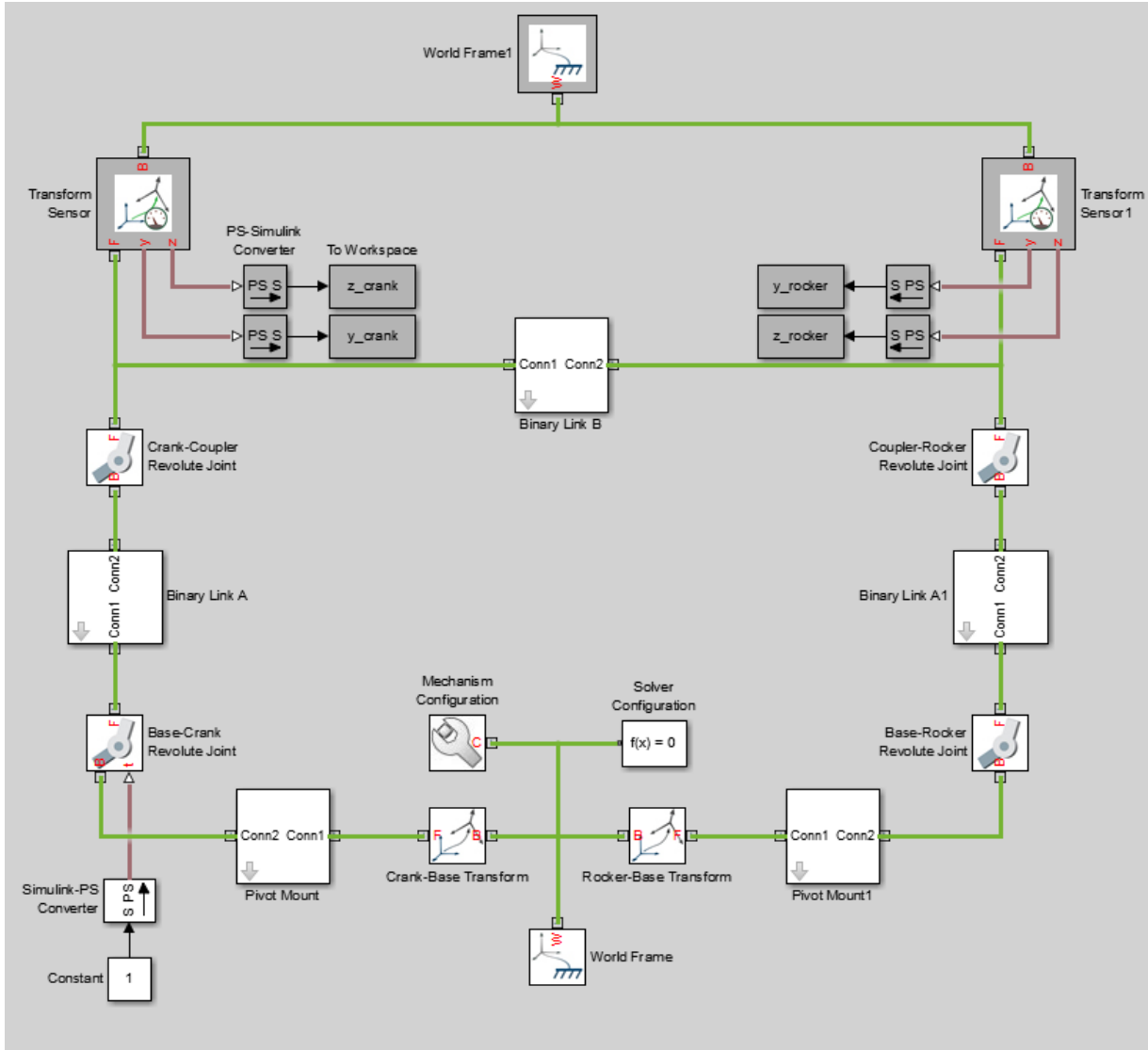
Real joints dissipate energy due to damping. You can specify joint damping directly in the block dialog boxes. In each Revolute Joint block dialog box, under **Internal Mechanics > Damping Coefficient**, enter `5e-4` and press **OK**.

Sense Link Position Coordinates

1 Add these blocks to the model. The blocks enable you to sense frame position during simulation.

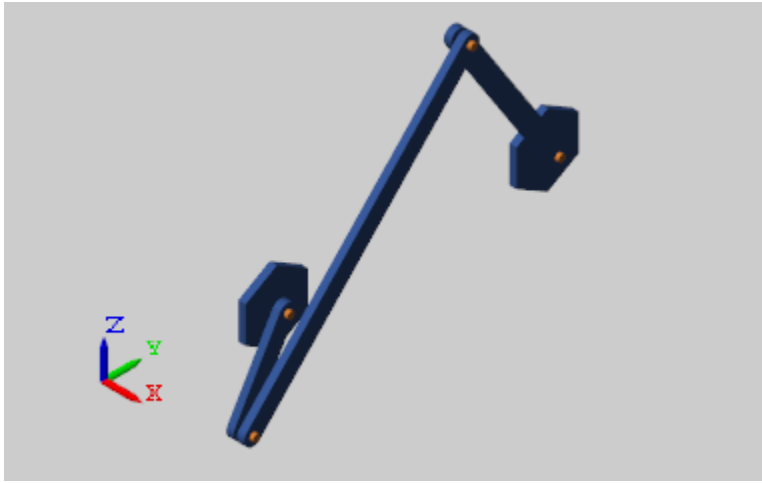
Library	Block	Quantity
Simscape > Multibody > Frames and Transforms	Transform Sensor	2
Simscape > Multibody > Frames and Transforms	World Frame	1
Simscape > Utilities	PS-Simulink Converter	4
Simulink > Sinks	To Workspace	4

2. In the Transform Sensor block dialog boxes, select **Translation > Y** and **Translation > Z**. Resize the block as needed.
3. In the **Output signal unit** parameters of the PS-Simulink Converter block dialog boxes, enter `cm`.
4. In the **Variable Name** parameters of the To Workspace block dialog boxes, enter the variable names:
   - `y_crank`
   - `z_crank`
   - `y_rocker`
   - `z_rocker`
5. Connect and name the blocks as shown in the figure, rotating them as needed. Ensure that the To Workspace blocks with the z_crank and z_rocker variable names connect to the z frame ports of the Transform Sensor blocks. The new blocks are shaded gray.



Simulate Model

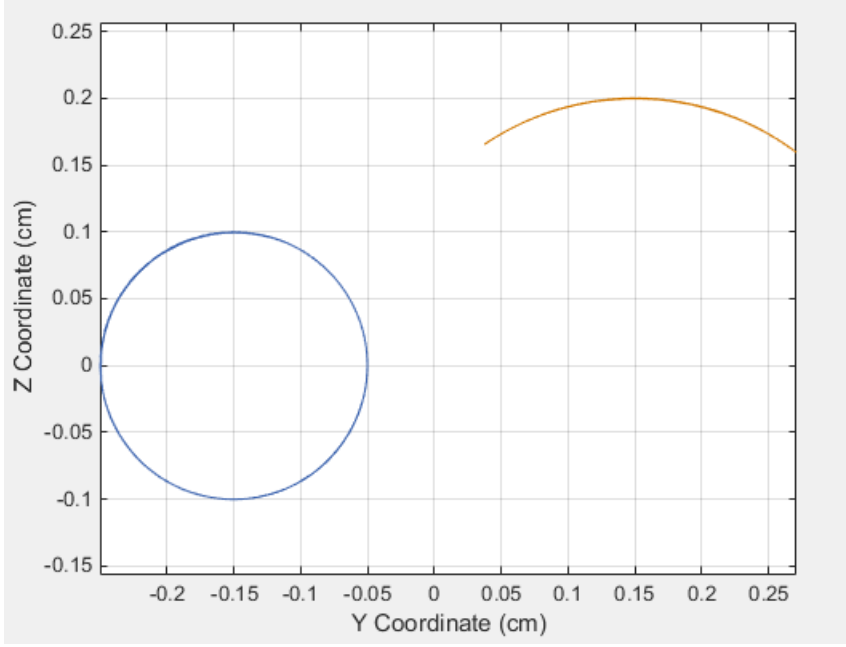
Run the simulation. You can do this in the Simulink tool bar by clicking the run button. Mechanics Explorer plays a physics-based animation of the four bar assembly.



Once the simulation ends, you can plot the position coordinates of the coupler link end frames, e.g., by entering the following code at the MATLAB command line:

```
figure;
plot(y_crank.data, z_crank.data, 'color', [60 100 175]/255);
hold;
plot(y_rocker.data, z_rocker.data, 'color', [210 120 0]/255);
xlabel('Y Coordinate (cm)');
ylabel('Z Coordinate (cm)');
axis equal; grid on;
```

The figure shows the plot that opens. This plot shows that the crank completes a full revolution, while the rocker completes a partial revolution, e.g., it oscillates. This behavior is characteristic of crank-rocker systems.

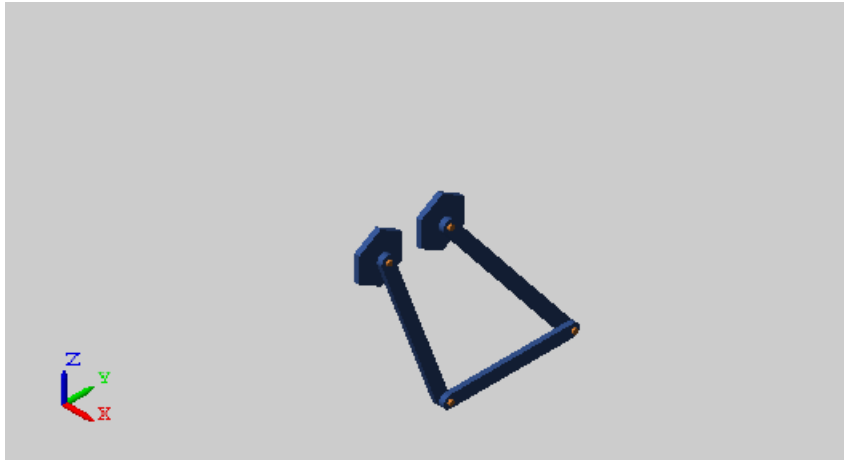


Simulate Model in Double-Crank Mode

Try simulating the model in double-crank mode. You can change the four-bar linkage into a double-crank linkage by changing the binary link lengths according to the table.

Block	Parameter	Value
Binary Link A	**Length**	25
Binary Link B	**Length**	20
Binary Link A1	**Length**	30
Crank-Base Transform	**Translation > Offset**	5
Rocker-Base Transform	**Translation > Offset**	5

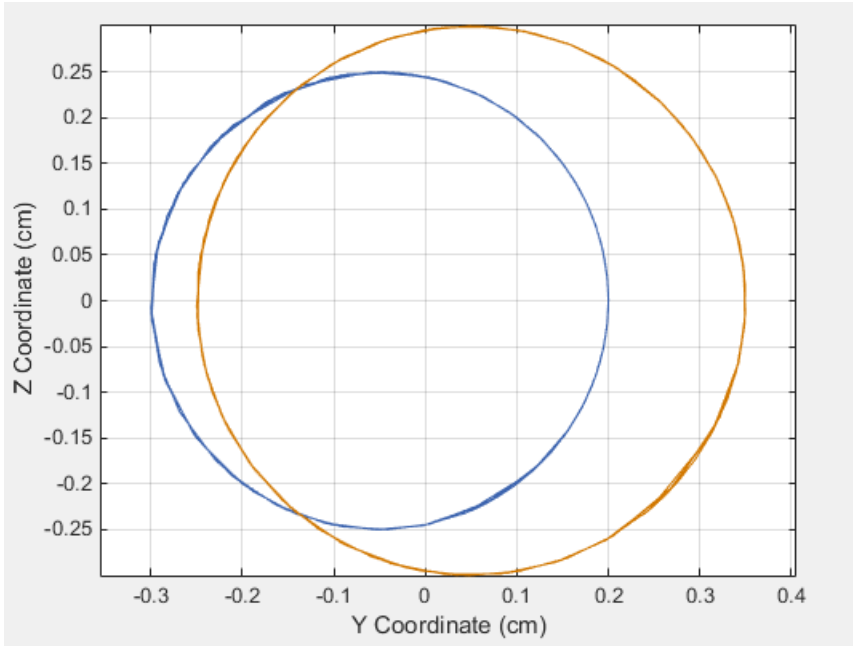
Update and simulate the model. The figure shows the updated visualization display in Mechanics Explorer.



Plot the position coordinates of the coupler link end frames. At the MATLAB command line, enter:

```
figure;
plot(y_crank.data, z_crank.data, 'color', [60 100 175]/255);
hold;
plot(y_rocker.data, z_rocker.data, 'color', [210 120 0]/255);
xlabel('Y Coordinate (cm)');
ylabel('Z Coordinate (cm)');
axis equal; grid on;
```

The figure shows the plot that opens. This plot shows that both links complete a full revolution. This behavior is characteristic of double-crank linkages.

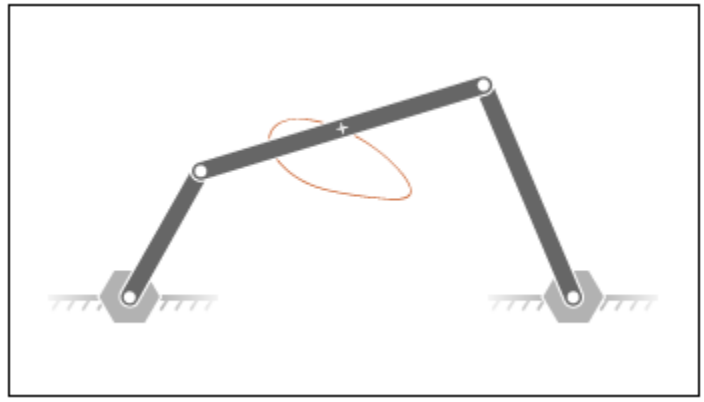

0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
Z Coordinate (cm)
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
Y Coordinate (cm)

Analyze Motion at Various Parameter Values

In this section...
“Model Overview” on page 3-85
“Build Model” on page 3-85
“Specify Block Parameters” on page 3-87
“Create Simulation Script” on page 3-88
“Run Simulation Script” on page 3-89

Model Overview

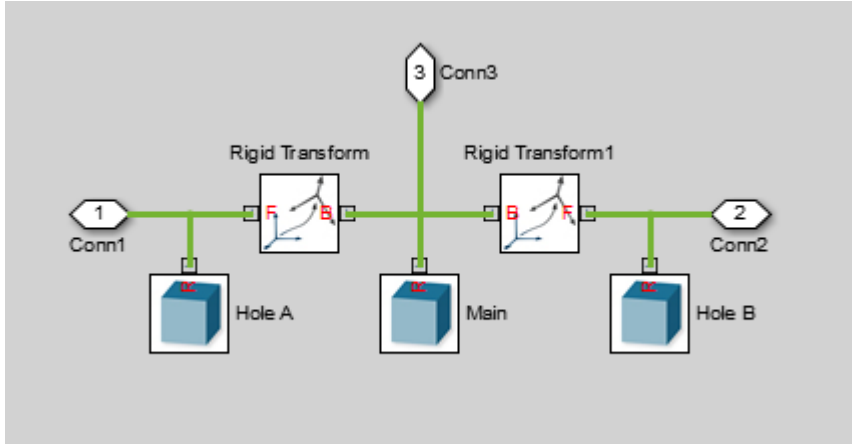
In this tutorial, you create a simple MATLAB script to simulate a four-bar model at various coupler lengths. The script uses the coupler motion coordinates, obtained using a Transform Sensor block, to plot the resulting coupler curve at each value of the coupler length. For information on how to create the four-bar model used in this tutorial, see “Model a Closed-Loop Kinematic Chain” on page 2-21.



Build Model

1 At the MATLAB command prompt, enter `smdoc_four_bar`. A four-bar model opens up. For instructions on how to create this model, see “Model a Closed-Loop Kinematic Chain” on page 2-21.

2 Under the mask of the Binary Link B block, connect a third Outport block as shown in the figure. You can add an Outport block by copying and pasting Conn1 or Conn2. The new block identifies the frame whose trajectory you plot in this tutorial.



3 Add the following blocks to the model. During simulation, the Transform Sensor block computes and outputs the coupler trajectory with respect to the world frame.

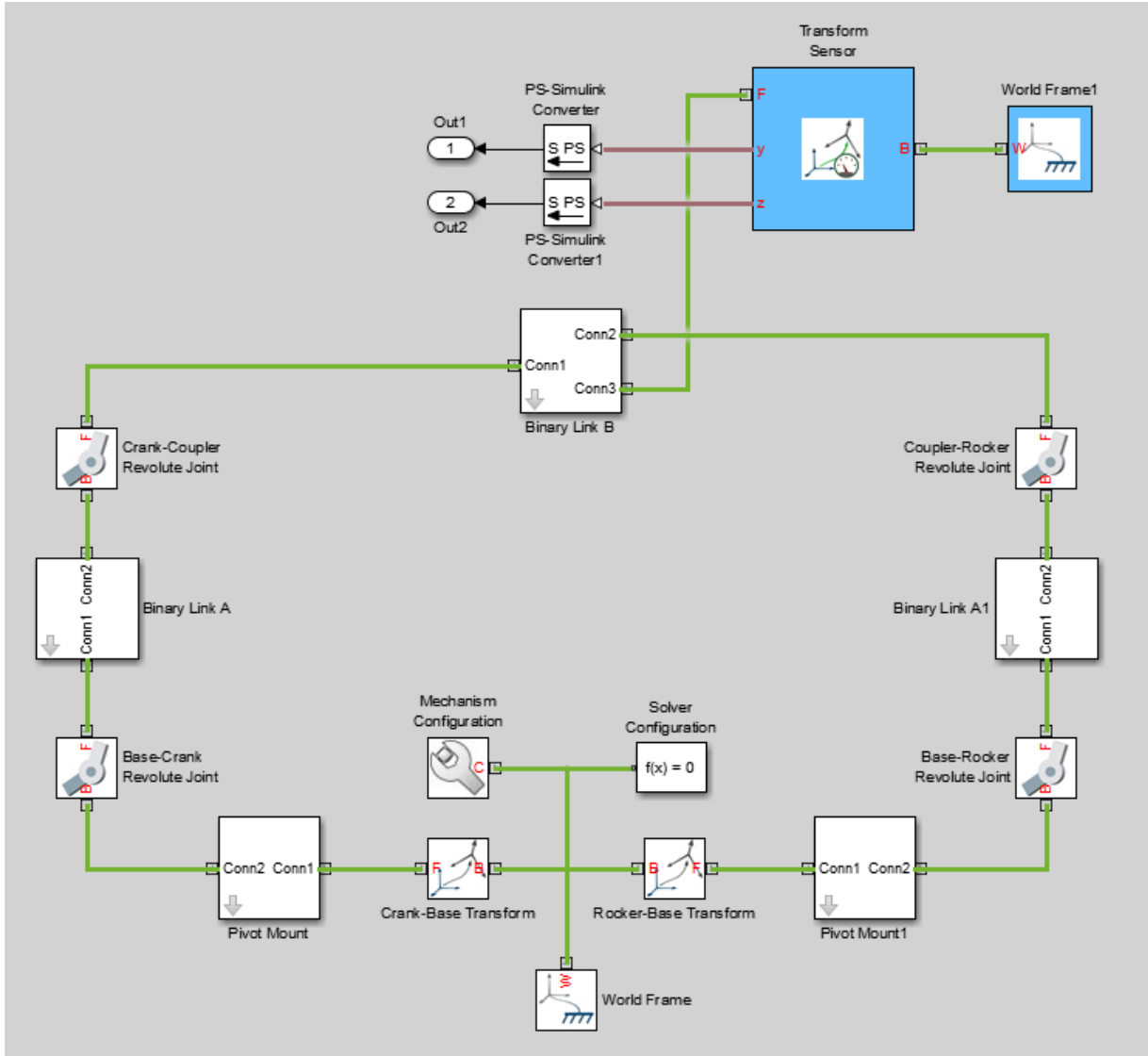
Library	Block	Quantity
Frames and Transforms	World Frame	1
Frames and Transforms	Transform Sensor	1
Simscape Utilities	PS-Simulink Converter	2
Simulink Sinks	Outport	2

4 In the Transform Sensor block dialog box, select these variables:

- **Translation** > **Y**
- **Translation** > **Z**

The block exposes frame ports y and z, through which it outputs the coupler trajectory coordinates.

5 Connect the blocks as shown in the figure. Be sure to flip the Transform Sensor block so that its base frame port, labeled B, connects to the World Frame block.



Specify Block Parameters

1 In the Mechanism Configuration block, change **Uniform Gravity** to `None`.

2 In the Base-Crank Revolute Joint block, specify the following velocity state targets. The targets provide an adequate source of motion for the purposes of this tutorial.

- Select **State Targets** > **Specify Velocity**.
- In **State Targets** > **Specify Velocity** > **Value**, enter `2` rev/s.
- Deselect **State Target** > **Specify Position**.

3 Specify the following link lengths. The coupler link length is parameterized in terms of a MATLAB variable, `LCoupler`, enabling you change its value iteratively using a simple MATLAB script.

Block	Parameter	Value
Binary Link B	Length	LCoupler
Binary Link A1	Length	25

4 Save the model in a convenient folder, naming it smdoc_four_bar_msensing.

Create Simulation Script

Create a MATLAB script to iteratively run simulation at various coupler link lengths:

1 On the MATLAB toolstrip, click **New Script**.

2 In the script, enter the following code:

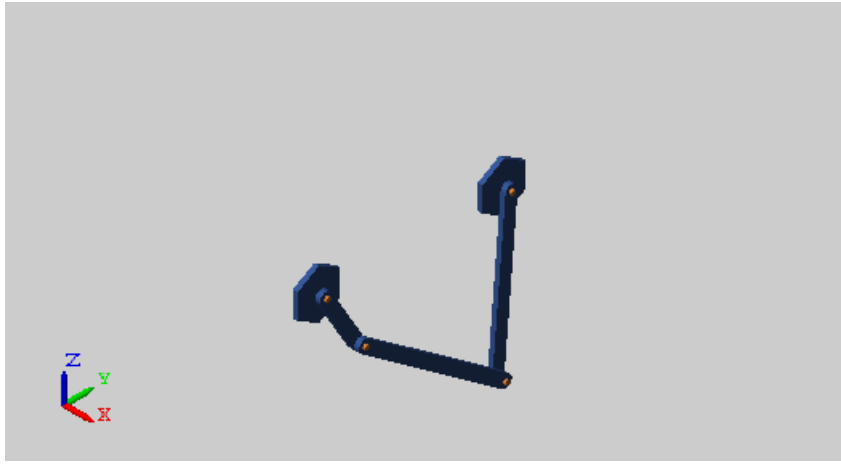
```
% Run simulation nine times, each time
% increasing coupler length by 1 cm.
% The original coupler length is 20 cm.
for i = (0:8);
    LCoupler = 20+i;

    % Simulate model at the current coupler link length (LCoupler),
    % saving the Outport block data into variables y and z.
    [~, ~, y, z] = sim('smdoc_four_bar_msensing');

    % Plot the [y, z] coordinates of each coupler curve
    % on the x = i plane. i corresponds to the simulation run number.
    x = zeros(size(y)) + i;
    plot3(x, y, z, 'Color', [1 0.8-0.1*i 0.8-0.1*i]);
    view(30, 60); hold on;
 end
```

The code runs simulation at nine different coupler link lengths. It then plots the trajectory coordinates of the coupler link center frame with respect to the world frame. Coupler link lengths range from 20 cm to 28 cm.

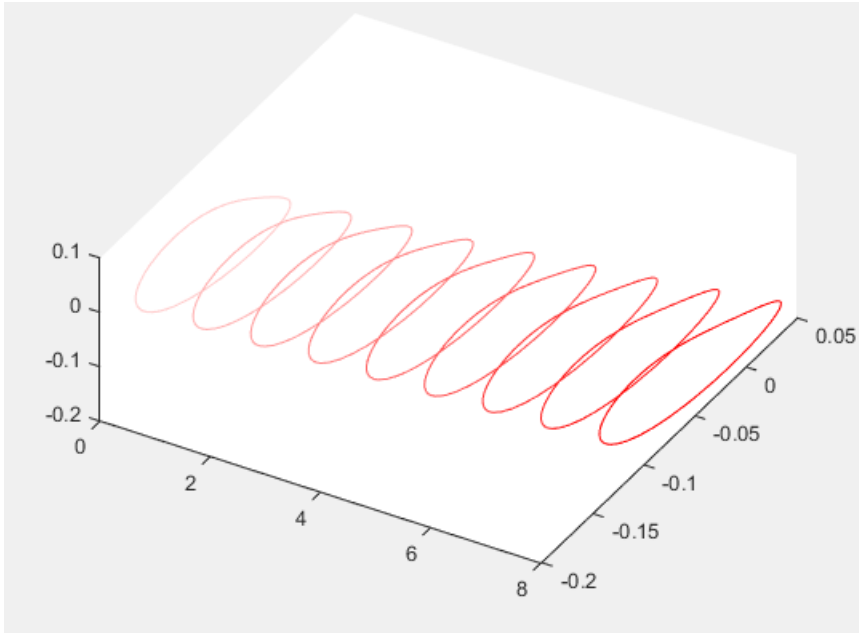
3 Save the script as sim_four_bar in the folder containing the four-bar model.

Run Simulation Script

Run the sim_four_bar script. In the MATLAB Editor toolstrip, click the **Run** button or, with the editor active, press **F5**. Mechanics Explorer opens with a dynamic 3-D view of the four-bar model.



Simscape Multibody iteratively runs each simulation, adding the resulting coupler link curve to the active plot. The figure shows the final plot.



You can use the simple approach shown in this tutorial to analyze model dynamics at various parameter values. For example, you can create a MATLAB script to simulate a crank-slider model at different coupler link lengths, plotting for each simulation run the constraint force acting on the piston.

Sense Forces and Torques Acting at Joints

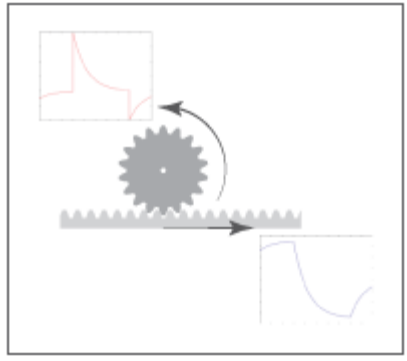
In this section...
“Overview” on page 3-91
“Open Model” on page 3-92
“Sense Actuation Torque” on page 3-92
“Sense Constraint Forces” on page 3-95
“Sense Total Forces” on page 3-97

Overview

Simscape Multibody provides force and torque sensing in joint blocks. You can use this sensing capability to compute and output various types of forces and torques acting directly at joints. Force and torque types that you can sense include those due to:

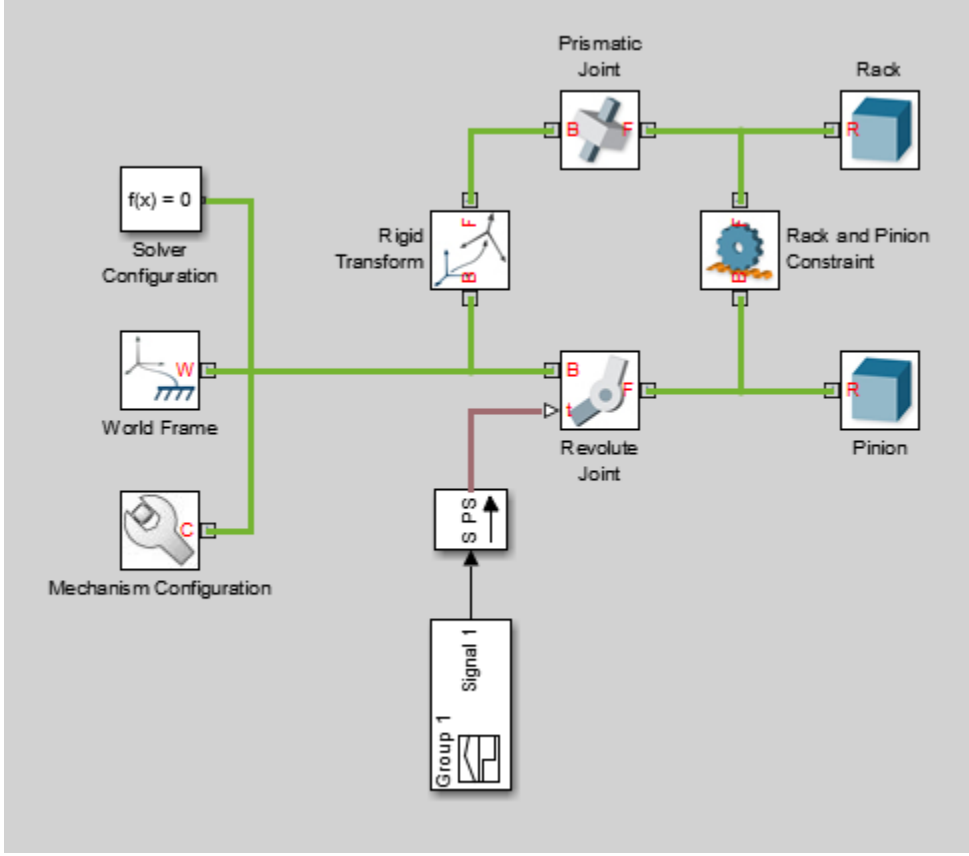
- Joint actuation inputs
- Joint constraints
- Joint actuation inputs, constraints, and internal mechanics combined

In this tutorial, you explore the different types of force and torque sensing that Simscape Multibody joint blocks provide.



Open Model

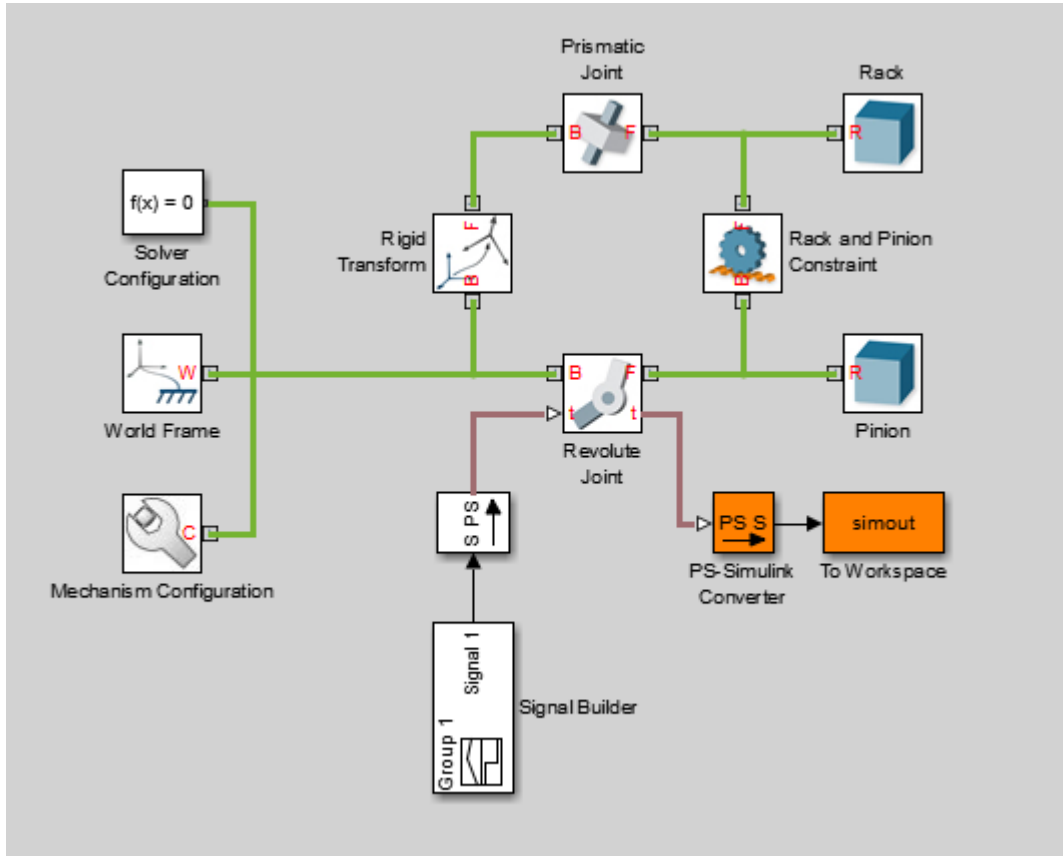
At the MATLAB command prompt, enter `smdoc_rack_pinion_c`. Simscape Multibody opens a rack and pinion model that you can use to explore the force and torque sensing capability of joint blocks.



Sense Actuation Torque

The rack and pinion model contains an actuation torque input that drives the pinion revolute joint. A Simulink-PS Converter block processes the input signal using a second-order filter, smoothing any abrupt changes or discontinuities the signal may have. To sense the actuation torque as observed at the Revolute Joint block:

1. In the Revolute Joint block dialog box, select **Z Revolute Primitive (Rz) > Sensing > Actuator Torque**. The block exposes a physical signal port, labeled t. This port outputs the 3-D vector components of the joint actuator torque in a Simscape physical signal.
2. Drag the following blocks into the model:
   - PS-Simulink Converter from the **Simscape > Utilities** library
   - To Workspace from the **Simulink > Sinks** library
3. Connect the blocks as shown in the figure.

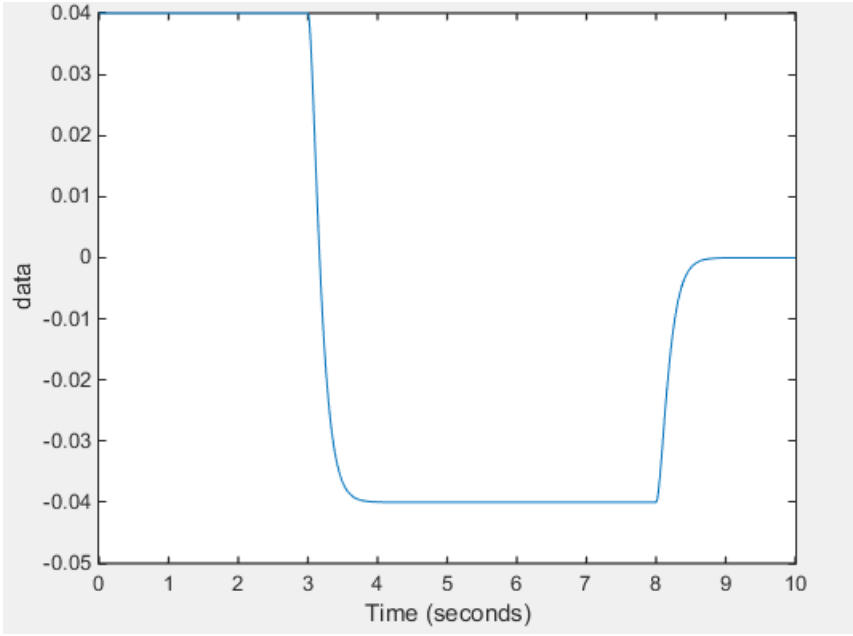


4. Simulate the model. The To Workspace block outputs the actuator torque signal into a time-series variable, simout, available in the MATLAB base workspace.

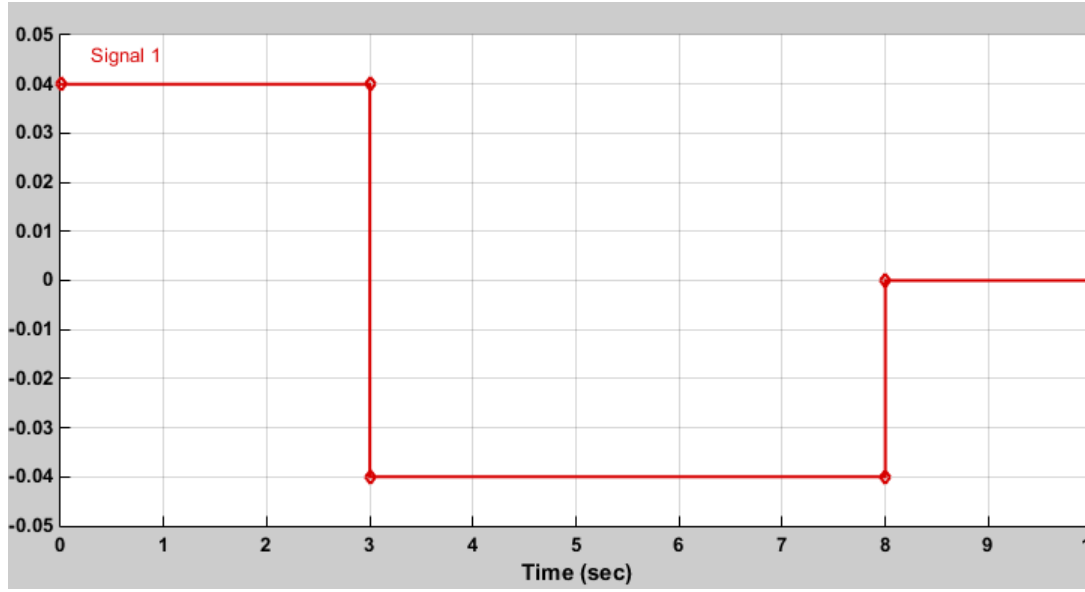
5 At the MATLAB command prompt, enter:

```
figure;
plot(simout);
```

MATLAB plots the vector components of the joint actuator torque. All but the Z component are zero throughout the simulation.



Compare the actuator torque plot to the original input signal in the Signal Builder block. Neglecting any signal smoothing due to the second-order filtering, the two signals are identical. The following figure shows the original input signal.



Actuator force and torque sensing enables you to analyze the required forces and torques to yield a prescribed joint trajectory. Use this feature in your model to perform inverse dynamic and other types of analysis.

Sense Constraint Forces

Joint constraint forces, which act normal to the joint primitive axes, restrict motion to the allotted joint degrees of freedom. In the Revolute Joint block, the constraint forces resist the pull of gravity, keeping the pinion fixed with respect to the world frame. To sense the constraint forces:

1. In the Mechanism Configuration block, set **Uniform Gravity** to `Constant`. This setting ensures that gravity acts on the rack and pinion system. Check that the gravity vector is `[0 0 -9.80665]`.
2. In the Revolute Joint block dialog box, select **Composite Force/Torque Sensing** > **Constraint Force**. The block exposes the physical signal port fc. This port provides the vector components of the joint-wide constraint force in a Simscape physical signal. By default, this is the constraint force that the follower port frame exerts on the base port frame, resolved in the base port frame.

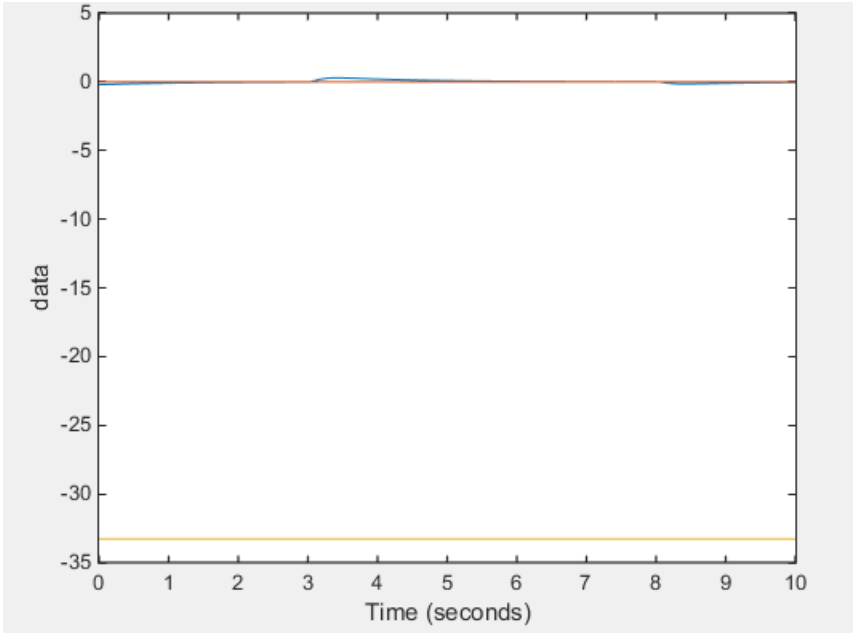
3 Deselect **Z Revolute Primitive (Rz) > Sensing > Actuator Torque**.

4 Check that the PS-Simulink Converter block now connects to the physical signal port fc.

5 Simulate the model. At the MATLAB command prompt, enter:

```
figure;
plot(simout);
```

MATLAB plots the constraint force components with respect to time. All but one component are zero throughout simulation. The Z component, which opposes the gravity vector, is the only component needed to hold the joint frames in place.



Constraint forces ensure that weld joint frames remain fixed with respect to each other. You can place a Weld Joint block inside a body subsystem to sense the internal forces and torques acting within that body during simulation. For an example of how you can do this in a double pendulum model, see “Sense Constraint Forces” on page 3-99.

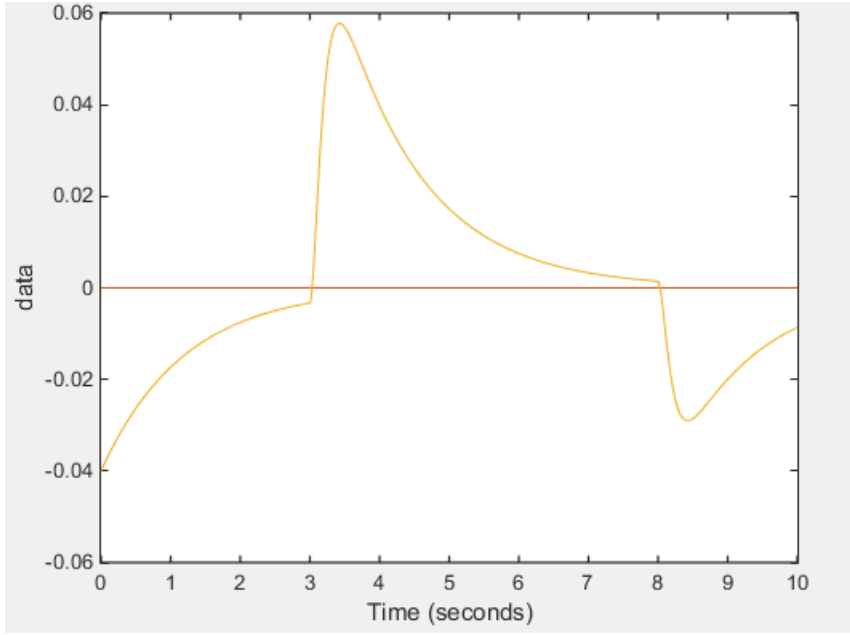
Sense Total Forces

In addition to actuation and constraint forces and torques, joint frames can also interact by exchanging internal forces and torques. These forces and torques, which are due to spring and damper elements internal to the joint itself, enable you to account for mechanical energy dissipation and storage between the joint frames. You can sense the total composite force and torque acting at a joint, which includes contributions from actuation, constraint, and internal forces and torques. To sense the total torque acting between the port frames of the Revolute Joint block:

1. In the Revolute Joint block dialog box, select **Composite Force/Torque Sensing** > **Total Torque**. The block exposes the physical signal port tt. This port outputs the total torque acting between the joint frames as a Simscape physical signal.
2. Deselect **Composite Force/Torque Sensing** > **Constraint Force**.
3. Simulate the model.
4. At the MATLAB command prompt, enter:

   ```
   figure;
   plot(simout);
   ```

   MATLAB plots the vector components of the total torque vector as a function of time. All but one component are zero throughout simulation. The nonzero component, a torque directed about the Z axis, contains torque contributions from actuation and internal torques, but none from constraint torques.



The torque peaks correspond to the actuation torque values specified in the input signal. These peaks decay with time due to the internal damping torques specified in the Revolute Joint block dialog box. The damping torques cause the energy dissipation evident in the transient portions of the total torque plot.

To verify that the total torque excludes any contribution from constraint torques, try sensing the constraint torques directly. A plot of the constraint torques will show that they are in fact negligible.

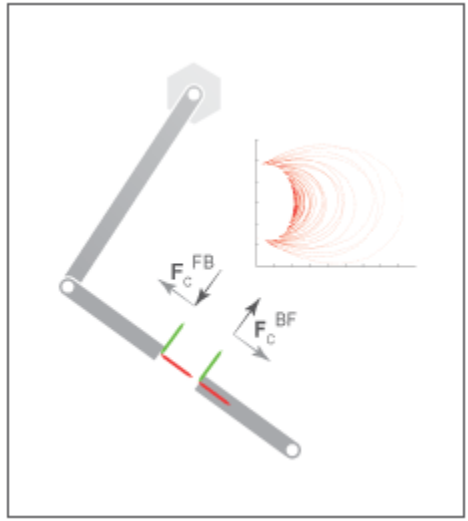
Sense Constraint Forces

In this section...
“Model Overview” on page 3-99
“Add Weld Joint Block to Model” on page 3-100
“Add Constraint Force Sensing” on page 3-101
“Add Damping to Joints” on page 3-102
“Simulate Model” on page 3-102
“Plot Constraint Forces” on page 3-103

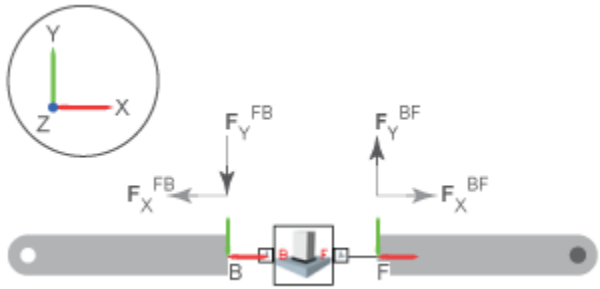
Model Overview

Simscape Multibody provides various types of force and torque sensing. Using joint blocks, you can sense the actuation forces and torques driving individual joint primitives. You can also sense the total and constraint forces acting on an entire joint.

In this tutorial, you use a Weld Joint block to sense the time-varying internal forces that hold a body together. A double-pendulum model, `smdoc_double_pendulum`, provides the starting point for the tutorial. For information on how to create this model, see “Model an Open-Loop Kinematic Chain” on page 2-16.



By connecting the Weld Joint block between solid elements in a binary link subsystem, you can sense the constraint forces acting between them. The following figure shows the constraint forces that you sense in this tutorial. The longitudinal constraint force aligns with the X axis of the weld joint frames. The transverse constraint force aligns with the Y axis. The constraint force along the Z axis is negligible and therefore ignored.

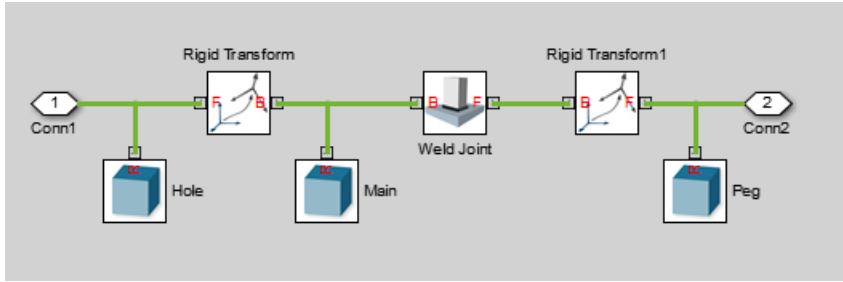


The Weld Joint block enables you to sense the constraint force that the follower frame exerts on the base frame or, alternatively, the constraint force that the base frame exerts on the follower frame. The two forces have the same magnitude but, as shown in the binary link schematic, opposite directions. In this tutorial, you sense the constraint force that the follower frame exerts on the base frame.

You can also select the frame to resolve the constraint force measurement in. The resolution frame can be either the base frame or the follower frame. Certain joint blocks allow their port frames to have different orientations, causing the same measurement to differ depending on your choice of resolution frame. However, because the Weld Joint block provides zero degrees of freedom, both resolution frames yield the same constraint force vector components.

Add Weld Joint Block to Model

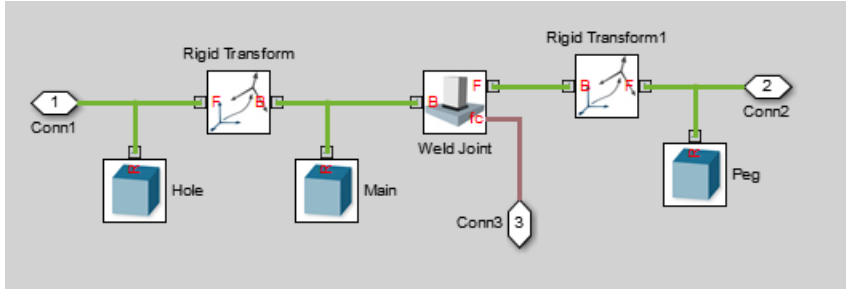
1. At the MATLAB command prompt, enter `smdoc_double_pendulum`. A double-pendulum model opens up.
2. Click the Look Inside Mask arrow in the Binary Link A1 subsystem block.
3. From the **Simscape** > **Multibody** > **Joints** library, drag a Weld Joint block.
4. Connect the Weld Joint block as shown in the figure. This block enables you to sense the constraint forces that hold the body together during motion. Because it provides zero degrees of freedom between its port frames, it has no effect on model dynamics.



Add Constraint Force Sensing

1 In the Weld Joint block dialog box, select **Constraint Force**. The block exposes a physical signal output port labeled fc.

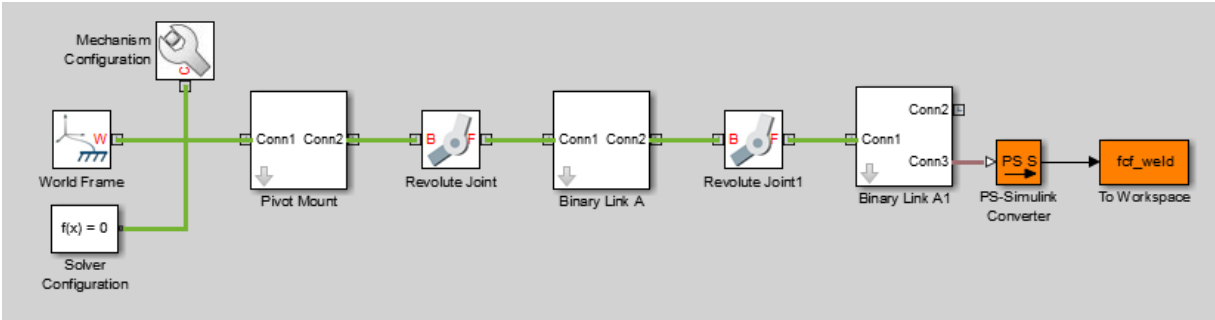
2 Add a Simscape Output port to the subsystem block diagram. Connect the block as shown in the figure and exit the subsystem view.



3 Drag the following blocks into the main window of the model. These blocks enable you to output the constraint force signal into the MATLAB workspace.

Library	Block
Simscape > Utilities	PS-Simulink Converter
Simulink > Sinks	To Workspace

4 Connect the blocks as shown in the figure. Check that the PS-Simulink Converter block connects to the newly added Simscape port.



5 Specify these block parameters.

Block	Dialog Box Parameter	Value
PS-Simulink Converter	**Output signal unit**	`mN`
To Workspace	**Variable name**	`fcf_weld`

Units of `mN` are appropriate for this model, which contains Aluminum links roughly 30 cm × 2 cm × 0.8 cm.

Add Damping to Joints

In each Revolute Joint block dialog box, select **Internal Mechanics** > **Damping Coefficient** and enter `1e-5`. The damping coefficient enables you to model energy dissipation during motion, so that the double-pendulum model eventually comes to rest.

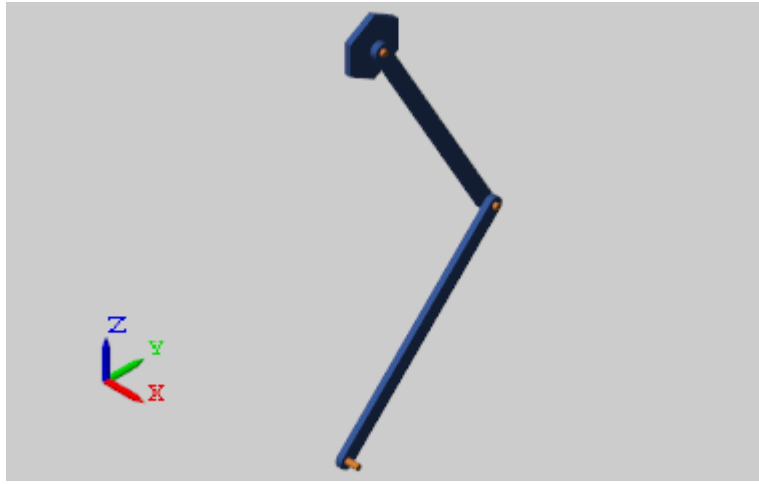
Simulate Model

1 In the Simulink Editor menu bar, select **Simulation** > **Model Configuration Parameters**.

2 In the Solver tab of the Configuration Parameters window, set the **Solver** parameter to `ode15s`. This is the recommended solver for physical models.

3 In the same tab, set the **Max step size** parameter to `0.001` s.

4 Run the simulation. You can do this from the Simulink Editor menu bar, by selecting **Simulation** > **Run**. Mechanics Explorer opens with a dynamic view of the model. In the Mechanics Explorer menu bar, select the Isometric View button to view the double pendulum from an isometric perspective.

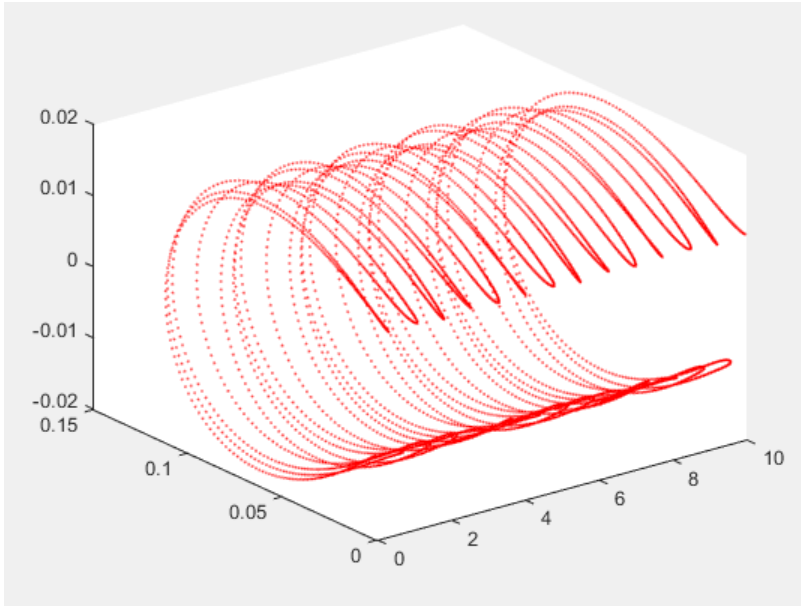


Plot Constraint Forces

At the MATLAB command prompt, enter the following plot commands:

```
figure;
grid on;
xlabel('T, s');
ylabel('F_{C,X}, mN');
zlabel('F_{C,Y}, mN');
plot3(fcf_weld.time, fcf_weld.data(:,1), fcf_weld.data(:,2),...
'.', 'MarkerSize', 1, 'Color', 'r');
```

MATLAB plots the axial and transverse constraint forces with respect to time in 3-D. The figure shows the resulting plot.

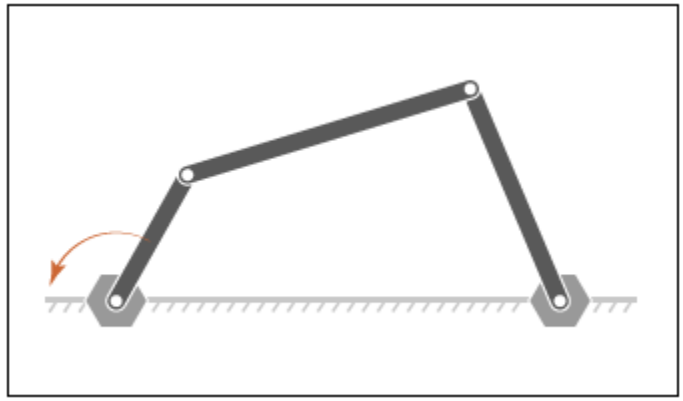

0.02
0.01
0
-0.01
-0.02
0.15
0.1
0.05
0
0
2
4
6
8
10

Specify Joint Motion Profile

In this section...
“Model Overview” on page 3-105 “Build Model” on page 3-105 “Simulate Model” on page 3-108

Model Overview

In this tutorial, you prescribe the time-varying crank angle of a four-bar linkage using a Revolute Joint block. Then, during simulation, you sense the actuation torque at the same joint corresponding to the prescribed motion.



Build Model

1 At the MATLAB command prompt, enter `smdoc_four_bar`. A four-bar model opens. This is the model you create in tutorial “Model a Closed-Loop Kinematic Chain” on page 2-21.

2 In the dialog box of the Base-Crank Revolute Joint block, specify the following parameters settings.

Parameter	Setting
Actuation > Torque	`Automatically Computed`

Parameter	Setting
Actuation > Motion	`Provided by Input`
Sensing > Actuator Torque	Selected

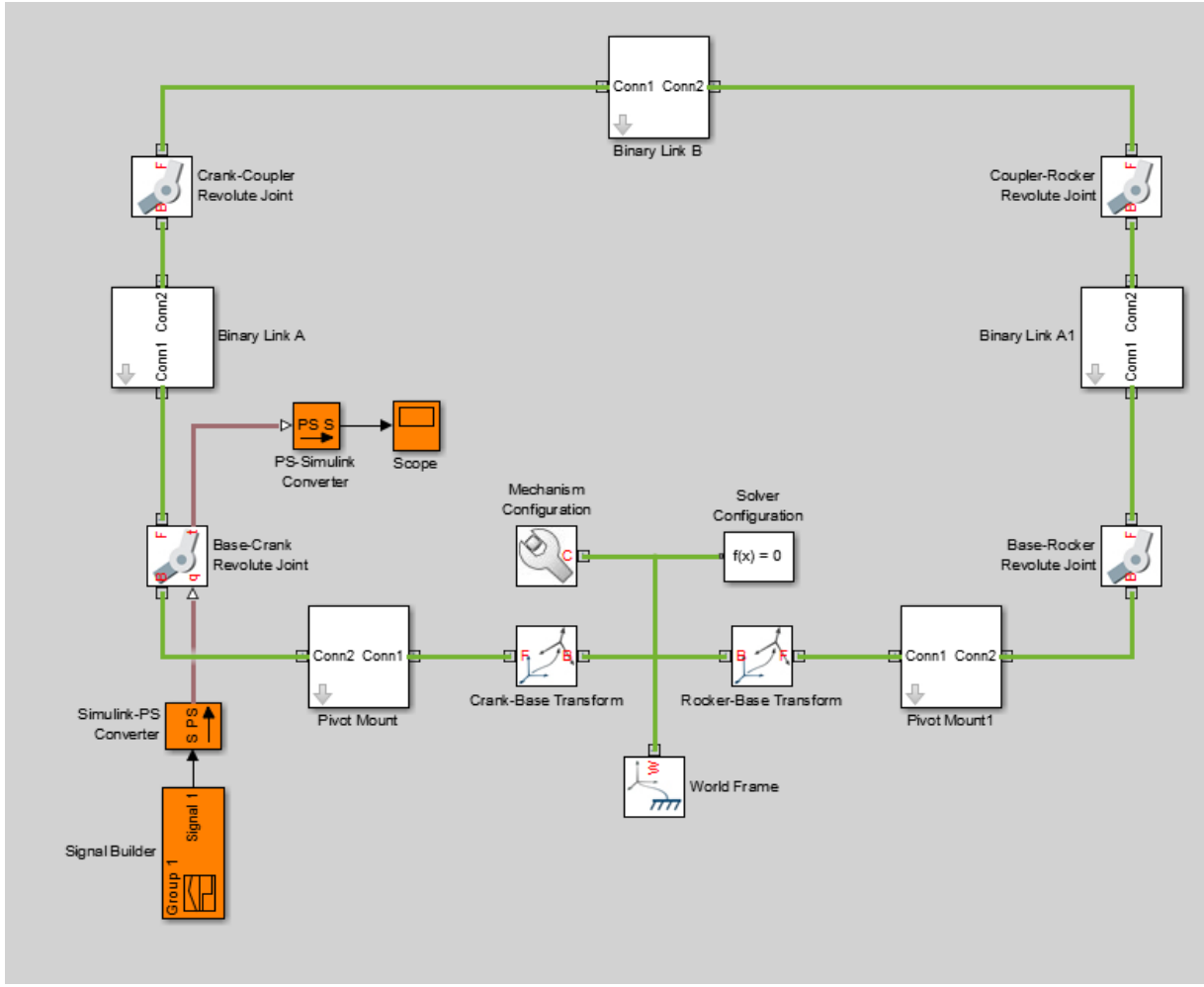
The joint block displays two physical signal ports. Input port q accepts the joint angular position. Output port t provides the joint actuation torque required to achieve that angular position.

3 In each of the four Revolute Joint block dialog boxes, set **Internal Mechanics > Damping Coefficient** to `5e-4 N*m/(deg/s)`. During simulation, damping forces between the joint frames account for dissipative losses at the joints.

4 Drag the following blocks into the model. These blocks enable you to specify an actuation torque signal and plot the joint position.

Block	Library
Simulink-PS Converter	**Simscape > Utilities**
PS-Simulink Converter	**Simscape > Utilities**
Scope	**Simulink > Sinks**
Signal Builder	**Simulink > Sources**

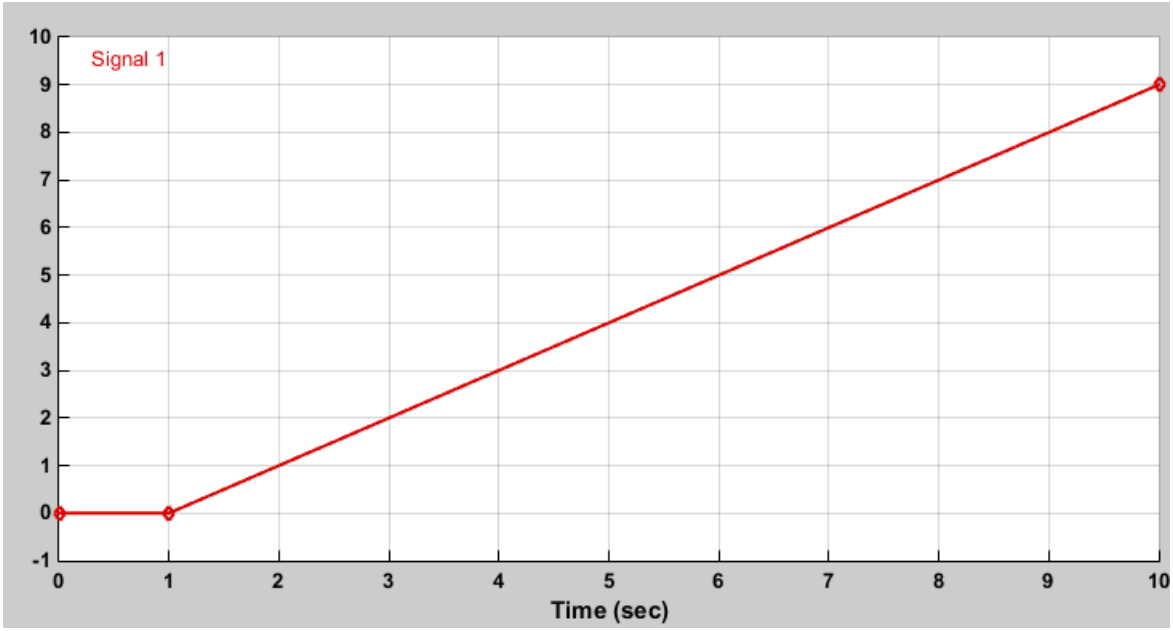
5 Connect the blocks as shown in the figure.



6 In the Input Handling tab of the Simulink-PS Converter block dialog box, specify the following block parameters.

Parameter	Value
Filtering and derivatives	`Filter input`
Input filtering order	`Second-order filtering`

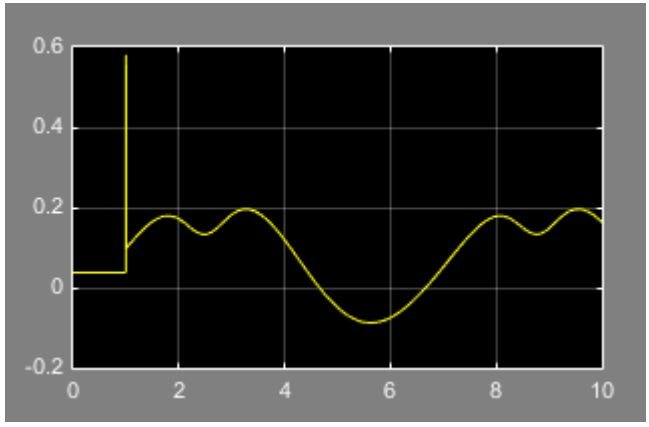
7 In the Signal Builder window, specify the joint angular trajectory as shown in the figure.



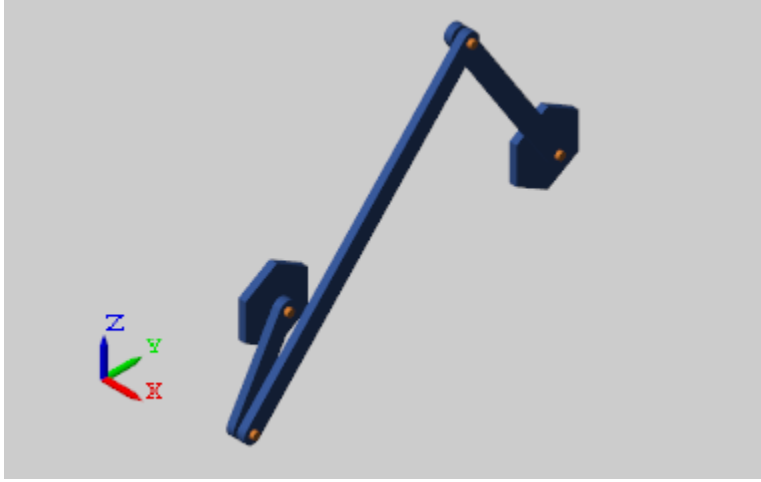
This signal corresponds to a constant angular speed of 1 rad/s from t = 1s onwards.

Simulate Model

Run the simulation, e.g., by selecting **Simulation** > **Run** from the Simulink menu bar. Mechanics Explorer opens with a dynamic display of the four-bar model.



Open the Scope window. The scope plot shows the joint actuation torque with which you can achieve the motion you prescribed.



See Also

Related Examples

- “Sense Motion Using a Transform Sensor Block” on page 3-67
- “Prescribe Joint Motion in Planar Manipulator Model” on page 3-110
- “Specifying Motion Input Derivatives” on page 3-32

Prescribe Joint Motion in Planar Manipulator Model

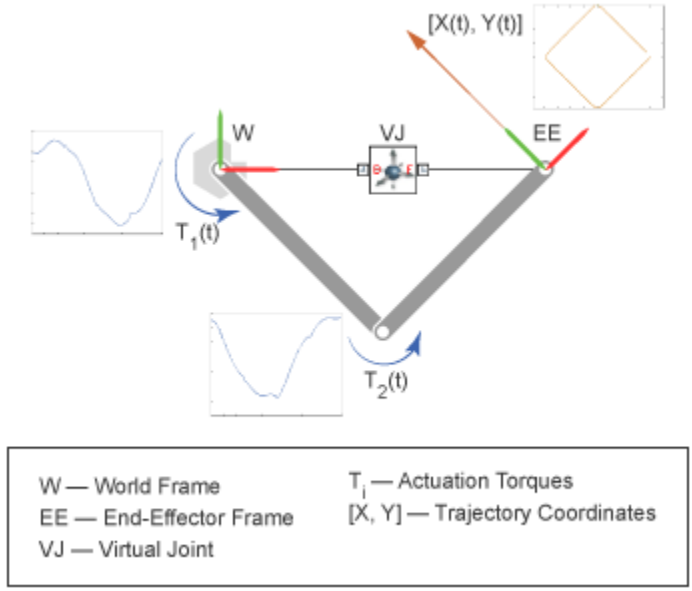
In this section...

"Model Overview" on page 3-110
"Add Virtual Joint" on page 3-111
"Prescribe Motion Inputs" on page 3-112
"Sense Joint Actuation Torques" on page 3-116
"Simulate Model" on page 3-117

Model Overview

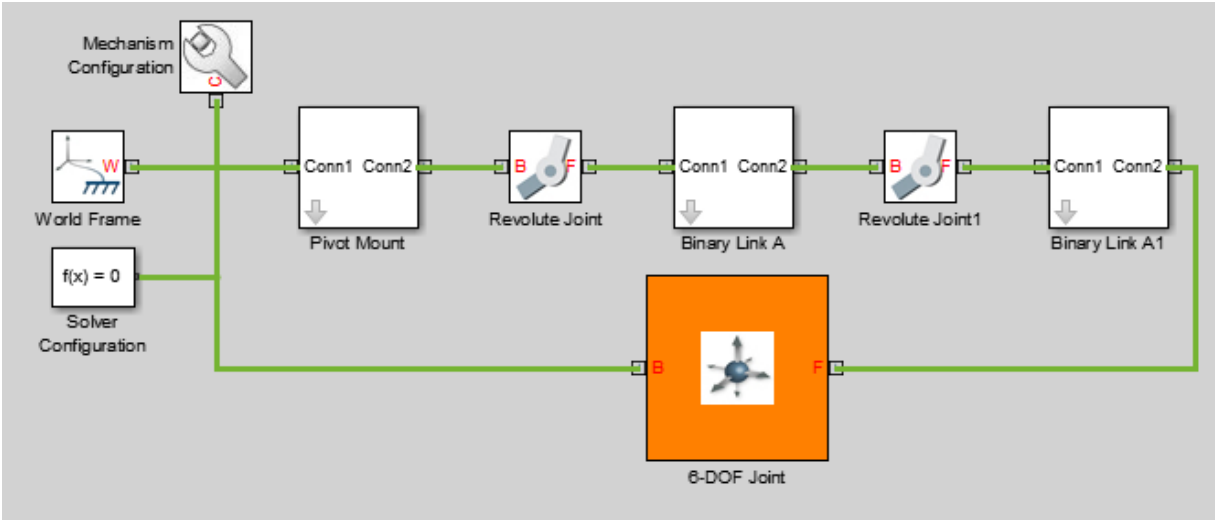
In this tutorial, you prescribe the time-varying trajectory coordinates of a planar manipulator end frame with respect to the world frame using a 6-DOF Joint block. This block provides the requisite degrees of freedom between the two frames, but it does not represent a real physical connection between them. The joint it represents is said to be virtual.

The time-varying coordinates trace a square pattern, achieved by automatically computing and applying actuation torques at the various manipulator joints. During simulation, you can output the automatically computed torques and plot them using Simulink blocks or MATLAB commands, e.g. for analysis purposes.



Add Virtual Joint

1. At the MATLAB command prompt, enter `smdoc_double_pendulum`. A double pendulum model, which in this tutorial you adapt as a simple planar manipulator model, opens. For instructions on how to create this model, see “Model an Open-Loop Kinematic Chain” on page 2-16
2. From the **Simscape** > **Multibody** > **Joints** library, drag a 6-DOF Joint block and connect it as shown in the figure. This block represents a virtual joint, which you use to specify the manipulator end frame with respect to the world frame.



Note Check that the base port frame (B) connects to the world frame. The base port frame functions as the reference frame for any joint motion input that you provide. Switching the base and follower port frames causes the block to interpret any motion input with respect to a different frame, possibly altering the manipulator end frame trajectory.

Prescribe Motion Inputs

1 In the 6-DOF Joint block dialog box, specify these parameters settings.

Parameter	Select
Y Prismatic Primitive (Py) > Actuation > Motion	`Provided by Input`
Z Prismatic Primitive (Pz) > Actuation > Motion	`Provided by Input`

The block exposes two physical signal ports through which you can provide the joint motion inputs.

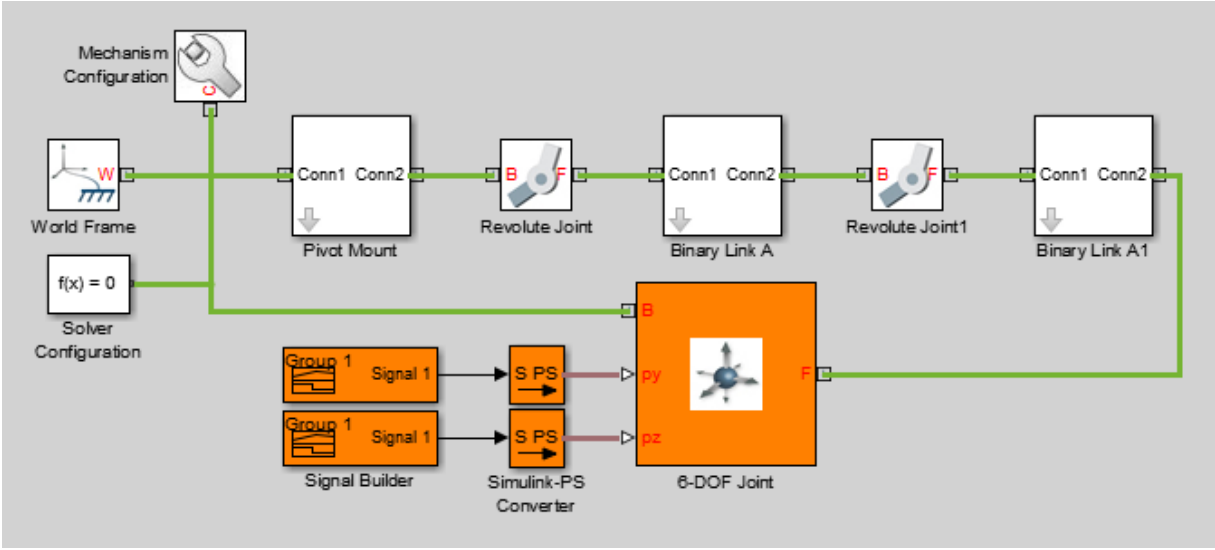
2 Drag these blocks into the model.

Library	Block	Quantity
Simscape > Utilities	Simulink-PS Converter	2

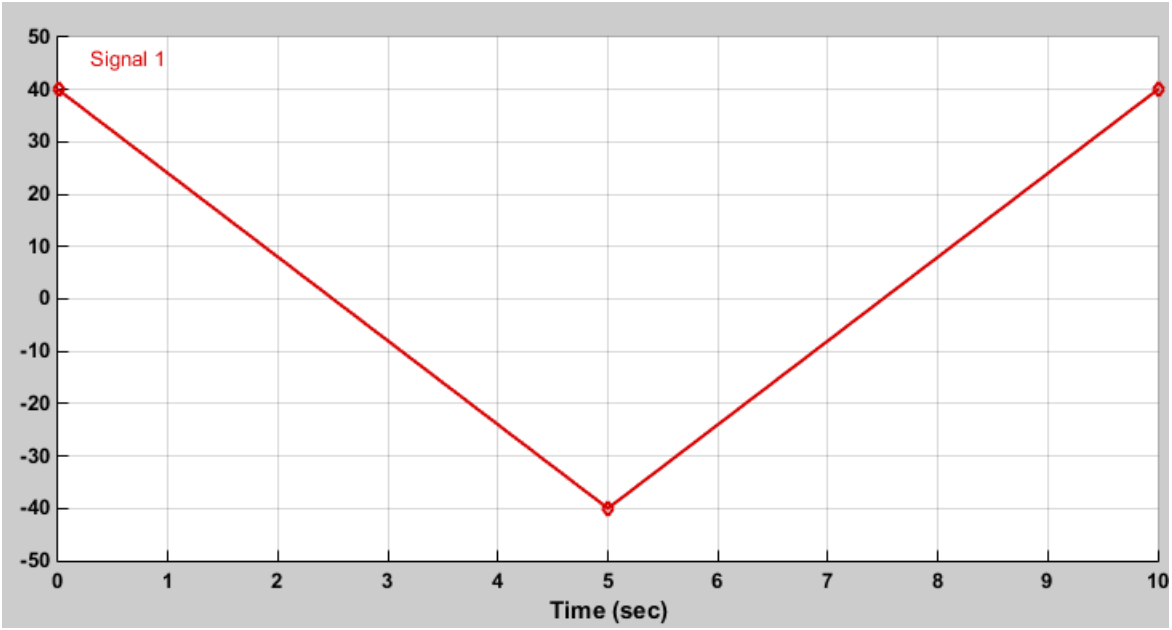
Library	Block	Quantity
Simulink > Sources	Signal Builder	2

The Signal Builder blocks provide the motion inputs as Simulink signals. The Simulink-PS Converter blocks convert the Simulink signals into Simscape physical signals compatible with Simscape Multibody blocks.

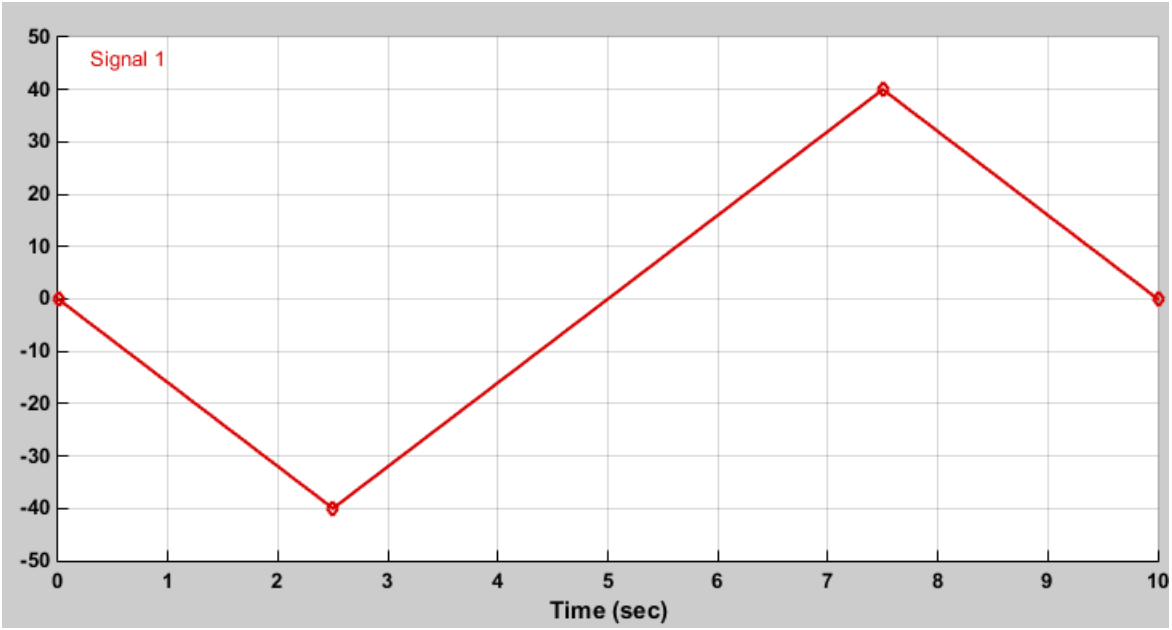
3 Connect the blocks as shown in the figure.



4 Open the dialog box of the Signal Builder block connected to port py of the 6-DOF Joint block. Specify this signal, the time-varying Y coordinate of the square trajectory the manipulator end frame is to follow.



5 Open the dialog box of the Signal Builder block connected to port pz of the 6–DOF Joint block. Specify this signal, the time-varying Z coordinate of the square trajectory the manipulator end frame is to follow.



6 In the dialog boxes of the Simulink-PS Converter blocks, specify the input signal units and filtering settings. Simscape Multibody requires that you either specify second-order filtering or provide the first two time derivatives of the trajectory coordinates.

Parameter	Value
Units > Input signal unit	`cm`
Input Handling > Filtering and derivatives	`Filter input`
Input Handling > Input filtering order	`Second-order filtering`
Input Handling > Input filtering time constant (in seconds)	`0.1`

Small filtering constants can slow simulation significantly. For most Simscape Multibody models, a value of 0.1 seconds is a good choice. In this tutorial, this value suffices.

Sense Joint Actuation Torques

1 In the dialog boxes of the two Revolute Joint blocks, set the following actuation and sensing parameters.

Parameter	Setting
Actuation > Torque	`Automatically Computed`
Sensing > Actuation Torque	Selected

Simscape Multibody requires the number of joint primitive degrees of freedom with motion inputs to equal the number with automatically computed joint actuation forces and torques. If the model does not meet this condition, simulation fails with an error.

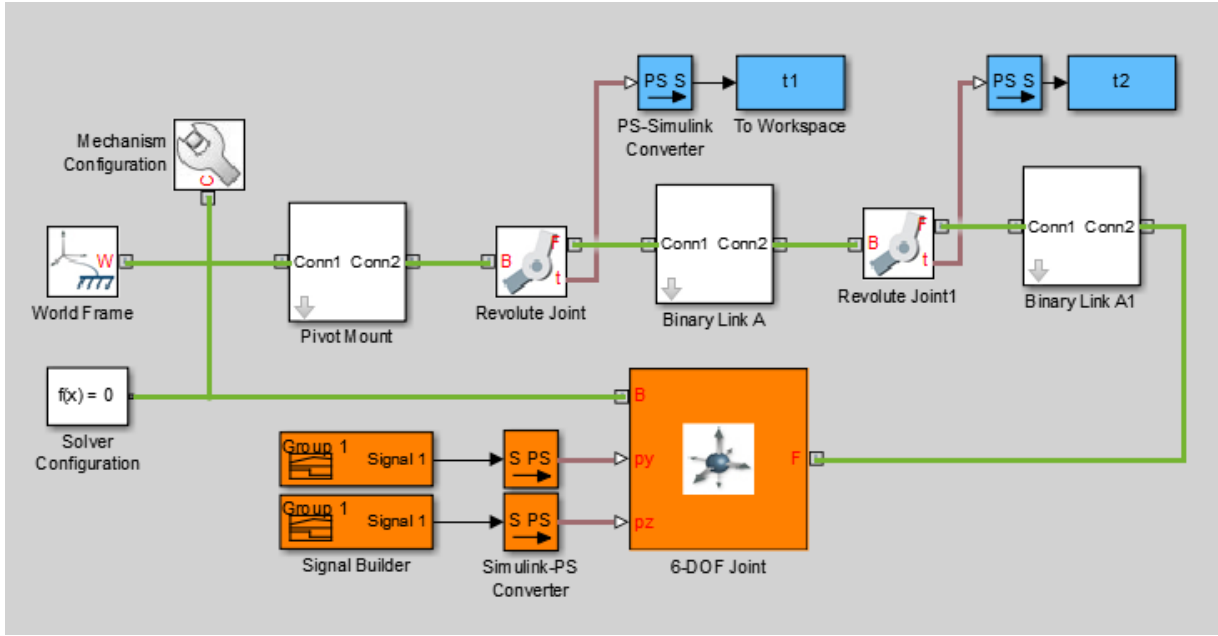
2 Drag these blocks into the model.

Library	Block	Quantity
Simscape > Utilities	PS-Simulink Converter	2
Simulink > Sinks	To Workspace	2

The PS-Simulink Converter blocks convert the physical signal outputs into Simulink signals compatible with other Simulink blocks.

3 In the two To Workspace block dialog boxes, enter the variable names `t1` and `t2`.

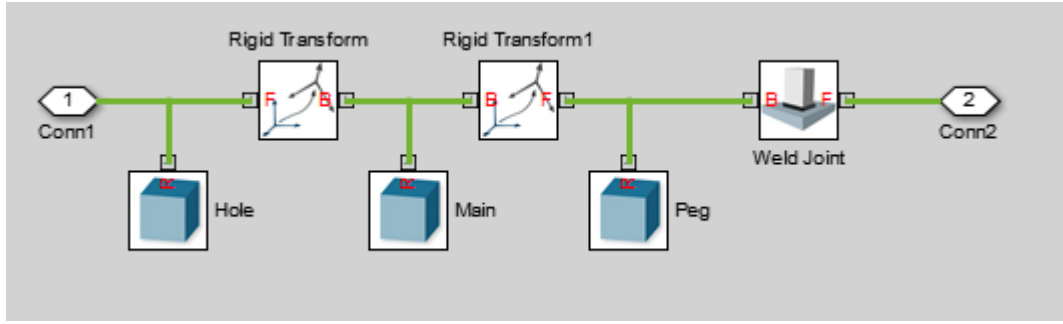
4 Connect the blocks as shown in the figure.



Simulate Model

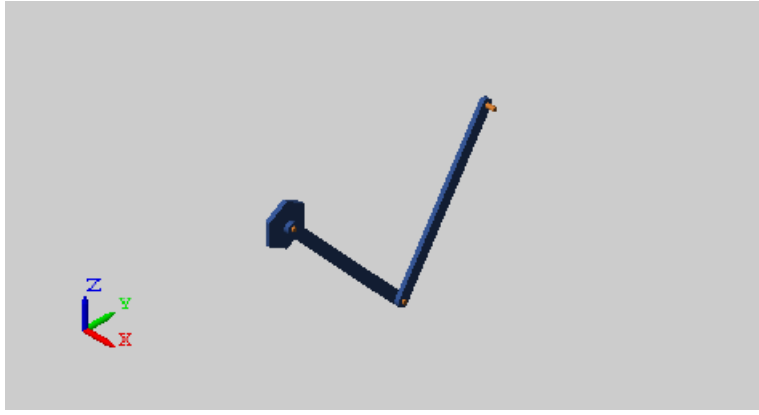
Attempt to run the simulation. You can do this in the Simulink Editor menu bar, by selecting **Simulation** > **Run**. Simulation fails with an error arising from the closed kinematic loop present in the model. Simscape Multibody requires this loop to contain at least one joint block without motion inputs or automatically computed actuation forces or torques.

1 From the **Simscape** > **Multibody** > **Joints** library, drag a Weld Joint block and connect it inside one of the Binary Link A subsystems.



Adding the Weld Joint block ensures that the now-closed-loop system contains at least one joint block without motion inputs or computed actuation torques.

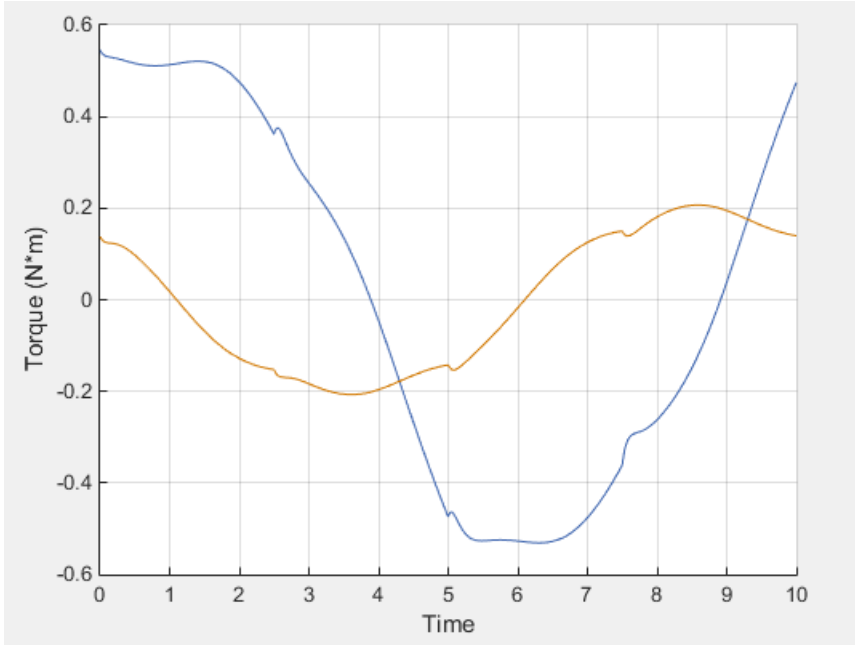
Run the simulation once again. Mechanics Explorer opens with a dynamic 3-D display of the two-bar linkage.



Plot the computed actuation torques acting at the two revolute joints in the linkage. At the MATLAB command line, enter this code:

```
figure;
hold on;
plot(t1.time, t1.data, 'color', [60 100 175]/255);
plot(t2.time, t2.data, 'color', [210 120 0]/255);
xlabel('Time');
ylabel('Torque (N*m)');
grid on;
```

The plot shows the time-varying actuation torques acting at the two revolute joints. These torques enable the manipulator end frame to trace the prescribed square trajectory.



See Also

Related Examples

- “Sense Motion Using a Transform Sensor Block” on page 3-67
- “Specify Joint Motion Profile” on page 3-105
- “Specifying Motion Input Derivatives” on page 3-32

More About

- “Specifying Joint Actuation Inputs” on page 3-25
- “Actuating and Sensing with Physical Signals” on page 3-36

Simulation and Visualization

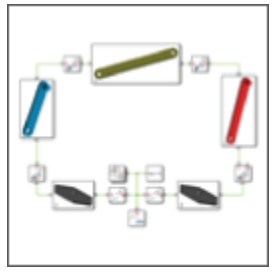
Simulation

- "Update and Simulate a Model" on page 4-2
- "Multibody Simulation Issues" on page 4-5

Update and Simulate a Model

In this section...
“Create or Open a Model” on page 4-2
“Update the Block Diagram” on page 4-2
“Examine the Model Assembly” on page 4-3
“Configure the Solver Settings” on page 4-3
“Run Simulation and Analyze Results” on page 4-4

Create or Open a Model

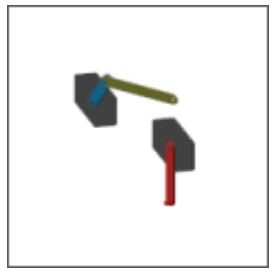


You can create a model manually or import it from a supported CAD application. For an example showing how to create a model manually, see “Model an Open-Loop Kinematic Chain” on page 2-16. For an example showing how to import a model, see “Import a Robotic Arm CAD Model” on page 6-17.

Update the Block Diagram

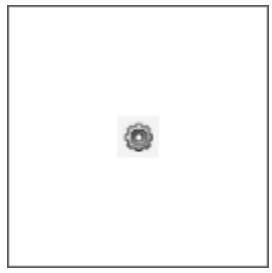
In the Simulink menu bar, select **Simulation** > **Update Diagram**. Mechanics Explorer opens with a static visualization of the model in its initial state.

Examine the Model Assembly



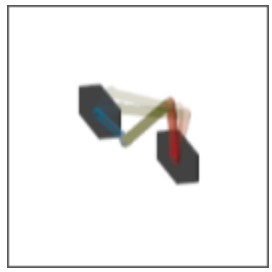
Check the model assembly in the visualization pane of Mechanics Explorer. Look for bodies placed and oriented in unexpected ways. Use the Simscape Variable Viewer or the Simscape Multibody Model Report to identify any assembly issues. For an example showing the use of Model Report, see “Model a Closed-Loop Kinematic Chain” on page 2-21.

Configure the Solver Settings



In the Simulink menu bar, select **Simulation > Model Configuration Parameters**. Pick a solver and specify the desired step sizes. Ensure that the time steps are small enough to accurately capture the fastest meaningful changes in your model. Use care, though, as small time steps slow down simulation.

Run Simulation and Analyze Results



In the Simulink menu bar, select **Simulation** > **Run**. Mechanics Explorer plays a physics-based animation of your model. Examine any data generated during simulation, for example, through Simulink Scope plots. For an example showing how to work with sensing data from a model, see “Sense Motion Using a Transform Sensor Block” on page 3-67.

Multibody Simulation Issues

Under certain conditions, a model that you simulate can behave in unexpected ways. Some issues that you can encounter while simulating a Simscape Multibody model include:

Limited visualization in models with For Each Subsystem blocks. Models with one or more For Each Subsystem blocks simulate with limited visualization. The Mechanics Explorer visualization utility displays the model in only one of the instances provided by the For Each Subsystem blocks. Model simulation is not affected.

No visualization in models with Model blocks. Models with Model blocks (often referred to as referenced models) simulate without visualization. During simulation, Simscape Multibody software issues a warning at the MATLAB command line. If previously closed, The Mechanics Explorer visualization utility does not open.

No Simscape local solver support. Simscape Multibody software does not support Simscape local solvers. If you select a local solver in the Simscape Solver Configuration block, the solver does not apply to the Simscape Multibody portion of a model—which relies exclusively on the global Simulink solver selected in the **Model Configuration Parameters** window.

See Also

Related Examples

- “Configure Model for Rapid Accelerator Mode” on page 7-9

Visualization and Animation

- "Enable Mechanics Explorer" on page 5-2
- "Working with Animation" on page 5-3
- "Manipulate the Visualization Viewpoint" on page 5-5
- "Visualization Cameras" on page 5-10
- "Create a Dynamic Camera" on page 5-15
- "Selective Model Visualization" on page 5-19
- "Selectively Show and Hide Model Components" on page 5-25
- "Visualize Simscape Multibody Frames" on page 5-32
- "Go to a Block from Mechanics Explorer" on page 5-37
- "Create a Model Animation Video" on page 5-39

Enable Mechanics Explorer

The Mechanics Explorer visualization utility opens by default whenever you update or simulate a Simscape Multibody model. Each model that you update or simulate adds a new tab to Mechanics Explorer. If Mechanics Explorer does not open, ensure that model visualization is turned on:

1. In the Simscape Multibody menu bar, select **Simulation** > **Model Configuration Parameters**.
2. Expand the **Simscape Multibody** node and select **Explorer**.
3. Ensure the **Open Mechanics Explorer on model update or simulation** check box is selected.

Working with Animation

In this section...
“Animation Playback” on page 5-3
“Looping Playback” on page 5-3
“Changing Playback Speed” on page 5-3
“Jumping to Playback Time” on page 5-4

Animation Playback

Animation is cached during model simulation. What you see when you run a simulation is the animation playback, unless the simulation is slower than the animation caching. In that case, the animation goes no faster than the simulation can produce the cache.

Once a partial or complete animation is cached from simulation, starting the animation again plays back the cache, without running the simulation a second time. You can move backward and forward to any time in the cached animation.

The animation cache is stored until you close Mechanics Explorer. When you simulate the model, the cache is updated with new animation data. To create a permanent record of a model animation, you must create an animation video. See “Create a Model Animation Video” on page 5-39

Looping Playback

Use the **Toggle Loop** button in the Mechanics Explorer playback toolstrip to automatically replay an animation from the start once it reaches the end. The cached animation replays indefinitely until you click the **Pause** button. Enable looping by clicking the **Toggle Loop** button. Disable looping by clicking the button again.

Changing Playback Speed

Use the playback speed slider in the Mechanics Explorer toolstrip to set the animation playback speed ratio relative to real time. Set the slider to a number greater than 1 for faster playback. Set it to a number smaller than 1 for slower playback.

You can set the slider to multiples of 2 from 1/256 to 256. For slower or faster animations, adjust the base playback speed for the model. To change this parameter, from the Mechanics Explorer menu bar, select **Tools > Animation Settings**.

Jumping to Playback Time

Use the playback slider in the Mechanics Explorer toolstrip to move the playback time to an arbitrary point in the animation timeline. The playback time counter shows the current playback time. Alternatively, enter the desired playback time directly in the playback time counter.

Manipulate the Visualization Viewpoint

In this section...
“Model Visualization” on page 5-5
“Select a Standard View” on page 5-5
“Set View Convention” on page 5-6
“Rotate, Roll, Pan, and Zoom” on page 5-7
“Split Model View” on page 5-8

Model Visualization

Multibody models lend themselves to 3-D visualization, a qualitative means of analysis that you can use to examine body geometries, mechanical connections, and trajectories in three-dimensional space. In Simscape Multibody, you can visualize a model using Mechanics Explorer, adjusting the view point and detail level as needed. You can modify the model view by:

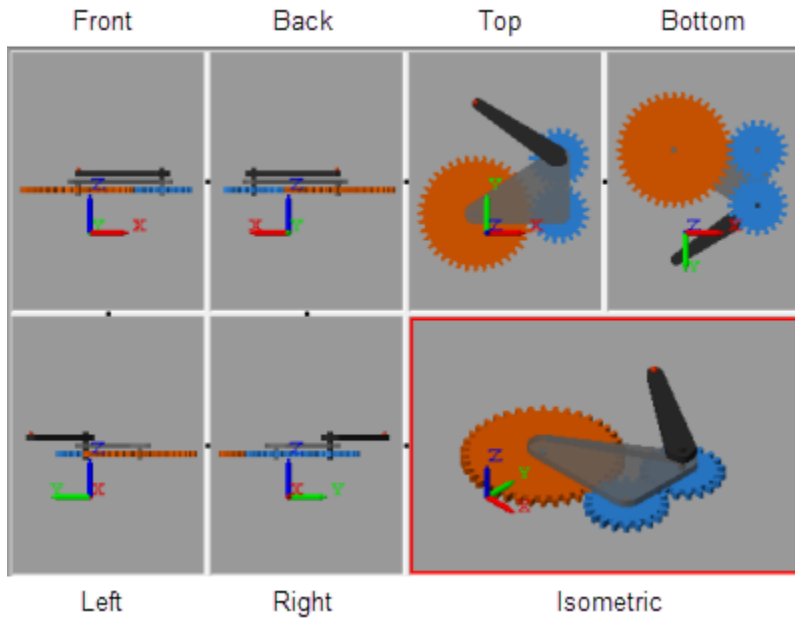
- Selecting a view convention.
- Selecting a standard view.
- Rotating, panning, and zooming.

Select a Standard View

Some view points are so widely used that they are called standard. The isometric view point, corresponding to equal 120° angles between any two world frame axes, is one example. In Mechanics Explorer, you can select such view points by clicking the standard view buttons.

Standard View Buttons

The figure shows a Cardan gear model from the different view points using a `Z up (XY Top)` view convention.

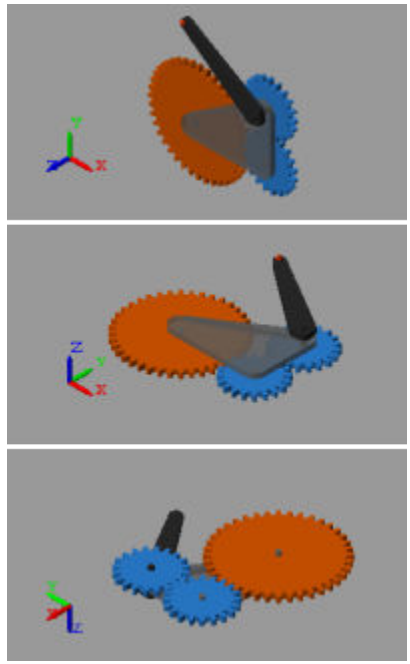


Set View Convention

The view convention helps to determine the perspective from which you view your model. You can align three world frame axes with the vertical direction on your screen, each corresponding to a different view convention:

- `Y up (XY Front)`
- `Z up (XY Top)`
- `Z down (YZ Front)`

The figure shows a Cardan gear model from an isometric perspective using the three view conventions: Y up, Z up, and Z down.



To change the view convention:

1. In the Mechanics Explorer tool strip, set **View convention** to one of the three options.
2. Select a standard view button.

The new view convention takes effect the moment you select a standard view.

Rotate, Roll, Pan, and Zoom

To view your model from an arbitrary point of view or at varying zoom levels, use the Rotate, Roll, Pan, and Zoom buttons. You can find these buttons in the Mechanics Explorer tool strip:

- — Rotate the camera about a general 3-D axis.
- — Roll the camera about its current aim axis.
- — Pan the camera in the current visualization plane.

- — Increase or decrease the camera zoom level.
- — Change the camera zoom to show only the selected region.

You can also use keyboard-and-mouse shortcuts. The table summarizes the available shortcuts.

Button	Shortcut
Rotate	1 Click and hold the mouse scroll wheel. 2 Move the mouse in the direction you want to rotate the model.
Pan	1 Press and hold **Shift**. 2 Click and hold the mouse scroll wheel. 3 Move the mouse in the direction you want to pan the model.
Zoom	1 Press and hold **Ctrl**. 2 Click and hold the mouse scroll wheel. 3 Move the mouse up to zoom in, down to zoom out.

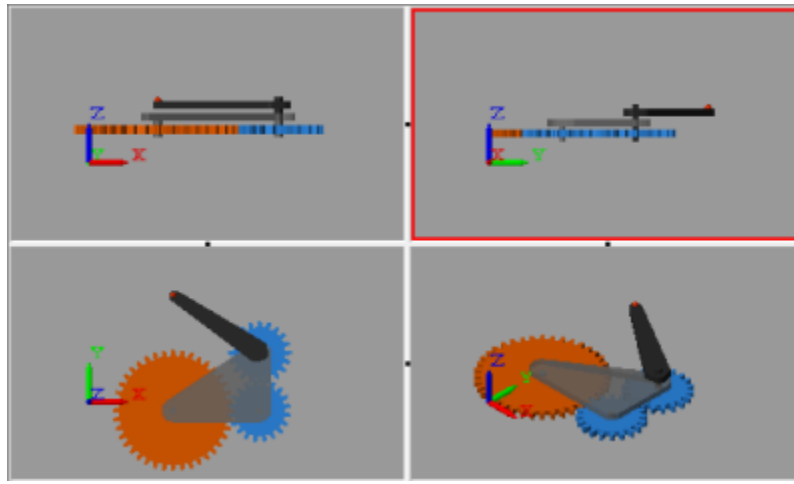
Split Model View

You can view your model from different perspectives, for example, to examine its motion in different planes. So that you can compare different model views, Mechanics Explorer enables you to split the visualization pane into tiles, each with its own view. To split the screen, you use the Mechanics Explorer toolstrip buttons shown in the figure.

Use the buttons to:

- Split the model view into four equally sized tiles, each with a different view point (front, right, top, and isometric views).
- Merge all tiles into a single pane with the view point of the last highlighted tile.
- Split a visualization tile vertically or horizontally into two equally sized tiles.

The figure shows the Cardan gear model with a four-way visualization split.



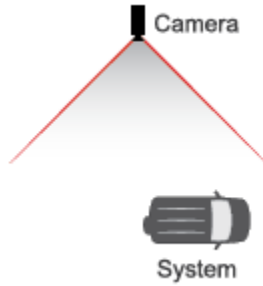
You can merge two tiles by clicking the black dot between the tiles. To ensure that the resulting tile uses the view point of one or the other tile, select that tile first before clicking the black dot between the tiles.

Visualization Cameras

In this section...
“Camera Types” on page 5-10 “Global Camera” on page 5-11 “Dynamic Cameras” on page 5-12 “Camera Trajectory Modes” on page 5-12 “Dynamic Camera Selection” on page 5-13 “Dynamic Camera Reuse” on page 5-14

Camera Types

Cameras define the model viewpoints used during animation playback. Mechanics Explorer supports two camera types—global and dynamic. The global camera provides a static viewpoint that you can manipulate interactively during animation playback. Dynamic cameras provide moving viewpoints that you predefine using **Camera Manager**.

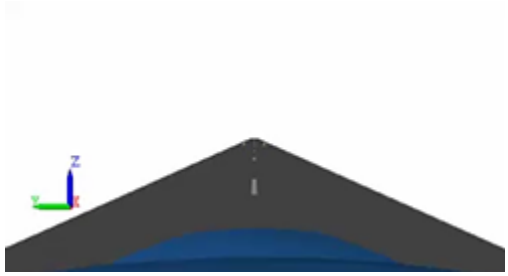


Camera in a Model

The moving viewpoint of a dynamic camera enables you to more easily track the motion of a system. You can use a dynamic camera to keep a moving vehicle such as an automobile or aircraft in view during animation playback. You must define and select a dynamic camera in order to use it in a model. See “Create a Dynamic Camera” on page 5-15.

The figure shows a model visualization captured from the view point of a dynamic camera. This model is part of the Configuring Dynamic Cameras - Vehicle Slalom

featured example. You can open the model from the MATLAB command prompt by entering `sm_vehicle_slalom`.



Example of a Visualization Captured with a Dynamic Camera

Global Camera

The global camera is a static camera that:

- Has no planned trajectory.

  You must manipulate the camera manually to change the camera viewpoint, for example, by using the Pan, Rotate, Roll, and Zoom buttons.
- Is external to the model.

  You cannot position the global camera between bodies, for example, to prevent one body from obstructing another during animation playback.
- Uses an orthographic projection.

  Apparent body sizes remain constant regardless of object distance to the camera. This effect, shown in the figure, is consistent with a camera located relatively far from the model.

The global camera is the default camera for all model visualization tiles—each a subdivision of the model visualization pane, when split. In the absence of custom dynamic cameras, the global camera is the only camera available in a model.

Dynamic Cameras

Dynamic cameras are custom cameras that:

- Have planned trajectories.

  Every dynamic camera follows a trajectory that you prespecify through Camera Manager. You cannot use the Pan, Rotate, Roll, or Zoom buttons during animation playback.
- Can be internal to a model.

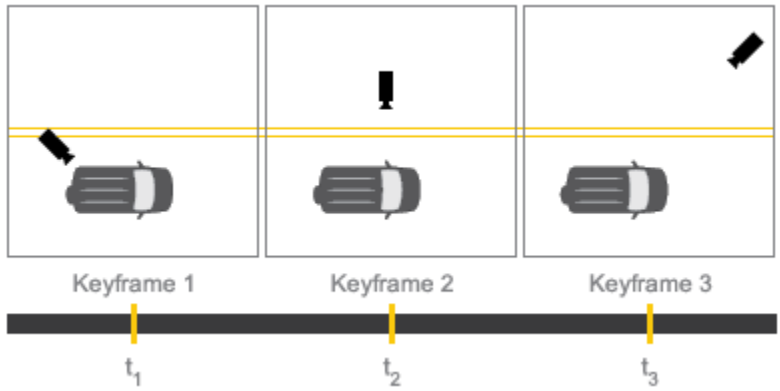
  Dynamic cameras can be inside or outside the perimeter of a model. Position a camera between bodies for a viewpoint internal to the model.
- Use a perspective projection.

  Apparent body sizes vary noticeably with object distance to the camera, creating a more realistic 3-D effect. This effect, shown in the figure, is consistent with a camera located relatively close to the model.

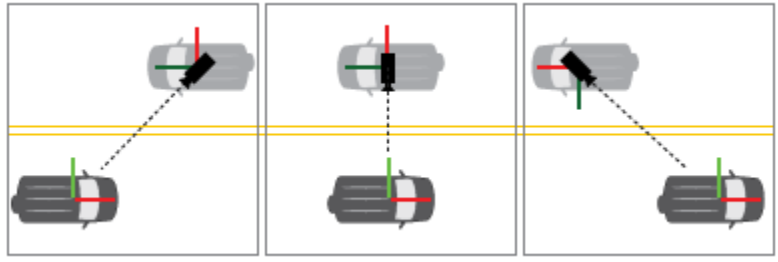
Camera Trajectory Modes

Camera Manager provides two dynamic camera modes that you use to define the camera trajectories:

- `Keyframes` — Set the camera viewpoints at various playback times. Each viewpoint constitutes a keyframe. During playback, the camera transitions between the keyframes using the smooth interpolation method of the `pchip` MATLAB function. Use this camera mode to obtain camera trajectories independent of any components in your model.

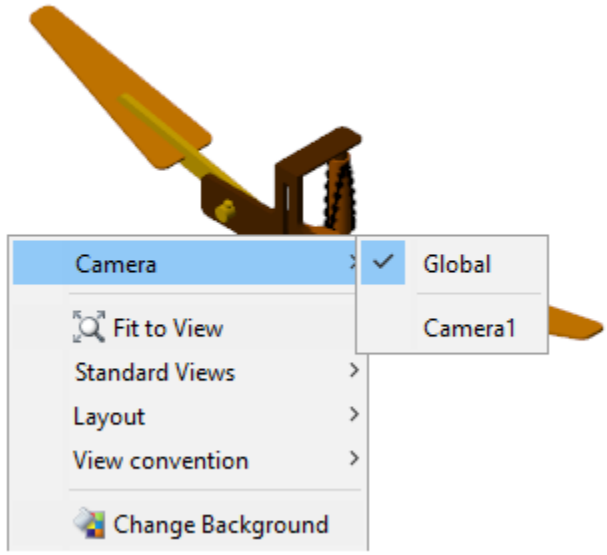


- `Tracking` — Constrain the camera position, aim, and up vector to coordinate frames in your model. During playback, the camera moves with the frames it is constrained to, translating and rotating as needed to satisfy the specified constraints. Use this camera mode to track frames and bodies during playback.



Dynamic Camera Selection

To visualize a model using a dynamic camera, you must first select that camera. To do this, Mechanics Explorer provides the list of available cameras in the visualization pane context-sensitive menu. Right-click the visualization pane to open the menu and select **Camera** to select from the list.



Dynamic Camera Reuse

Dynamic cameras exist only in the models they are defined in. The camera trajectories are based on model-specific frames or viewpoints and are not transferable to other models. You cannot move, copy, or reference a dynamic camera outside of its model. To use a camera in a different model, recreate the camera in that model.

See Also

Related Examples

- “Create a Dynamic Camera” on page 5-15

Create a Dynamic Camera

In this section...
“Start a New Camera Definition” on page 5-15
“Define a Keyframes Camera” on page 5-16
“Define a Tracking Camera” on page 5-17
“Select a Dynamic Camera” on page 5-17

Start a New Camera Definition

If you are new to dynamic cameras, see “Visualization Cameras” on page 5-10. To start a new dynamic camera:

1. Simulate the model that you want to add the camera to.

   Dynamic cameras exist only in the models that you define them in.

2. In the Mechanics Explorer menu bar, select **Tools > Camera Manager**.

   Camera Manager opens with a list of previously created dynamic cameras. The list is by empty until you create your first camera.

3. In Camera Manager, click the button.

   Camera Manager switches to a camera definition view that lets you select the camera mode and specify the camera motion.

4. In the **Camera Name** field, enter a name for your camera.

   Make the camera name descriptive so that you can later identify it when selecting an active camera from the Mechanics Explorer visualization context-sensitive menu.

Complete the camera definition by selecting the camera mode and specifying the camera motion. See:

- “Define a Keyframes Camera” on page 5-16 to define the camera motion in `Keyframes` mode. Keyframes are viewpoints that you specify at various playback times and that Simscape Multibody software interpolates to obtain smooth camera trajectories.

- "Define a Tracking Camera" on page 5-17 to complete the camera definition in `Tracking` mode. Tracking constraints include position, aim, and up vector constraints that you specify relative to coordinate frames in a model.

Define a Keyframes Camera

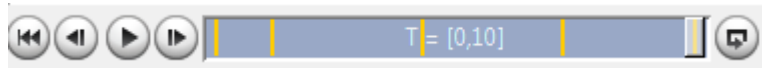
1. In Camera Manager, set the **Mode** parameter to `Keyframes`.

   Camera Manager switches to a `Keyframes` view that lets you define the camera keyframes.
2. In the Mechanics Explorer toolstrip, set the playback time for the current keyframe.

   Drag the playback slider to the desired point in the animation timeline. Alternatively, enter the time directly in the playback time counter.
3. In the visualization pane or tile, manipulate the model viewpoint for your keyframe.

   Use the Rotate, Roll, Pan, and Zoom buttons to manipulate the model viewpoint. Use the preset view buttons to obtain standard views such as front, side, or isometric.
4. In the Camera Manager `Keyframes` window, click the **Set** button.

   Playback must be paused or stopped. Camera Manager commits the keyframe to the camera. The playback slider identifies the keyframe with a colored line marker located at the specified playback time.

5. Set new keyframes as in steps 2–4 until you are satisfied with the camera motion.

   Simscape Multibody software transitions between keyframes using the smooth interpolation method of the `pchip` MATLAB function to yield the final camera motion.
6. Click the **Save** button in the camera definition and main panes of Camera Manager.

   Camera Manager saves the camera and its motion to the model. The visualization context-sensitive menu adds the camera to the list of available cameras.

To edit an existing keyframe, use the **Previous** and **Next** buttons to navigate to the keyframe you want to edit. Then, repeat the procedure for adding a keyframe. Use the colored markers in the playback slider to identify the existing keyframes in your dynamic camera.

Click the **Remove** button if you want to delete the current keyframe. Click the **Save** button in the main pane to commit your changes to the camera.

Define a Tracking Camera

1. In Camera Manager, set the **Mode** parameter to `Tracking`.

   Camera Manager switches to a `Tracking` view that lets you define the camera constraints—position, aim, and up vector—relative to frames in your model.

2. In the Camera Manager tracking window, set the camera **Position**, **Aim**, and **Up Vector** constraints:

   a. In the tree view or visualization pane, select a frame to constrain the camera to.

      If using the visualization pane, click a frame icon. If using the tree view pane, click a frame node. It is not enough to click the body that the frame belongs to.

   b. Click the **Use Selected Frame** button to constrain the camera motion to the frame.

      If you accidentally select the wrong frame, pick a new frame and click the **Use Selected Frame** button again.

   c. For the **Aim** and **Up Vector** dropdown lists, select how to constrain the camera:

      - The **Position** constraint fixes the camera to the frame origin only and has no options dropdown list.
      - The **Aim** constraint provides the option to aim the camera at the frame origin or along a selected frame axis.
      - The **Up vector** constraint provides the option to align the up vector along a selected frame axis.

Select a Dynamic Camera

The dynamic cameras that you create through Camera Manager are by default inactive during animation playback. To set a particular camera as the active camera for a visualization pane, use the visualization pane context-sensitive menu. You can perform this task separately for each visualization pane that you have open in Mechanics Explorer:

1 Right-click the visualization pane or tile whose camera you want to switch.

The visualization context-sensitive menu opens up.

2 Select **Cameras** and, from the cameras list, select the desired camera.

The model viewpoint switches to that provided by the selected camera.

Selective Model Visualization

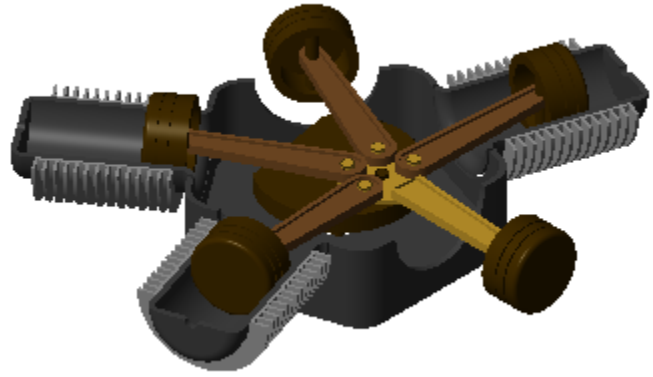
In this section...
“What Is Visualization Filtering?” on page 5-19
“Changing Component Visibility” on page 5-20
“Visualization Filtering Options” on page 5-21
“Components You Can Filter” on page 5-21
“Model Hierarchy and Tree Nodes” on page 5-22
“Filtering Hierarchical Subsystems” on page 5-22
“Updating Models with Hidden Nodes” on page 5-23
“Alternative Ways to Enhance Visibility” on page 5-24

What Is Visualization Filtering?

A multibody model can get so complex that you cannot easily tell its components apart. Solids, bodies, and multibody subsystems often hide behind each other, hindering your efforts to examine geometry, pose, and motion on model update or during simulation.

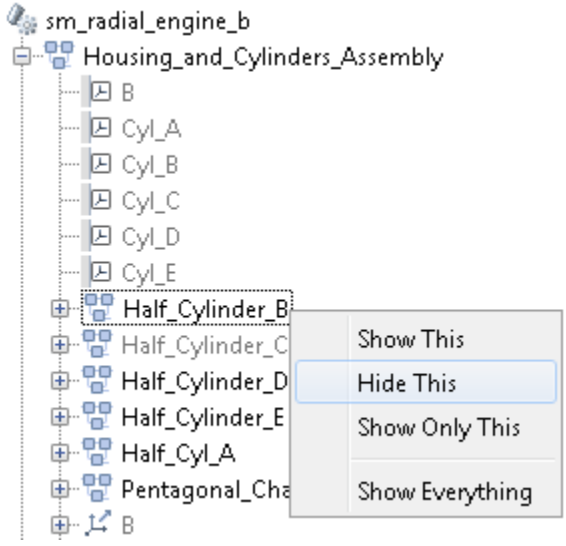
Visualization filtering is a Mechanics Explorer feature that lets you selectively show and hide parts of your model. By showing only those parts that you want to see, you can more easily discern any components placed within or behind other components—such as an engine piston traveling inside a cylinder casing.

The figure shows an example of visualization filtering. Two cylinders, one at the front and one at the rear, are hidden in the model visualization of the sm_radial_engine featured example. For a tutorial showing how to use visualization filtering, see “Selectively Show and Hide Model Components” on page 5-25.



Changing Component Visibility

You can show and hide components through a context-sensitive menu accessible in the tree-view pane of Mechanics Explorer. Right-click a model-tree node to open the menu and select the desired option. The figure shows the visualization filtering menu.



Visualization Filtering Options

The visualization filtering menu provides four options for you to select from:

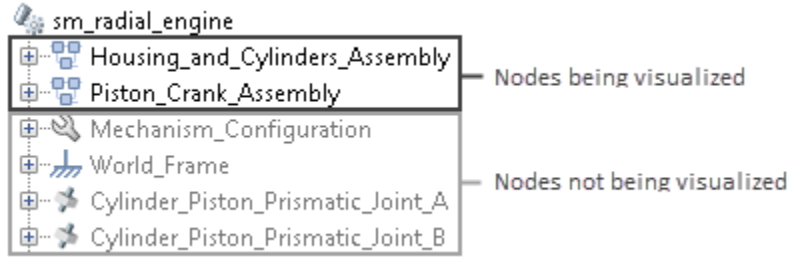
- `Show This` — Enable visualization for the selected component. This option has no effect if the component is already visible.
- `Hide This` — Disable visualization for the selected component. This option has no effect if the component is already hidden.
- `Show Only This` — Enable visualization for the selected component and disable visualization for the remainder of the model. This option has no effect if the selected component is already the only component visible.
- `Show Everything` — Enable visualization for every component in the model. This option has no effect if every component in the model is already visible.

Components You Can Filter

You can filter the visualization of any component with solid geometry. This includes individual solids, bodies, and multibody subsystems. In general, if a subsystem contains at least one Solid block, then you can switch its visualization on and off.

Frames, joints, constraints, forces, and torques have no solid geometry to visualize and therefore cannot be filtered in Mechanics Explorer. You can still open the visualization filtering context-sensitive menu by right-clicking these nodes, but only one option is active—`Show Everything`.

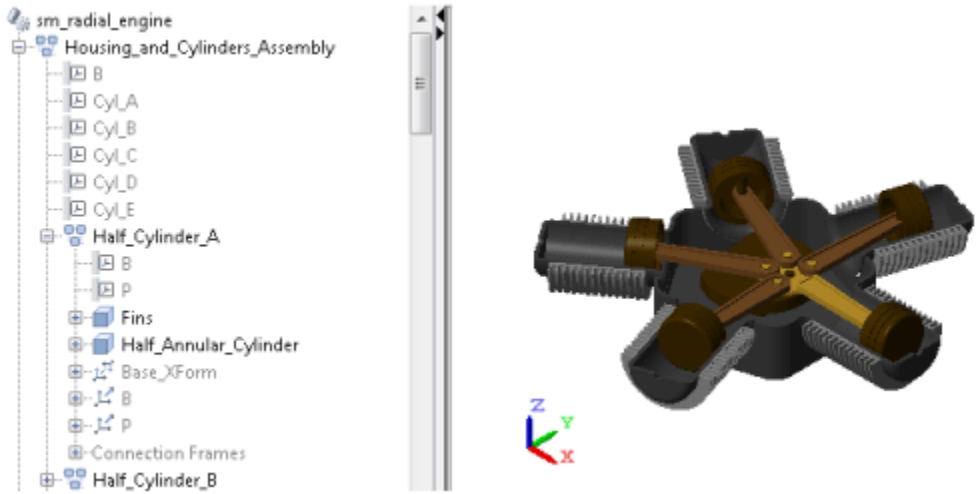
The tree-view pane identifies any node not being visualized by graying out its name. This includes nodes that can be visualized but are currently hidden and nodes that cannot be visualized at all. The figure shows an example with the grayed-out names of nodes not being visualized.



Model Hierarchy and Tree Nodes

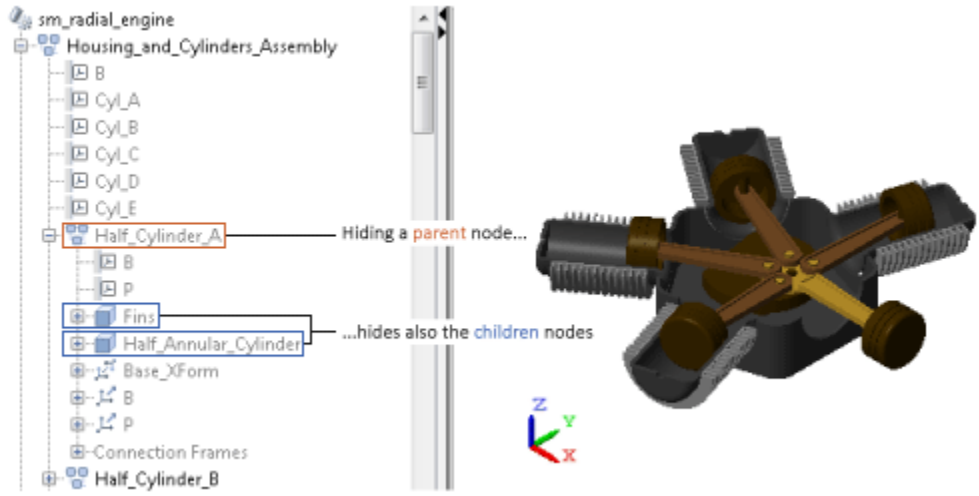
Multibody models are hierarchical in nature. They often contain multibody subsystems comprising rigid-body subsystems, each with one or more solids. The tree-view pane of Mechanics Explorer represents such a model structure through nodes arranged hierarchically. A node is a parent node if it contains other nodes, and a child node if it appears under another node. Nodes can simultaneously be children to some nodes and parents to others.

The figure shows portion of the tree-view pane of the sm_radial_engine featured example. The Half_Cylinder_A node is a child to the Housing_and_Cylinder_Assembly node and a parent to the Fins and Half_Annular_Cylinder nodes.



Filtering Hierarchical Subsystems

Any changes to the visualization settings of a tree node apply equally to all children of that node, if any. Nodes higher up in the model tree are not affected. As shown in the following figure, hiding the Half_Cylinder_A node in the sm_radial_engine model causes the Fins and Half_Annular_Cylinder nodes (children nodes) to hide, but not the Housing_and_Cylinders_Assembly node (parent node) or the Half_Cylinder_B node (sibling node).



If you want to show part of a subsystem you have previously hidden, you can change the visibility settings for the children nodes that you want to show. For example, if after hiding the Half_Cylinder_A node, you want to show the Fins child node, you need only right-click the Fins node and select `Show This`. Such changes have no effect on the remainder of the Half_Cylinder_A parent node.

Updating Models with Hidden Nodes

The following apply when you update or simulate a model with previously hidden nodes:

- If the model remains unchanged, the node visibility settings remain unchanged—that is, the hidden nodes remain hidden and the visible nodes remain visible. This happens even if you save the Mechanics Explorer configuration to the model by clicking the icon.
- If you close Mechanics Explorer before updating the model, Mechanics Explorer reopens with all nodes visible, including any nodes you may have previously hidden.
- If you change the name of a block corresponding to a hidden node—e.g., a Solid block or a Subsystem block containing a Solid block—the hidden node and any children nodes it may have become visible.

- If you uncomment a block that corresponds to a hidden node and that you had previously commented out, the hidden node and any children nodes it may have become visible.
- If you add to a hidden Subsystem block a Solid block or another Subsystem block with a Solid block, the child node corresponding to the new block becomes visible upon model update but the visibility of the hidden parent node remains unchanged.
- If you change the parameters of a block corresponding to a hidden node, that node and its children nodes retain their original visibility settings—that is, hidden nodes remain hidden and visible nodes remain visible.

Alternative Ways to Enhance Visibility

Visualization filtering is not the only approach you can use to enhance component visibility in a model. However, it is often the simplest. It is also the only approach that doesn't require you to modify the model in any way. Alternative approaches you can use include:

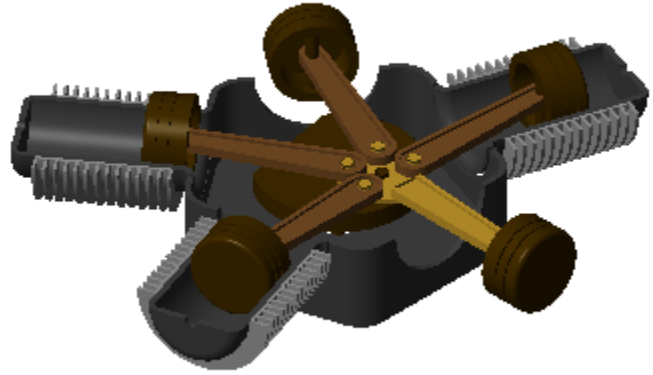
- Lowering the opacity of obstructive components—those obscuring other parts of the model—for example, making the cylinder encasing an engine piston transparent.
- Modeling obstructive components only in part—for example, treating engine cylinders as half-cylinders to preserve piston visibility during simulation.
- Omitting obstructive components altogether if they serve a purely aesthetic purpose and have no impact on model dynamics—for example, removing the cylinder subsystems from the sm_radial_engine featured example.
- Commenting out or through obstructive components if they serve a purely aesthetic purpose and have no impact on model dynamics—for example, removing the cylinder subsystems from the sm_radial_engine featured example.

Selectively Show and Hide Model Components

In this section...
“Visualization Filtering” on page 5-25
“Open Example Model” on page 5-26
“Update Example Model” on page 5-26
“Hide Half-Cylinder Subsystem” on page 5-27
“Show Solid in Hidden Subsystem” on page 5-28
“Show Only Piston Subsystem” on page 5-29
“Show Everything” on page 5-30

Visualization Filtering

Visualization filtering is a Mechanics Explorer feature that enables you to selectively show and hide solids, bodies, and multibody subsystems. This tutorial shows you how to use this feature to control the visualization of a Simscape Multibody model, for example, to observe a model component that might otherwise remain obstructed during simulation. For more information, see “Selective Model Visualization” on page 5-19.

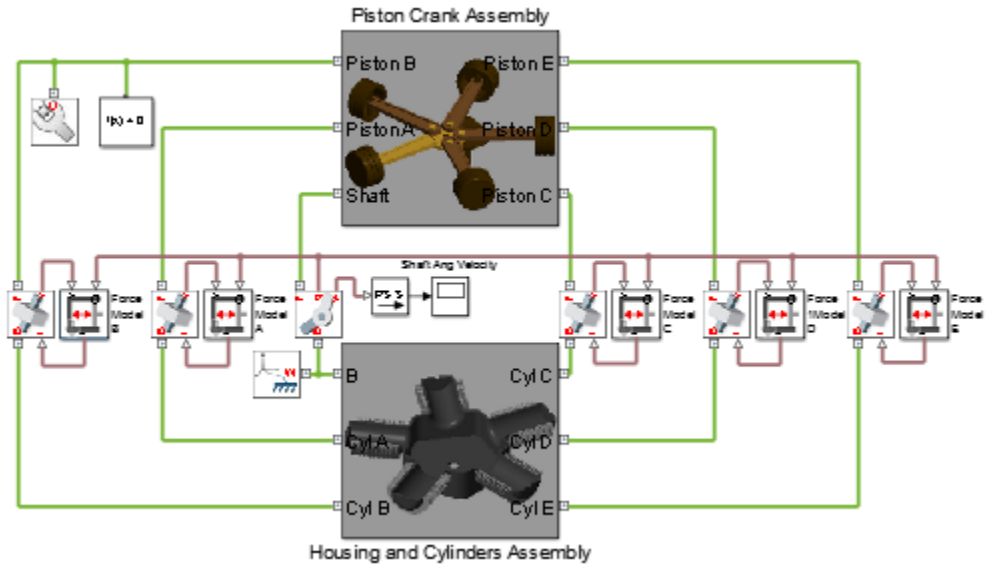


Radial Engine Visualization with Two Cylinders Hidden

Open Example Model

In this tutorial, you filter the visualization of the Simscape Multibody radial engine featured example. To open this model, at the MATLAB command prompt, enter `sm_radial_engine`.

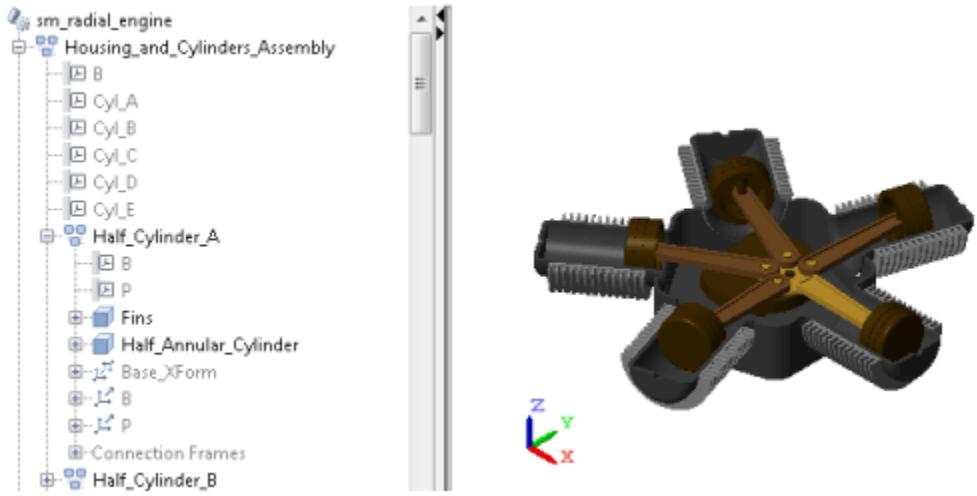
The model contains two top-level subsystems—the housing subsystem, named Housing_and_Cylinders_Assembly, and the piston subsystem, named Piston_Crank_Assembly. The housing subsystem contains five half cylinders. The piston subsystem contains five pistons that travel inside the half cylinders.



Radial Engine Block Diagram

Update Example Model

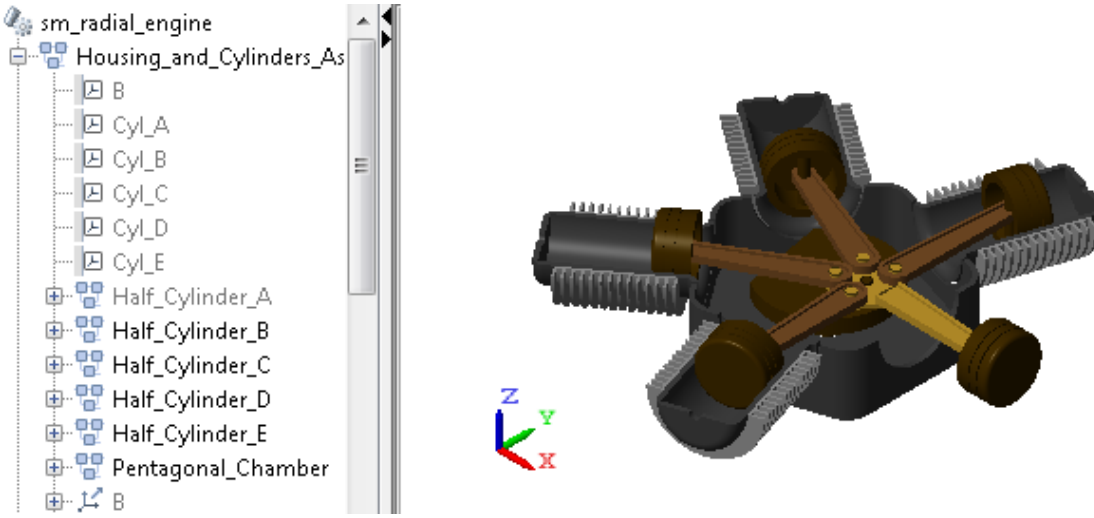
To open Mechanics Explorer, the Simscape Multibody visualization utility you must first update the example model. To do this, in the Simulink menu bar, select **Simulation > Update Diagram** (Windows shortcut **Ctrl + D**). Note the tree-view pane on the left side of Mechanics Explorer. You access the visualization filtering menu by right-clicking a node on this pane.



Radial Engine Model Visualization

Hide Half-Cylinder Subsystem

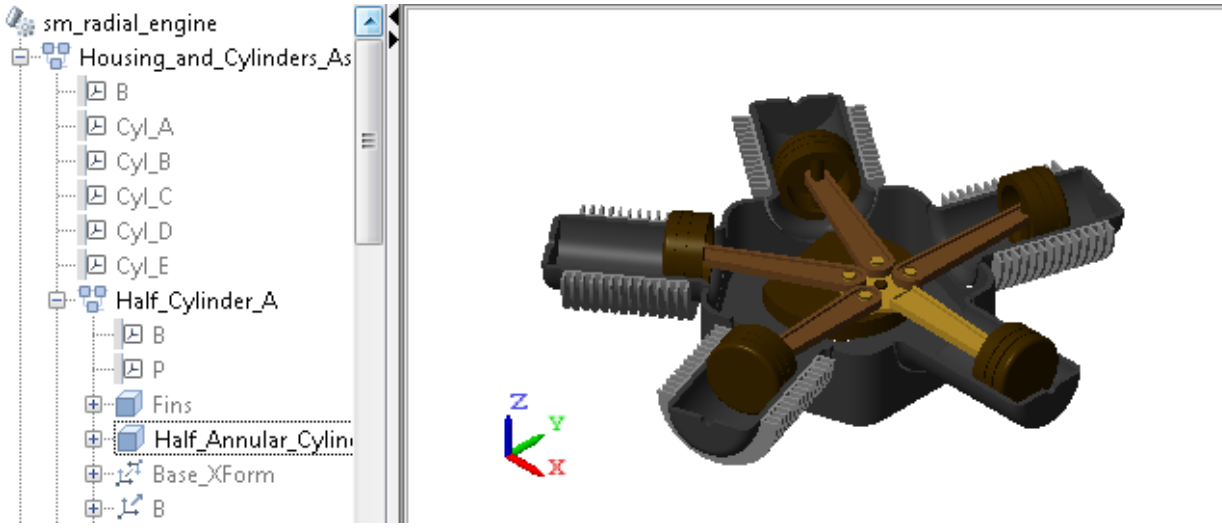
In the tree-view pane, expand the Housing_and_Cylinders_Assembly node. Right-click the Half_Cylinder_A node and select `Hide This`. Mechanics Explorer hides the half-cylinder subsystem and the solids it contains, corresponding to the nodes Fins and Half_Annular_Cylinder. The hidden-node names are grayed out in the tree-view pane. The figure shows the resulting model visualization.



Radial Engine with Hidden Half-Cylinder Subsystem

Show Solid in Hidden Subsystem

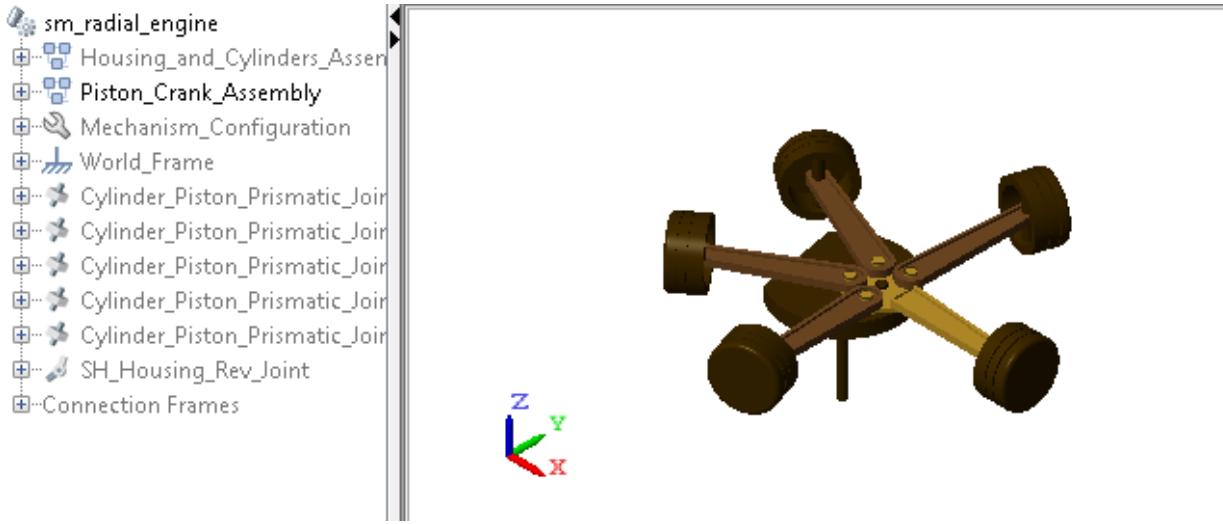
In the tree-view pane, expand the Half_Cylinder_A node. Then, right-click the Half_Annular_Cylinder node and select `Show This`. The half-cylinder solid is now visible, but the remainder of its parent of its parent subsystem—in this case, just the Fins solid—remains hidden. The newly visible half-cylinder node name is no longer grayed out in the tree-view pane. The figure shows the resulting model visualization.



Radial Engine with Visible Solid in Hidden Half-Cylinder Subsystem

Show Only Piston Subsystem

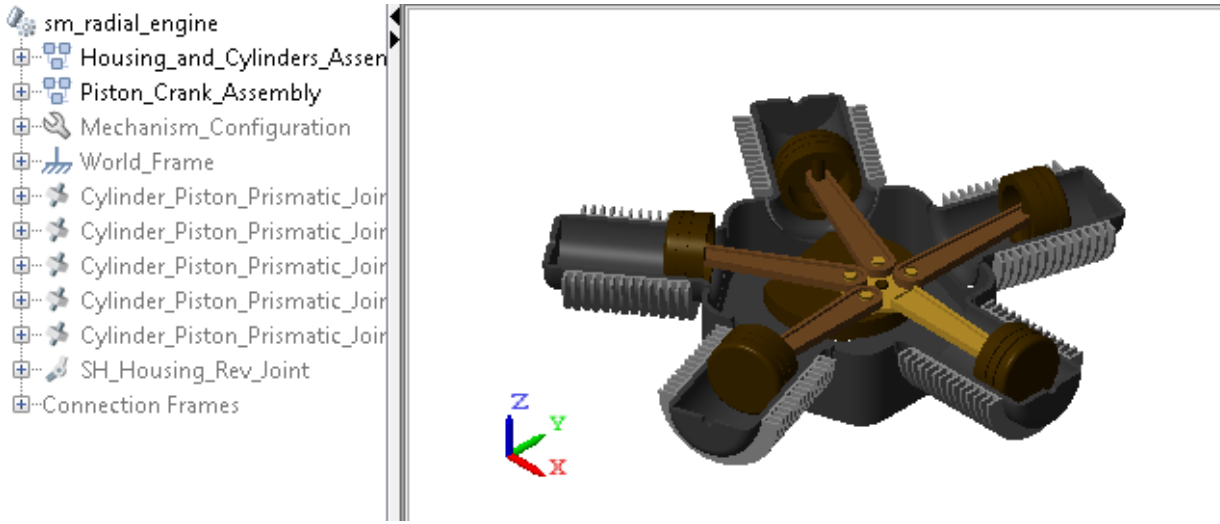
In the tree-view pane, collapse the Housing_and_Cylinders_Assembly node. Then, right-click the Piston_Crank_Assembly node and select `Show Only This`. Mechanics Explorer shows the selected node and hides the remainder of the model. In the tree-view pane, the name of the selected node is the only that is not grayed out. The figure shows the resulting model visualization.



Radial Engine with Only Piston Subsystem Visible

Show Everything

In the tree-view pane, right-click any node and select `Show Everything`. All hidden components become visible. The corresponding nodes are no longer grayed out in the tree-view pane. The figure shows the resulting model visualization.


sm_radial_engine
Housing_and_Cylinders_Assen
Piston_Crank_Assembly
Mechanism_Configuration
World_Frame
Cylinder_Piston_Prismatic_Joir
Cylinder_Piston_Prismatic_Joir
Cylinder_Piston_Prismatic_Joir
Cylinder_Piston_Prismatic_Joir
Cylinder_Piston_Prismatic_Joir
SH_Housing_Rev_Joint
Connection Frames
Z
Y
X

Visualize Simscape Multibody Frames

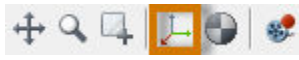
In this section...
“What Are Frames?” on page 5-32
“Show All Frames” on page 5-32
“Highlight Specific Frames” on page 5-33
“Visualize Frames via Graphical Markers” on page 5-35

What Are Frames?

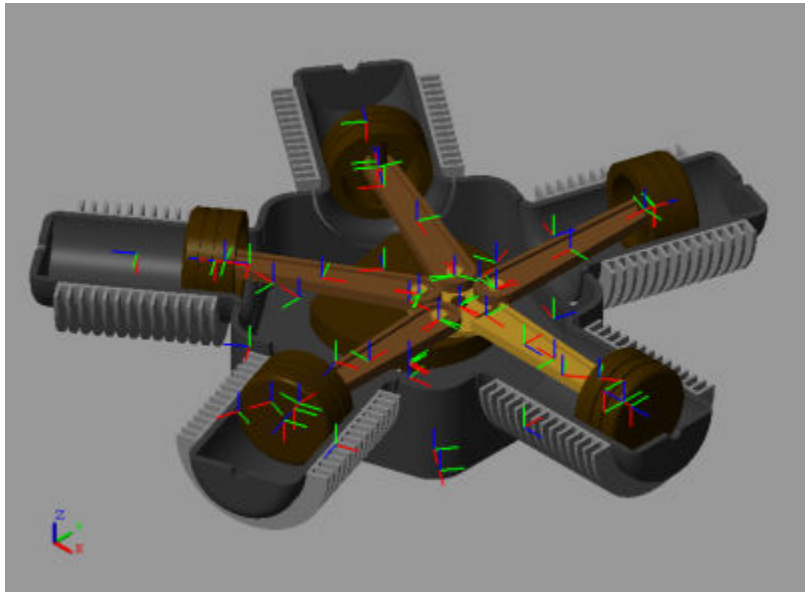
Simscape Multibody models are based on frames, abstract axis triads that contain all the position and orientation data in a model. These constructs enable you to connect solids into rigid bodies, assemble rigid bodies into mechanisms, and prescribe and sense forces, torques, and motion. Given their importance, then, it makes sense to visualize where and how you place different frames in a model.

Show All Frames

The easiest way to view the frames in your model is to toggle their visibility on. You can do this by clicking the **Toggle visibility of frames** icon in the Mechanics Explorer tool strip, shown in the following figure.



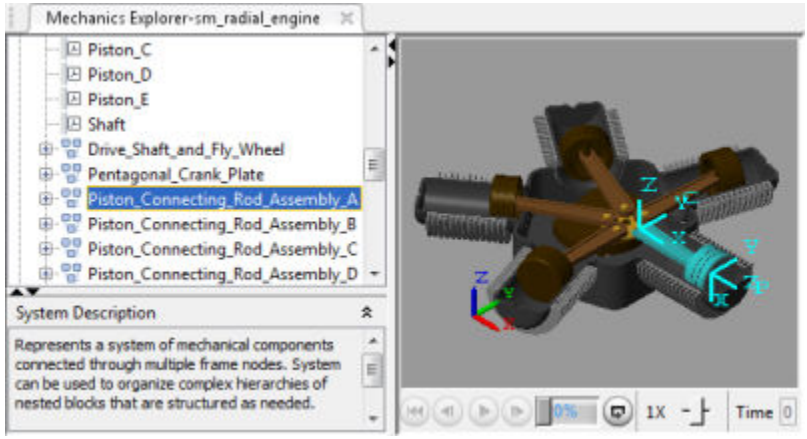
Alternatively, you can select **View > Show Frames** in the menu bar. Mechanics Explorer shows all the frames in your model, suiting this approach well for models with small numbers of frames. The figure shows a radial engine model with frame visibility toggled on.



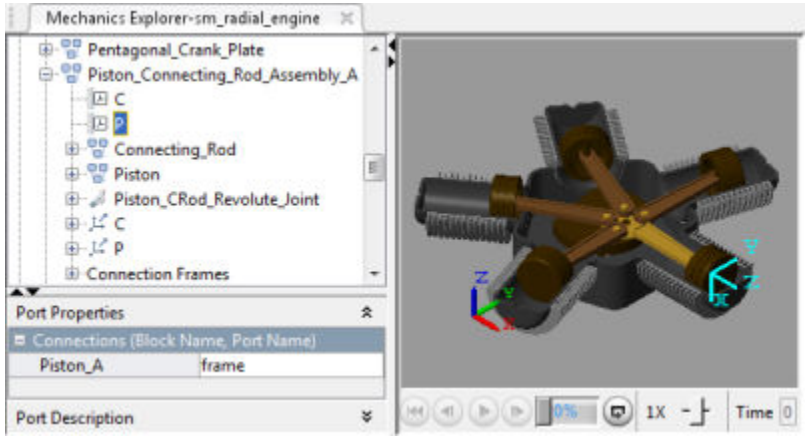
If your model has many frames, a different approach may be ideal, as toggling frame visibility may clutter the visualization pane with frames that you don't want to track.

Highlight Specific Frames

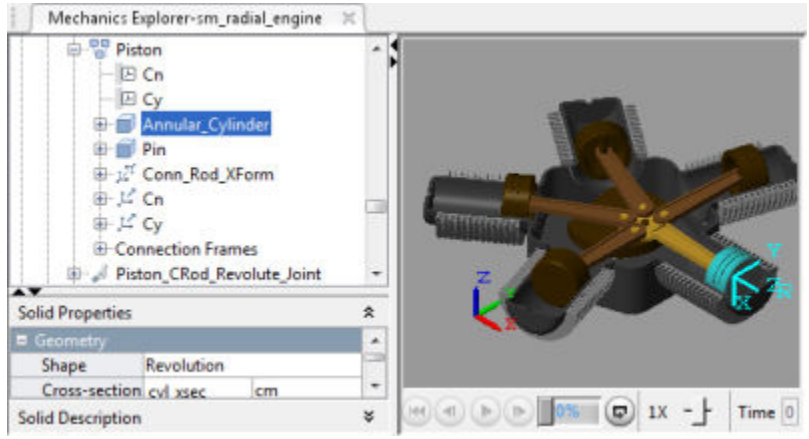
To view only the port frames of a block, including those of a subsystem block, you can select a node in the tree view pane. Mechanics Explorer highlights the port frames associated with the selected node using a turquoise color. The following figure shows an example in which one of the connecting rod assemblies in the radial engine model is highlighted in turquoise.



You can also select individual port frames, which you expose by expanding the tree nodes. For example, expanding the Piston_Connecting_Rod_Assembly_A node exposes the port frame P node, which you can then select in order to highlight that frame. The figure shows the result.

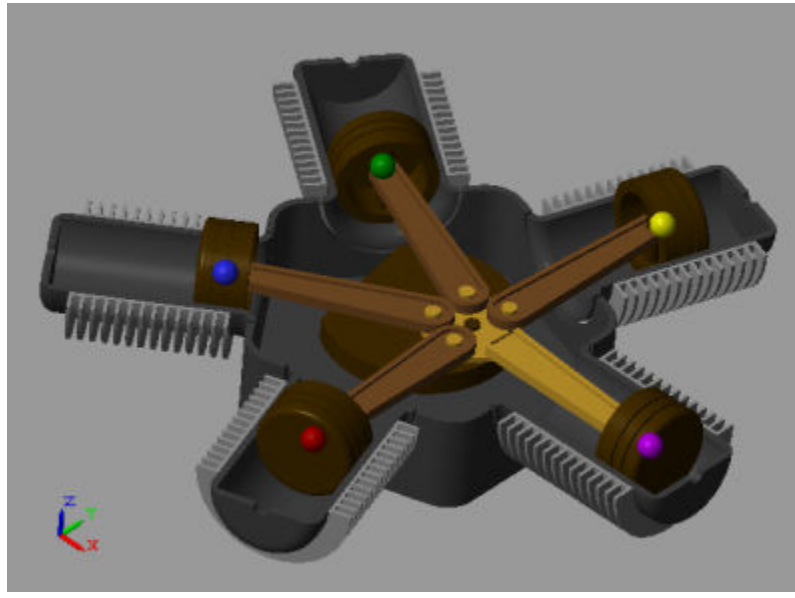


Finally, you can select individual solids directly in the visualization pane, highlighting their reference frames. The figure shows the result of selecting one of the piston solids directly. Mechanics Explorer highlights the solid and its reference frame, while the tree view pane reveals the associated Solid block name. This is the block that you need to change if you want to modify this particular solid.



Visualize Frames via Graphical Markers

If a frame in your model has special significance—e.g., if its origin is the point of application for an external force—you can connect to it a graphical marker. So that you can perform this task, the Body Elements library provides a Graphic block. Simply connect the block to the frame you want to visualize and select the marker type to use—sphere, cube, or frame. The figure shows the radial engine model with a sphere marker highlighting each of the piston connection frames.



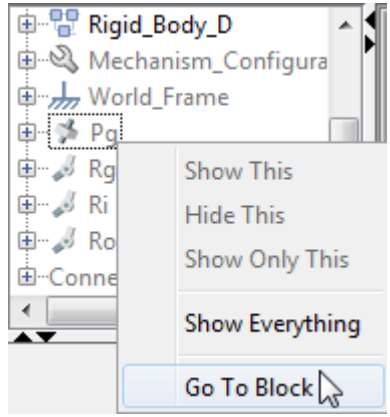
Go to a Block from Mechanics Explorer

The first indication that something is wrong in a model is often an unexpected result in the visualization pane. Unexpected results can include disparities in solid shape and size, incorrect translation and rotation transforms between solids, and even joints and constraints that fail to assemble.

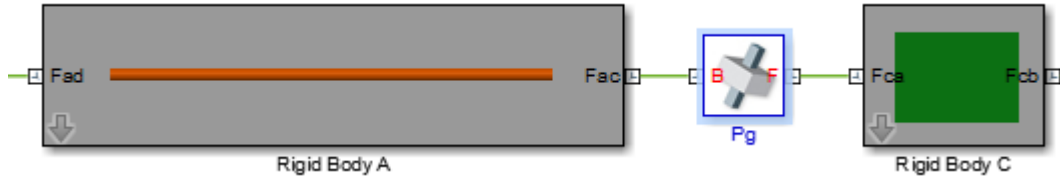
To help you troubleshoot such modeling issues, Mechanics Explorer enables you to go directly to a block associated with a node in the tree view pane. This feature helps you also to iterate on a model that is working properly, for example, if you want to replace a body subsystem with an alternative version.

To highlight a block corresponding to a Mechanics Explorer tree node:

1. In the tree view pane of Mechanics Explorer, right-click the node whose block you want to examine.



2. From the context-sensitive menu, select **Go to Block**. Simscape Multibody brings the block diagram to the front and highlights the block corresponding to the selected node.



For an example showing how to troubleshoot a model using Mechanics Explorer block highlighting, see “Troubleshoot an Assembly Error” on page 2-27.

Create a Model Animation Video

In this section...
“UI and Command-Line Tools” on page 5-39
“Before Creating a Video” on page 5-39
“Create a Video Using Video Creator” on page 5-39
“Create a Video Using smwritevideo” on page 5-40

UI and Command-Line Tools

You can create a model animation video interactively, using the **Video Creator** tool, or programmatically, using the `smwritevideo` function. The tool and function provide equivalent ways to perform the same task. Use the tool to more intuitively configure and create a video. Use the function for your command-line workflows, e.g., to automate video capture following model simulation.

Before Creating a Video

- Mechanics Explorer must be set to open on model update.

  You can view and change the current setting in the **Simscape Multibody** > **Visualization** tab of the Model Configuration Parameters window.
- Only the active visualization tile in Mechanics Explorer is recorded.

  A visualization tile is a subdivision of the Mechanics Explorer visualization pane that shows a specific view of the model. The active tile is demarcated by a colored bounding box.
- The video viewpoint is always that of the active tile.

  To change the model viewpoint in the recorded video, you must change the viewpoint of the active tile. Use a dynamic camera for a moving point or the global camera for a static viewpoint.

Create a Video Using Video Creator

1 Simulate the model to record.

Animations and the videos generated from them are based on your model simulation data.

2 In Mechanics Explorer, select **Tools > Video Creator**.

Video Creator relies on your model visualization and is accessible through Mechanics Explorer only.

3 In Video Creator, specify the desired video parameters.

Video parameters that you can modify include the video frame rate, frame size, playback speed ratio, and video file format.

4 Click the **Create** button.

Video Creator generates a model animation video and saves it with the specified name in the specified folder.

5 Save the model animation video in a folder for which you have write privileges.

Create a Video Using smwritevideo

1 Simulate the model to record. E.g.,

```
modelName = 'sm_dump_trailer';
sim(modelName);
```

Simscape Multibody software simulates the dump trailer featured example.

2 Define any video parameters that deviate from your current Video Creator settings. E.g.,

```
fps = 60; speedRatio = 2;
```

All remaining video parameters are configured as specified in Video Creator.

3 Define the desired name and path of the animation video.

```
videoName = 'dump_trailer_animation'
```

In the absence of a path, the function saves the video in the current folder. You must have write privileges to the folder in order to save the video.

4 Call the `smwritevideo` function with the video name and parameters as function arguments.

```
smwritevideo(modelName,videoName,...
'PlaybackSpeedRatio',speedRatio,'FrameRate',fps);
```

The `smwritevideo` function creates a video of the model animation and saves it with the name `dump_trailer_animation` in the current folder.

Multibody Model Import

CAD and URDF Model Import

- "CAD Translation" on page 6-2
- "Install the Simscape Multibody Link Plug-In" on page 6-12
- "Import a CAD Assembly Model" on page 6-15
- "Import a Robotic Arm CAD Model" on page 6-17
- "Onshape Import" on page 6-21
- "Import an Onshape Humanoid Model" on page 6-27
- "URDF Primer" on page 6-31
- "URDF Import" on page 6-41
- "Import a URDF Humanoid Model" on page 6-50

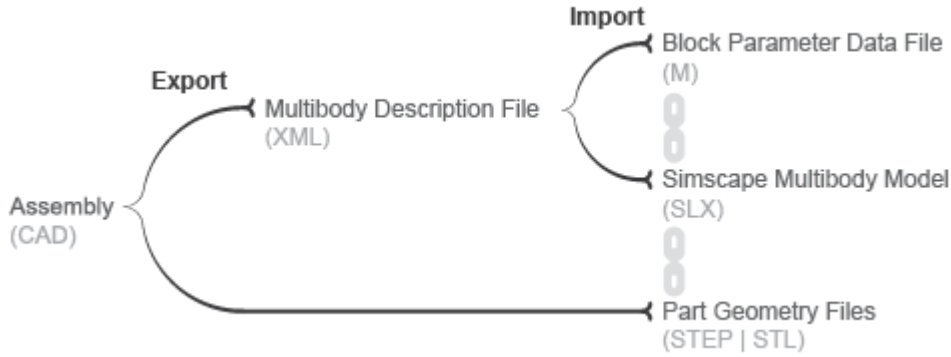
CAD Translation

In this section...
“Translating a CAD Model” on page 6-2
“What’s in a Translated Model?” on page 6-3
“What’s in a Data File?” on page 6-6
“Exporting a CAD Model” on page 6-7
“Importing a CAD Model” on page 6-8
“Simplifying Model Topology” on page 6-9
“Updating an Existing Data File” on page 6-10

Translating a CAD Model

You can translate a CAD model into an equivalent Simscape Multibody block diagram. The conversion relies on the `smimport` function featuring an XML multibody description file name as its central argument. The XML file passes to Simscape Multibody software the data it needs to recreate the original model—or an approximation of it if unsupported constraints exist in the model.

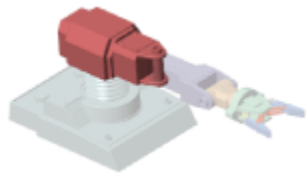
You translate a CAD model in two steps—export and import. The export step converts the CAD assembly model into an XML multibody description file and a set of STEP or STL part geometry files. The import step converts the multibody description and part geometry files into an SLX Simscape Multibody model and an M data file. The model obtains all block parameter inputs from the data file.



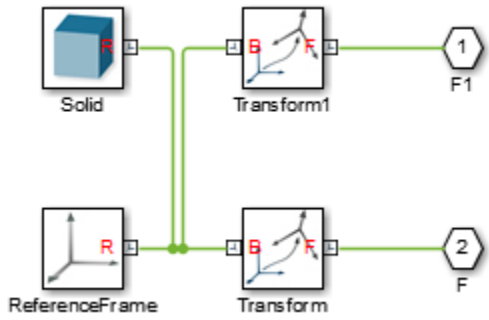
CAD Translation Steps

What's in a Translated Model?

The translated model represents the CAD parts—referred to as bodies in Simscape Multibody software—using Simulink subsystems that comprise multiple Solid and Rigid Transform blocks. The Solid blocks provide the body geometries, inertias, and colors. The Rigid Transform blocks provide the frames with the required poses for connection between bodies.

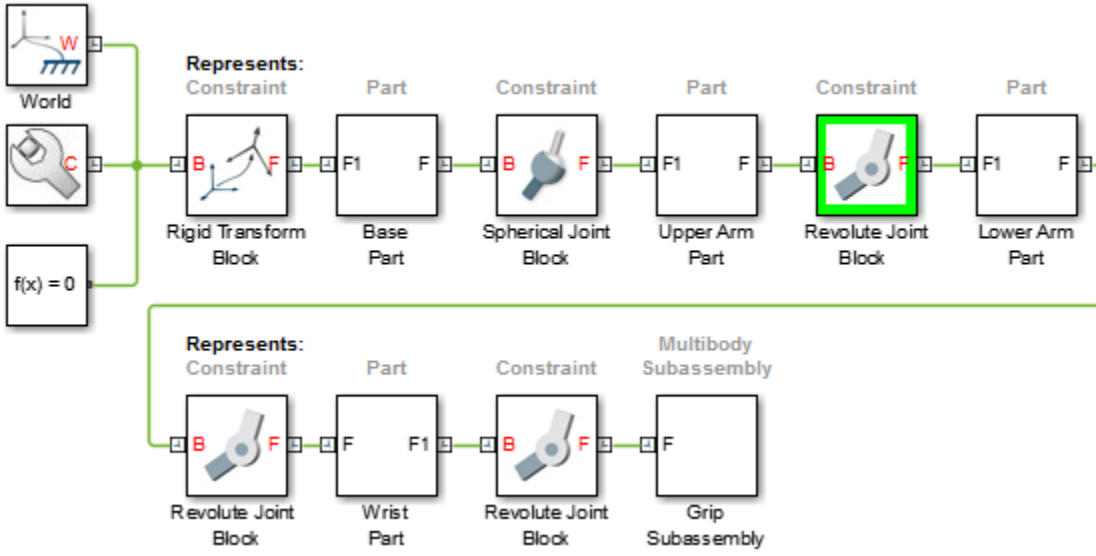


Consider the upper arm body of a CAD robotic arm model, shown in the figure. The Simulink subsystem for this body consists of one Solid block connected to a pair of Rigid Transform blocks. The Solid block provides the reference to the upper arm geometry file and the inertial properties derived from the CAD model. The Rigid Transform blocks provide the frames for connection to the robot base and lower arm bodies.



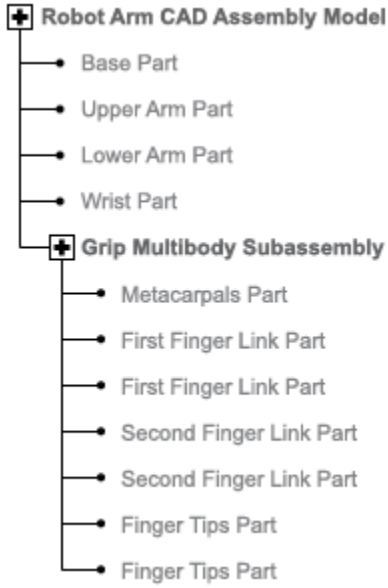
Simulink Subsystem Representing Upper Arm Body

CAD joints, constraints, and mates translate into Simscape Multibody software as combinations of joint and constraint blocks. In the CAD robotic arm example, the constraints between the upper arm and the lower arm translate into a Revolute Joint block. This block sits between the Simulink Subsystem blocks that represent the upper arm and lower arm bodies.



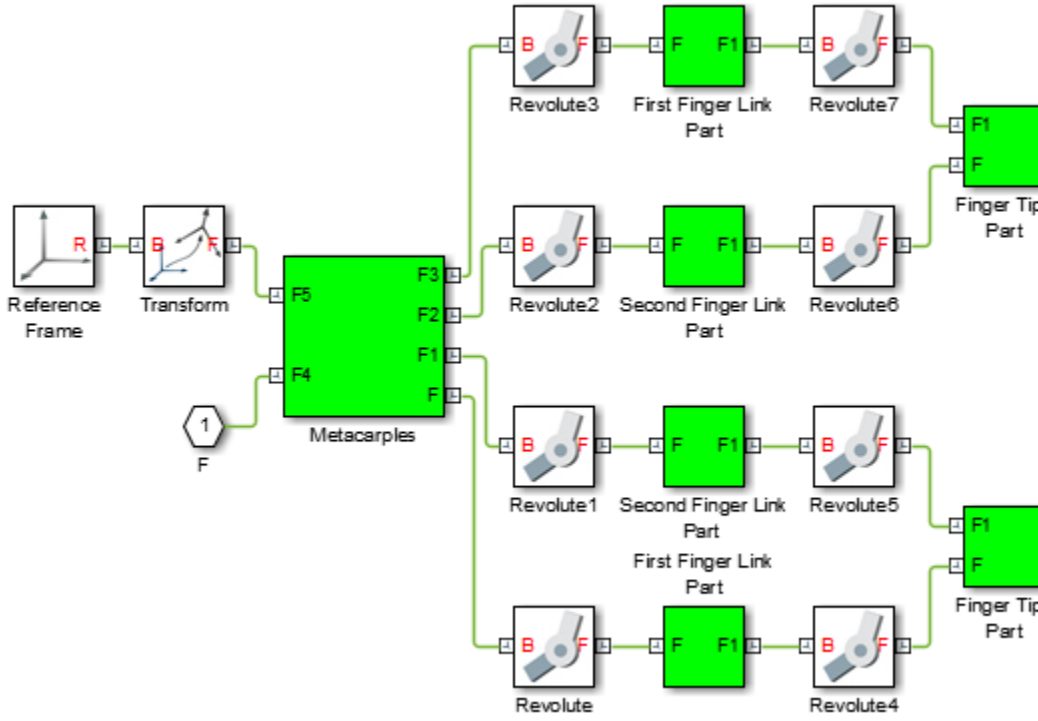
Simulink Subsystem Representing Upper Arm Body

By default, the translated model preserves the structural hierarchy of the original CAD model. If the source model is a CAD model with multibody subassemblies, the subassemblies convert in Simscape Multibody software into multibody Simulink subsystems. Consider again the CAD robotic arm model. The model contains a grip multibody subassembly with seven bodies, shown schematically in the figure.



CAD Robotic Arm Model Hierarchy

During translation, the grip subassembly converts into a Simulink subsystem with seven Simulink subsystems, one for each body.



Multibody Simulink Subsystem with Body Simulink Subsystems

What's in a Data File?

Blocks in the translated model are parameterized in terms of MATLAB variables defined in the data file. These variables are stored in structure arrays named after the various block types. The structure arrays are nested in a parent data structure named `smiData` or a custom string that you specify.

Consider an imported model with a data structure named `smiData`. If the model contains Revolute Joint blocks, the parameter data for these blocks is the structure array `smiData.RevoluteJoint`. This structure array contains a number of data fields, each corresponding to a different block parameter.

The structure array fields are named after the block parameters. For example, the position state target data for the Revolute Joint blocks is in a field named

`Rz_Position_Target`. If the model has two Revolute Joint blocks, this field contains two entries—`smiData.RevoluteJoint(1).Rz_Position_Target` and `smiData.RevoluteJoint(2).Rz_Position_target`.

Each structure array index corresponds to a specific block in the imported model. The index assignments can change if you regenerate a data file from an updated XML multibody description file. The `smimport` function checks the prior data file, when specified, to ensure the index assignments remain the same. See "Updating an Existing Data File" on page 6-10.

Exporting a CAD Model

You can technically export a CAD assembly model from any CAD application. The Simscape Multibody Link CAD plug-in provides one means to export a model in a valid XML format. The plug-in is compatible with three desktop CAD applications: SolidWorks®, PTC® Creo™, and Autodesk® Inventor®. The plug-in generates not only the XML multibody description file but also any geometry files required for visualization in the final translated model.

The Simscape Multibody `smexportonshape` function provides another means to export a CAD model, from a cloud application named Onshape®. This function exports in a format compatible with the newer Simscape Multibody blocks only. See "Onshape Import" on page 6-21 for more information on exporting (and importing) CAD models from an Onshape account.

If you use an unsupported CAD application, you can create a program that uses the CAD API and Simscape Multibody XML schema to generate the multibody description and part geometry files. This task requires knowledge of XML documents, XSD schema definitions, and CAD APIs. See the schema website for the XSD schema definitions. See MATLAB Central for an example program built on the SolidWorks CAD API.

If a URDF converter exists for your CAD application, you may be able to export your model in URDF format and import the URDF file into the Simscape Multibody environment. Note, however, that the URDF specification forbids closed-chain model topologies, such as those of four-bar linkages and gear assemblies. For more information, see "URDF Import" on page 6-41.

A Note About Export Errors

If the Simscape Multibody Link plug-in cannot export a part geometry file or translate a CAD constraint set, the software issues an error message. The error message identifies

the bodies with missing geometry files and any unsupported constraints. You can import the generated XML multibody description file into Simscape Multibody software, but the resulting model may not accurately represent the original CAD assembly model.

Importing a CAD Model

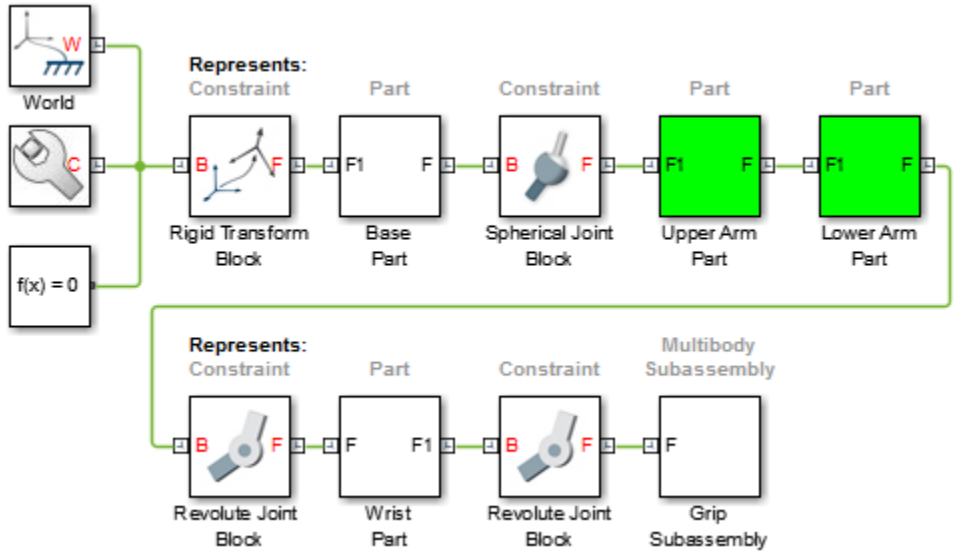
You import an XML multibody description file using the Simscape Multibody `smimport` function in its default import mode. The function parses the file and generates a Simscape Multibody model and associated data file. For step-by-step instructions on to import a CAD assembly model via its XML multibody description file, see "Import a CAD Assembly Model" on page 6-15.

Note Starting with software version R2017b, the Simscape Multibody Link plug-in exports in an XML format compatible only with Simscape Multibody Second Generation software. You must import all such XML files using the `smimport` function. Models generated with this function comprise only second-generation blocks—those accessible by entering the command `sm_lib` at the MATLAB command prompt.

However, if you have any older XML files in a format compatible with Simscape Multibody First Generation software, you can continue to import them using the `mech_import` function. Models generated with this function comprise only first-generation blocks—those accessible by entering the command `mechlib` at the MATLAB command prompt.

CAD Import Errors

If a part geometry file is invalid or missing, the corresponding body cannot show in the Simscape Multibody visualization utility. If a CAD assembly model contains an unsupported constraint combination between bodies, Simscape Multibody software joins the bodies with a rigid connection. The rigid connection can take the form of a direct frame connection line, Rigid Transform block, or Weld Joint block.



Rigid Connection Due to Unsupported Constraints

If Simscape Multibody software cannot translate a CAD constraint combination, it issues a warning message on the MATLAB command window identifying the affected bodies and their connection frames. For example:

```
Warning: The set of constraints between upperarm_1_RIGID and forearm_1_RIGID
could not be mapped to a joint. A rigid connection has been added between port F
of upperarm_1_RIGID and port F1 or forearm_1_RIGID for these constraints.
```

Simplifying Model Topology

You can import a CAD model with a simplified topology. So that you can do this, the `smimport` function provides the `ModelSimplification` argument. You can set this argument to:

- `bringJointsToTop` to group each set of rigidly connected parts into a new subsystem and promote all joints to the top level of the model hierarchy.
- `groupRigidBodies` to group rigidly connected parts into subsystems (and leave joints in their original places in the model hierarchy).

- `None` to import the model as is, without simplification.

Use the `bringJointsToTop` or `groupRigidBodies` option if your CAD model has many rigidly connected components, such as nuts and bolts, that you prefer to group together—for example, to more intuitively grasp the key components of the model at a glance of the block diagram.

Use the `bringJointsToTop` option if your CAD model has joints inside subassemblies and you prefer to expose them at the top level—for example, to work with joint actuation and sensing signals without having to search for the joints inside different subsystems.

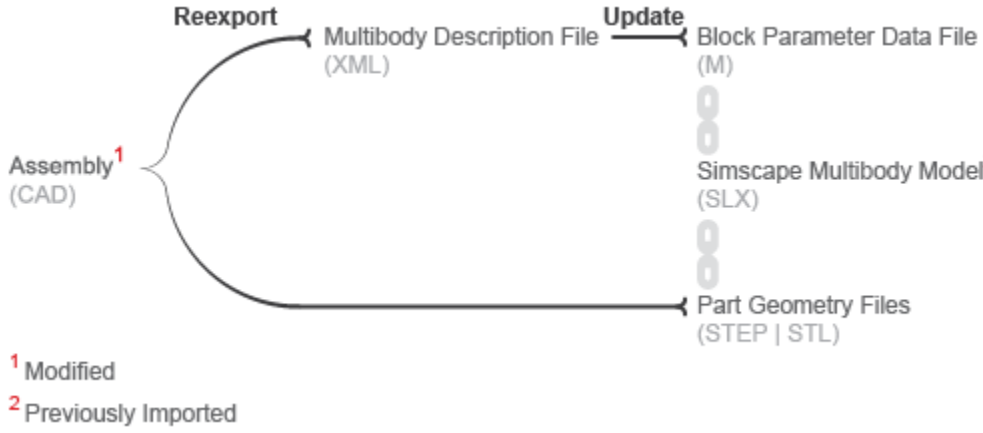
Note that model simplification is available for CAD import only. URDF models have flat topologies with little need for topology simplification.

Updating an Existing Data File

You regenerate the data file for a previously imported model by running the `smimport` function in `dataFile` mode. You specify this mode using the optional `ImportMode Name,Value` pair argument. The function uses the prior data file to keep the mapping between structure array indices and blocks consistent.

Before regenerating a data file, you must export a new XML multibody description file from the updated CAD assembly model. The `smimport` function uses the data in the new multibody description file to generate the new data file.

The function does not update the block diagram when run in `dataFile` mode. If you add or delete bodies in the source CAD assembly model, you must manually add or delete the corresponding blocks in the previously imported model.


Reexport
Multibody Description File
(XML)
Update
Block Parameter Data File
(M)
Assembly[1]
(CAD)
Simscape Multibody Model[2]
(SLX)
Part Geometry Files
(STEP | STL)
[1] Modified
[2] Previously Imported

CAD Update

Install the Simscape Multibody Link Plug-In

In this section...
“What is Simscape Multibody Link?” on page 6-12
“Alternatives to Simscape Multibody Link” on page 6-12
“Before You Begin” on page 6-13
“Step 1: Get the Installation Files” on page 6-13
“Step 2: Run the Installation Function” on page 6-13
“Step 3: Register MATLAB as an Automation Server” on page 6-14
“Step 4: Enable the Simscape Multibody Link Plug-In” on page 6-14

What is Simscape Multibody Link?

Simscape Multibody Link is a plug-in that you install on your CAD application to export your CAD assembly models. The plug-in generates the files that you need to import the model into the Simscape Multibody environment (using the `smimport` function). You can install the plug-in on three CAD applications:

- SolidWorks
- Autodesk Inventor®
- PTC Creo

Alternatives to Simscape Multibody Link

You can export an assembly model from Onshape CAD software using the `smexportonshape` function. This function generates the same files as the Simscape Multibody Link plug-in—one an XML multibody description file, the rest a set of body geometry files.

If you use a different CAD application, you can create a custom application based on the published Simscape Multibody XML schema. This approach requires some knowledge of XML. See the schema web page for an example.

If you can export a multibody model in URDF format, then you can also import that model using the `smimport` function. This approach may suit you if your CAD application is not among those supported and you have little experience with XML schemas.

Before You Begin

You must have a valid MATLAB license and one of the supported CAD applications:

- Autodesk Inventor software
- PTC Creo software
- SolidWorks software

Your MATLAB and CAD installations must have the same system architecture—e.g., Windows 64-bit.

Step 1: Get the Installation Files

1. Go to the Simscape Multibody Link download page.
2. Follow the prompts on the download page.
3. Save the zip archive and MATLAB file in a convenient folder.

   Select the file versions matching your MATLAB release number and system architecture—e.g., release R2015b and Win64 architecture. Do not extract the zip archive.

Step 2: Run the Installation Function

1. Run MATLAB as administrator.
2. Add the saved installation files to the MATLAB path.

   You can do this by entering `addpath('foldername')` at the MATLAB command prompt. Replace `foldername` with the name of the folder in which you saved the installation files—e.g., `C:\Temp`.
3. At the MATLAB command prompt, enter `install_addon('zipname')`.

   Replace `zipname` with the name of the zip archive—e.g., `smlink.r2015b.win64.zip`.

Step 3: Register MATLAB as an Automation Server

Each time you export a CAD assembly model, the Simscape Multibody Link plug-in attempts to connect to MATLAB. For the connection to occur, you must register MATLAB as an automation server. You can do this in two ways:

- In a MATLAB session running in administrator mode — At the command prompt, enter `regmatlabserver`.
- In an MS-DOS window running in administrator mode — At the command prompt, enter `matlab -regserver`.

Step 4: Enable the Simscape Multibody Link Plug-In

Before you can export an assembly, you must enable the Simscape Multibody Link plug-in on your CAD application. To do this, see:

- “Enable Simscape Multibody Link Inventor Plug-In” (Simscape Multibody Link)
- “Enable Simscape Multibody Link Creo-Pro/E Plug-In” (Simscape Multibody Link)
- “Enable Simscape Multibody Link SolidWorks Plug-In” (Simscape Multibody Link)

Import a CAD Assembly Model

In this section...
“Before You Begin” on page 6-15
“Example Files” on page 6-15
“Import a Model” on page 6-15
“After Model Import” on page 6-16

Before You Begin

You import a CAD model into Simscape Multibody software using the `smimport` function. The function parses an XML multibody description file and automatically generates the corresponding model. Geometry files accompanying the multibody description file format provide the body geometries for visualization purposes. You can import models of assemblies but not parts.

Example Files

You can try the CAD import workflow using the examples in your Simscape Multibody installation. The examples include a four-bar linkage, a robotic arm, and a Stewart platform. Each example comprises an XML multibody description file and a set of part geometry files. The files are in folders with path

matlabroot`\toolbox\physmod\sm\smdemos\import\modelFolder`,

where:

- *matlabroot* is the root folder of your MATLAB application, for example:

  `C:\Programs\MATLAB\`
- *modelFolder* is the name of the folder that contains the example file sets—`four_bar`, `robot`, or `stewart_platform`.

Import a Model

You import a model into Simscape Multibody software using the `smimport` function in its default mode. Consider the example file sets in your Simscape Multibody installation.

To recreate the CAD assembly model described by the files as Simscape Multibody block diagrams, enter:

```
smimport(multibodyDescriptionFile);
```

where *multibodyDescriptionFile* is the XML multibody description file name for the example model you want to import, specified as a string. Use `sm_robot` for the robotic arm model and `stewart_platform` for the Stewart platform model. For example, to import the robotic arm model, enter:

```
smimport('sm_robot');
```

The function generates a new Simscape Multibody block diagram and a supporting data file. The block diagram recreates the original CAD assembly model using Simscape Multibody blocks. The data file provides the numerical values of the block parameters used in the model.

After Model Import

Check the imported model for unexpected rigid connections between bodies. Simscape Multibody software replaces unsupported CAD constraints with rigid connections that may appear as direct frame connection lines, Rigid Transform blocks, or Weld Joint blocks.

A warning message in the MATLAB command window identifies the bodies and connection frames affected by the unsupported constraints. Replace the artificial rigid connections between the bodies with suitable Joint, Constraint, or Gear blocks from the Simscape Multibody library.

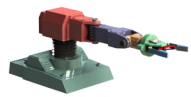
Update the block diagram to rule out model assembly errors. Run simulation to ensure the model dynamics are as expected. If you update the source CAD assembly model, you can generate an updated data file directly from a new multibody description file.

Import a Robotic Arm CAD Model

In this section...
“Example Overview” on page 6-17 “Example Files” on page 6-17 “Import the Model” on page 6-17 “Visualize the Model” on page 6-18 “Build on the Model” on page 6-20

Example Overview

This example shows how to generate a Simscape Multibody model from a multibody description XML file using the `smimport` function. The example is based on a multibody description file named sm_robot and a set of part geometry files included in your Simscape Multibody installation. These files describe the robotic arm model shown in the figure.



Example Files

The multibody description and part geometry files used in this example are located in the folder

```
matlabroot\toolbox\physmod\sm\smdemos\import\robot
```

where *matlabroot* is the root folder of your MATLAB installation, for example:

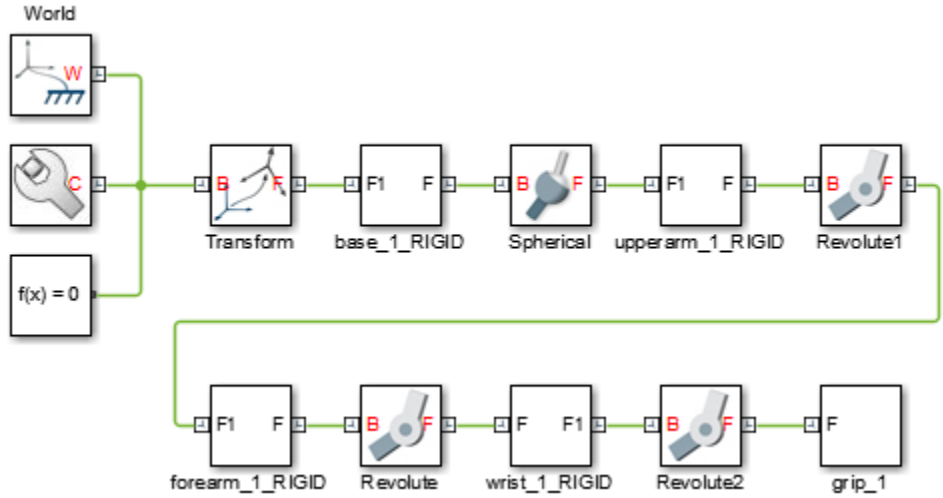
```
C:Programs\MATLAB\
```

Import the Model

At the MATLAB command prompt, enter the command:

```
smimport('sm_robot');
```

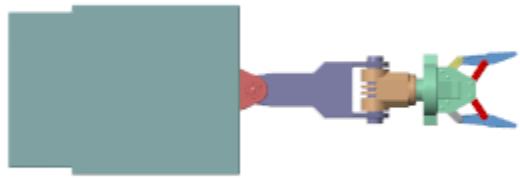
Simscape Multibody software generates the model described in the sm_robot.xml file using the default `smimport` function settings.



The blocks in the generated model are parameterized in terms of MATLAB variables. The numerical values of these variables are defined in a data file that is named sm_robot.m and stored in the same active folder as the generated model.

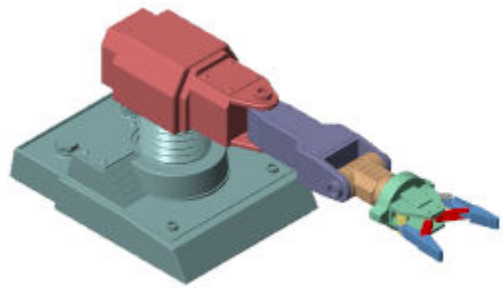
Visualize the Model

Update the diagram to visualize the model. You can do this from the Simulink menu bar by selecting **Simulation > Update Diagram**. Mechanics Explorer opens with a static visualization of the robotic arm model in its initial configuration.



The default view convention in Mechanics Explorer differs from that in the CAD application used to create the original assembly model. Mechanics Explorer uses a Z-axis-up view convention while the CAD application uses a Y-axis-up view convention.

Change the view convention from the Mechanics Explorer toolstrip by setting the **View convention** parameter to `Y up (XY Front)`. Then, select a standard view from the **View > Standard Views** menu to apply the new view convention.



Build on the Model

Try to simulate the model. Because the robotic arm lacks a control system, it simply flails under gravity. You can use Simulink blocks to create the control system needed to guide the robotic arm motion. A control system would convert motion sensing outputs into actuation inputs at the various joints. You can expose the sensing and actuation ports from the joint block dialog boxes.

See Also

`smimport`

Related Examples

- “Import a CAD Assembly Model” on page 6-15

Onshape Import

In this section...
“What Is Onshape?” on page 6-21
“What’s in an Onshape Model?” on page 6-21
“Preparing a Model for Import” on page 6-22
“Importing an Onshape Model” on page 6-22
“What Can You Import?” on page 6-24
“User Authentication and Account Permissions” on page 6-24
“Mapping to Simscape Multibody Blocks” on page 6-25
“Onshape Import Warnings and Errors” on page 6-25
“Physical Units” on page 6-26
“Obtaining Onshape Models to Import” on page 6-26

What Is Onshape?

Onshape is a third-party CAD application that you can import multibody models from. As with other CAD applications, you use Onshape software to model 3-D parts and articulated assemblies. Onshape is full-cloud software and does not rely on a local installation to run.

You must have an active Onshape account to use the software. The Simscape Multibody `smexportonshape` function replaces the Simscape Multibody Link plug-in as the CAD export means. The plug-in is incompatible with Onshape and cannot be used with Onshape models.

What’s in an Onshape Model?

Onshape models are similar in composition to other multibody models. Parts connect through mates such as Ball, Slider, and Revolute to form articulated linkages, mechanisms, and machines. The models are hierarchical, with parts and mates nested inside subassemblies that can in turn be nested inside larger subassemblies.

Each Onshape model exists in a cloud document. A document can have multiple Part Studio tabs, for modeling parts, and Assembly tabs, for mating parts. A Part Studio tab

can have multiple parts and these can be modeled in the relative poses anticipated in the final assembly—for example, to form rigid groups during assembly without the aid of Fixed mates.

Rigid groups are unique to Onshape models and map into Simulink Subsystem blocks with rigidly connected bodies enclosed.

A Word about Terminology

Onshape and Simscape Multibody models have different standard terms for what are often the same things. Parts in an Onshape model are bodies in a Simscape Multibody model. Mates and relations in an Onshape model are joints and constraints in a Simscape Multibody model. These terms are used interchangeably here.

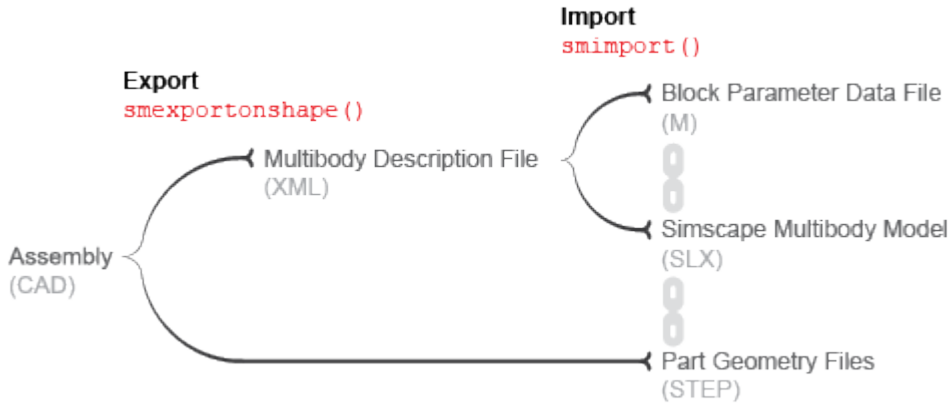
Preparing a Model for Import

Consider fixing one part in each Onshape subassembly. The fixed part determines the location of the subassembly reference frame in the imported model. You must fix the reference part directly in the subassembly tab containing that part. Parts fixed in the root assembly tab—that containing all other subassemblies or bodies—have no impact on the location of the subassembly reference frames.

Ensure that your Onshape parts are free of geometry errors and that your mates have been fully defined. If no mate exists between two parts—that is, if they have six relative degrees of freedom—the imported model will show a 6-DOF Joint block between the corresponding body subsystems. Fix at least one part at the root assembly level to prevent such a block from being added between the assembly and the World frame.

Importing an Onshape Model

You import an Onshape model into the Simscape Multibody environment using the `smexportonshape` and `smimport` functions. The `smexportonshape` function converts the Onshape model into an intermediate representation comprising an XML file and a set of STEP files. The `smimport` function converts the XML file into the final Simscape Multibody model and a supporting data file.



Onshape CAD Import Workflow

About the Intermediate Files

The XML file provides the `smimport` function the data it needs in order to recreate the Onshape model in the Simscape Multibody environment. This file is referred to as the *multibody description* file and is required for model import.

The STEP files provide the imported model, once generated, the 3-D geometries it needs in order to render the bodies in the visualization panes of Solid blocks and Mechanics Explorer. The files are referred to as *geometry files* and are optional for model import.

About the Imported Model

The geometry files are referenced in the Solid blocks of the imported model. If the geometry files are missing, or if the paths to the files change, the body geometries no longer show in the visualization panes of the Solid blocks and Mechanics Explorer. Simulation is unaffected provided that the model is otherwise still valid.

The remaining block parameters are specified in terms of MATLAB variables defined in the supporting data file generated by the `smimport` function. The variables are stored in a single data structure, with the data field names and indices identifying the block parameters that the variables correspond to—e.g., `smiData.Solid(2).mass`.

What Can You Import?

You can use the `smexportonshape` and `smimport` functions to import the contents of Onshape Assembly tabs only. Part Studio tabs lack the necessary information to generate a complete XML multibody description file and cause the `smexportonshape` function to error. Other Onshape document tabs, such as Drawing and Folder, present the same problem and cannot be exported.

If You Need to Import Single Parts

You can export part geometries directly from an Onshape document using the Onshape Export feature. This feature enables you to save geometries in various formats, but only two are compatible with Simscape Multibody models—STL and STEP. Once exported, the geometries can be individually imported into Simscape Multibody Solid blocks. See "Imported Solid Shapes" on page 1-46.

User Authentication and Account Permissions

Before you can export an Onshape model, the `smexportonshape` function needs to authenticate your Onshape account and verify that the Simscape Multibody Exporter app has access permission to your Onshape models.

Authentication occurs once per MATLAB session and is based on a protocol known as OAuth2. So that you can validate your account, the `smexportonshape` function automatically opens an Onshape log-in page on your first export attempt of a session.

You need give the Simscape Multibody Exporter app access permissions only once until these are revoked. So that you can set the permissions for the app, the `smexportonshape` function automatically opens an Onshape application authorization page on your first-ever export attempt.

You can revoke permissions at any time from the Onshape Applications page.

About the OAuth2 Protocol

OAuth2 is an authentication protocol that delegates the authentication process to the service hosting the account—in this case, Onshape. The oauth.net website describes the protocol as a valet key that gives third-party applications such as Simscape Multibody access to some, but not all, aspects of your account. In particular, Simscape Multibody can access your app permissions data, but it cannot see or store your private log-in credentials.

Mapping to Simscape Multibody Blocks

Simscape Multibody can map all Onshape mates but Tangent to equivalent blocks. The mappings are straightforward, with only slight differences between mate and block names standing out. The table shows the mappings used during model import.

Onshape Mate	Simscape Multibody Block or Feature
Ball	Spherical Joint
Cylindrical	Cylindrical Joint
Fastened	Direct connection line on page 1-26
Parallel	Angle Constraint
Pin Slot	Pin Slot Joint
Planar	Planar Joint
Revolute	Revolute Joint
Slider	Prismatic Joint

Onshape relations such as Gear and Linear are not supported. All relations in your model are ignored during import. You can often model the ignored relations using Simscape Multibody blocks—for example, using the Common Gear Constraint block to model a Gear relation between two gears. You may need to create and carefully place new frames before adding such blocks.

Onshape Import Warnings and Errors

Invalid Assembly URLs

The `smexportonshape` function expects the URL of an Onshape Assembly tab as an argument. An Onshape document often comprises Part Studio, Drawing, and other tabs. If you inadvertently specify the URL for the wrong document tab—one not containing an assembly—the function throws an error.

Zero-Mass Bodies

Onshape parts without assigned material translate into Simscape Multibody bodies with zero inertia. Such massless bodies can cause simulation to fail due to degenerate mass errors. A MATLAB warning identifies all massless bodies identified in your model, if any. You can manually specify the mass of a massless body after import using the Solid block for that body. However, as a best practice, always try to assign a material to each body in an assembly before exporting it.

Unsupported Mates and Relations

Onshape mates such as Tangent and relations such as Gear are not supported. The `smexportonshape` function throws a warning identifying all unsupported mates and relations, if any. Unsupported mates map into nothing in a Simscape Multibody model. For a list of Onshape mates that you can import, see “Mapping to Simscape Multibody Blocks” on page 6-25.

Physical Units

The block parameters in the imported model are in the default units of the Onshape model workspace. These units often include a mix derived from SI, CGS, and other unit systems. You can change the units for an entire model in your Onshape model workspace and for an individual block in your imported Simscape Multibody model.

Obtaining Onshape Models to Import

You can create a free account and create your own assembly model in—or import one into —your Onshape account. Many Onshape assembly models are public. You can import all such models into the Simscape Multibody environment.

See Also

`smexportonshape` | `smimport`

More About

- “Import an Onshape Humanoid Model” on page 6-27
- “CAD Translation” on page 6-2
- “URDF Import” on page 6-41

Import an Onshape Humanoid Model

In this section...
“Onshape Import” on page 6-27
“Example Overview” on page 6-27
“Export the Model” on page 6-28
“Import the Model” on page 6-28

Onshape Import

You can import a CAD assembly model from Onshape software into the Simscape Multibody environment. The import process occurs in two steps based on the `smexportonshape` and `smimport` functions. The `smexportonshape` exports the assembly model in an intermediate XML conforming to the Simscape Multibody XML schema. The `smimport` function converts the intermediate XML file into a Simscape Multibody version of the original Onshape model.

Example Overview

This example shows how to import an Onshape model of a humanoid robot assembly. The model comprises various parts representing the torso, head, and limbs of the robot. The parts connect through `Revolute` mates that represent the various joints. The model is identical to that shown in “Import a URDF Humanoid Model” on page 6-50. Enter the following URL in your web browser to access the model (Onshape login required):

```
https://cad.onshape.com/documents/5817806f96eae5105bfa5085/w/15ab3bfb58cacbf427d77ff3/e
```

Model Schematic

Export the Model

Use the `smexportonshape` function to export the model:

1. At the MATLAB command prompt, navigate to a folder for which you have write privileges—for example:

   ```
   cd C:\Users\JDoe\Documents\Models
   ```

2. Store the model URL in a MATLAB variable named `assemblyURL`:

   ```
   assemblyURL = 'https://cad.onshape.com/documents/5817806f96eae5105bfa5085/w/15ab3bf
   ```

3. Export the model and save the XML file name in a variable named `exportedModel`:

   ```
   exportedModel = smexportonshape(assemblyURL);
   ```

   You may be prompted to log in to your Onshape account. The `smexportonshape` function generates the XML multibody description file for this model and a set of STEP files for the various part geometries.

Import the Model

Use the `smimport` function to import the XML multibody description file:

```
smimport(exportedModel);
```

The function generates a Simscape Multibody model of the humanoid robot.

Build on the model, for example, by adding control systems to actuate the various joints. For a controlled example, at the MATLAB command prompt enter `sm_import_humanoid_urdf`. Simulate the model to view a simple animation.

See Also

`smexportonshape` | `smimport`

More About

- “Onshape Import” on page 6-21
- “CAD Translation” on page 6-2
- “URDF Import” on page 6-41

URDF Primer

In this section...
“What is URDF?” on page 6-31
“What’s in a URDF File?” on page 6-32
“XML Hierarchies and Kinematic Trees” on page 6-33
“Required and Optional URDF Entities” on page 6-36
“Create a Simple URDF Model” on page 6-37
“Obtaining URDF Models to Import” on page 6-40

What is URDF?

URDF, or Unified Robotics Description Format, is an XML specification used in academia and industry to model multibody systems such as robotic manipulator arms for manufacturing assembly lines and animatronic robots for amusement parks. URDF is especially popular with users of ROS, or Robotics Operating System—a framework that offers standard support for URDF models. You can import URDF models into the Simscape Multibody environment, for example, for simulation, analysis, and control design tasks. See the Humanoid Robot featured example for a simple use case.

Humanoid Robot URDF Model

What's in a URDF File?

Like other types of XML files, URDF files comprise various XML elements, such as <robot>, <link>, <joint>, nested in hierarchical structures known as XML trees. For example, the <link> and <joint> elements are said to be children of the <robot> element and, reciprocally, the <robot> element the parent of the <link> and <joint> elements.

```
<robot>
    <link>
        ...
    </link>
    <link>
        ...
    </link>
    <joint>
        ...
    </joint>
</robot>
```

Child elements, such as <link> and <joint> under <robot>, can in turn have child elements of their own. For example, the <link> element has the child elements <inertial> and <visual>. The <visual> element has the child elements <geometry> and <material>. And the <material> element has the child element <color>. Such chains of child elements are essential to define the properties and behavior of the parent elements.

```
<robot>
    <link>
        <inertial>
            ...
        </inertial>
        <visual>
            <geometry>
                ...
            </geometry>
            <material>
                <color />
            </material>
        </visual>
    </link>
    ...
</robot>
```

In addition to child elements, the XML elements in a URDF model can have attributes. For example, the `<robot>`, `<link>`, and `<joint>` elements all have the attribute `<name>`—a string that serves to identify the element. The `<color>` element has the attribute `rgba`—a numeric array with the red, green, blue, and alpha (or opacity) values of the link color. Attributes such as these help to completely define the elements in the model.

```
<robot name = "linkage">
    <link name = "root link">
        <inertial>
            ...
        </inertial>
        <visual>
            <geometry>
                ...
            </geometry>
            <material>
                <color rgba = "1 0 0 1" />
            </material>
        </visual>
    </link>
    ...
</robot>
```

XML Hierarchies and Kinematic Trees

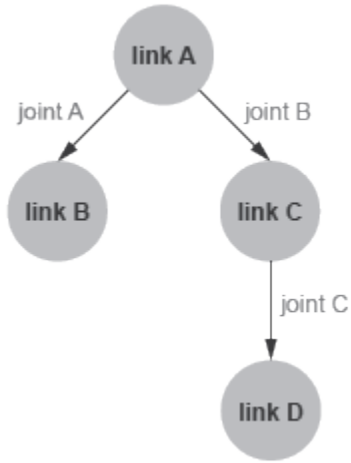
URDF links connect through joints in hierarchical structures not unlike those formed by nesting XML elements in a URDF file. `<joint>` elements enforce these hierarchies through `<parent>` and `<child>` elements that identify one link as the parent and the other as the child. Parent links can themselves be children—and child links parents—of other links in the model.

```
<robot name = "linkage">
    <joint name = "joint A ... >
        <parent link = "link A" />
        <child link = "link B" />
    </joint>
    <joint name = "joint B ... >
        <parent link = "link A" />
        <child link = "link C" />
    </joint>
    <joint name = "joint C ... >
        <parent link = "link C" />
```

```
        <child link = "link D" />
    </joint>
</robot>
```

<parent> and <child> Joint Elements

You can visualize the connections between links using a schematic known as a connectivity graph. The figure shows an example. Circles denote links and arrows denote joints. The arrow roots identify the parent nodes and the arrow tips the child nodes. The connectivity graph reveals the topology of the underlying model—here a simple kinematic tree with two branches.



Connectivity Graph of a Kinematic Tree

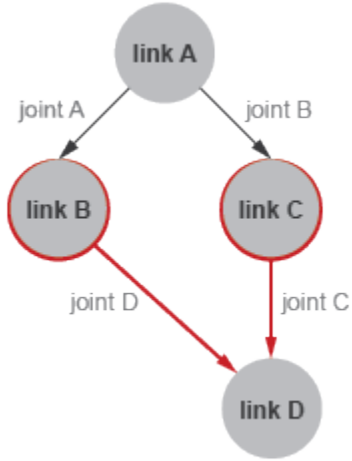
Model topology is important in URDF. The connectivity graph of a model can take the shape only of a kinematic tree—a kinematic chain, branched or unbranched, that is always open. Kinematic loops, each a closed chain formed by joining the ends of an otherwise open chain, are disallowed. This restriction impacts how <link> elements can connect in a URDF model.

The restriction translates to the following rule: no <link> element can serve as a child node in more than one <joint> element. Put another way, no <link> element can have more than one parent element in the model's connectivity graph. Only the root link, that at the origin of the connectivity graph, can have a number of parent nodes different from one (zero). Only one root link is allowed in a model.

```
<robot name = "linkage">
    <joint name = "joint A ... >
        <parent link = "link A" />
        <child link = "link B" />
    </joint>
    <joint name = "joint B ... >
        <parent link = "link A" />
        <child link = "link C" />
    </joint>
    <joint name = "joint C ... >
        <parent link = "link C" />
        <child link = "link D" />
    </joint>
    <joint name = "joint D ... >
        <parent link = "link B" />
        <child link = "link D" />
    </joint>
</robot>
```

Kinematic Loop URDF Example

The code declares a link, `link D`, as a child node in two `<joint>` elements, `joint C` and `joint D`. The `link D` element has two parents and forms a kinematic loop. The model violates the URDF connection rules and is invalid. The figure shows the connectivity graph of the model.



Connectivity Graph of a Kinematic Loop

Required and Optional URDF Entities

Not all elements and attributes listed in the URDF specification are required. Some, like <inertial> under <link>, are optional. The following code shows the various elements and attributes that you can use, with those that are optional colored green.

Elements and attributes shown as required inside optional elements are so only if the optional elements are used. The default values of optional attributes are shown in parentheses and in italic font. Note that this code is included only as a reference and that it does not represent a valid URDF model. Ellipses ("...") are invalid in URDF models and are used merely to break long lines of code for ease of viewing.

```
<robot name>
    <link name>
        <inertial>
            <origin xyz("0 0 0") rpy("0 0 0") />
            <mass value />
            <inertia ixx iyy izz ixy ixz iyz />
        </inertial>
        <visual name>
            <origin xyz("0 0 0") rpy("0 0 0") />
            <geometry>
                <box size />
                <cylinder radius length />
                <sphere radius />
                <mesh filename scale("1") />
            </geometry>
            <material name>
                <color rgba("0.5 0.5 0.5 1") />
                <texture filename />
            </material>
        </visual>
        <collision name>
            <origin xyz("0 0 0") rpy("0 0 0") />
            <geometry>
                <box size />
                <cylinder radius length />
                <sphere radius />
                <mesh filename scale("1") />
            </geometry>
        </collision>
    </link>
    <joint name type>
        <origin xyz("0 0 0") rpy("0 0 0") />
```

```
        <parent link />
        <child link />
        <axis xyz("1 0 0") />
        <calibration rising />
        <calibration falling />
        <dynamics damping("0") friction("0") />
        <limit† lower† upper† effort velocity />
        <mimic joint multiplier("1") offset("0") />
        <safety_controller soft_lower_limit("0") ...
        ... soft_upper_limit("0") k_position("0") k_velocity("0") />
    </joint>
</robot>
```

†Required for <joint> elements of type prismatic and revolute only.

Create a Simple URDF Model

As an example, create a URDF model of a double pendulum. In your text editor of choice, create a file with the code shown below and save the file as double_pendulum.urdf in a convenient folder. Include the file extension in the file name. A separate example shows how to import this model into the Simscape Multibody environment (see "Import a Simple URDF Model" on page 6-44).

```
<robot name = "linkage">
    <!-- links section -->>
    <link name = "link A">
        <inertial>
            <origin xyz = "0 0 0" />
            <mass value = "0.5" />
            <inertia ixx = "0.5" iyy = "0.5" izz = "0.5"
ixy = "0" ixz = "0" iyz = "0" />
        </inertial>
        <visual>
            <origin xyz = "0 0 0" />
            <geometry>
                <box size = "0.5 0.5 0.1" />
            </geometry>
            <material name = "gray A">
                <color rgba = "0.1 0.1 0.1 1" />
            </material>
        </visual>
    </link>
    <link name = "link B">
```

```
    <inertial>
        <origin xyz = "0 0 -0.5" />
        <mass value = "0.5" />
        <inertia ixx = "0.5" iyy = "0.5" izz = "0.5"
ixy = "0" ixz = "0" iyz = "0" />
    </inertial>
    <visual>
        <origin xyz = "0 0 -0.5" />
        <geometry>
            <cylinder radius = "0.05" length = "1"  />
        </geometry>
        <material name = "gray B">
            <color rgba = "0.3 0.3 0.3 1" />
        </material>
    </visual>
</link>
<link name = "link C">
    <inertial>
        <origin xyz = "0 0 -0.5" />
        <mass value = "0.5" />
        <inertia ixx = "0.5" iyy = "0.5" izz = "0.5"
ixy = "0" ixz = "0" iyz = "0" />
    </inertial>
    <visual>
        <origin xyz = "0 0 -0.5" />
        <geometry>
            <cylinder radius = "0.05" length = "1"  />
        </geometry>
        <material name = "gray C">
            <color rgba = "0.5 0.5 0.5 1" />
        </material>
    </visual>
</link>

<!-- joints section -->>
<joint name = "joint A" type = "continuous">
    <parent link = "link A" />
    <child link = "link B" />
    <origin xyz = "0 0 -0.05" />
    <axis xyz = "0 1 0" />
</joint>
<joint name = "joint B" type = "continuous">
    <parent link = "link B" />
    <child link = "link C" />
```

```
        <origin xyz = "0 0 -1" />
        <axis xyz = "0 1 0" />
        <dynamics damping ="0.002" />
    </joint>
</robot>
```

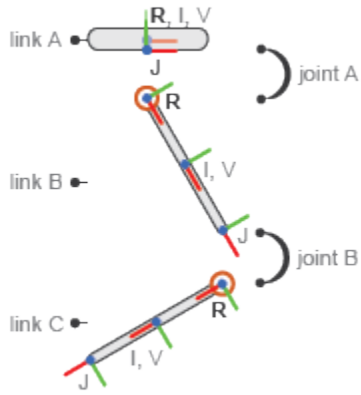
About the URDF Model

The code defines a multibody model named `linkage`. The model contains three links, named `link A`, `link B` and `link C`, that connect via two joints, named `joint A` and `joint B`. The `<parent>` and `<child>` elements of the joints identify how the links connect to each other: `link A` connects to `link B` and `link B` connects to `link C`. `link A` has no parent link—that is, it appears in `<joint>` elements as a child element only—and is therefore the root link.

The `<inertial>` element of `link A` defines the mass and moments of inertia (`ixx`, `iyy`, `izz`) of the link. The products of inertia (`ixy`, `ixz`, and `iyz`) are unspecified and have the URDF default value of zero. The `visual` element of `link A` defines the geometry type and material color for use in the model visualization. The geometry in this case is a box with width and thickness of `0.5` m and height of `0.1` m. The `<origin>` elements of the link `<inertial>` and `<visual>` specify the transforms from the link reference frame to the inertial and visual reference frames. Similar elements apply to `link B` and `link C`.

The `type` attribute of the `<joint>` elements defines the joints as continuous—a type of revolute joint without motion limits. The `<origin>` element specifies the location of the joint relative to the reference frame of the parent link element. For example, the `<origin>` element of `joint A` offsets the joint `0.05` m along the `-Z` axis relative to the origin of the `link A` reference frame. The `axis` element nested inside each `joint` element defines the rotational axis of the joint as the Cartesian vector `[0, 1, 0]`, or `+Y`.

The figure shows the components of the model—the links and joints—and the various frames they contain. **R** denotes a link reference frame, I a link inertial frame, and V a link visual frame. J denotes a joint reference frame—by definition held coincident with the reference frame of the child link. The inertial and visual frames are offset to the centers of the links and the joint frames to their lower edges.



Double-Pendulum Model Components

Obtaining URDF Models to Import

You can, but generally do not have to, manually create your own URDF files. For more complex models, it can be preferable to obtain URDF files from other sources. Robotics manufacturers and consultants often provide URDF models for their robotic systems. CAD applications such as SolidWorks and PTC Creo support URDF exporters that convert your CAD assemblies into URDF models. Consider these options when working with complex robotics models that may not be simple to create manually.

See Also

`smimport`

More About

- “URDF Import” on page 6-41
- “CAD Translation” on page 6-2

URDF Import

In this section...
“Importing a URDF Model” on page 6-41
“URDF Entities That You Can Import” on page 6-41
“Mapping to Simscape Multibody Blocks” on page 6-42
“Import a Simple URDF Model” on page 6-44
“URDF Import Limitations” on page 6-47
“Mesh Geometries” on page 6-48
“Physical Units” on page 6-48
“Differences from CAD Import” on page 6-48

Importing a URDF Model

You import a URDF model into the Simscape Multibody environment using the `smimport` function with the URDF file name as its principal argument. The function identifies the model type—URDF or XML—via the file extension. If you omit the extension, the function assumes that the file is in the XML format more commonly used for CAD import. For example, the command

```
smimport('sm_humanoid.urdf')
```

tells the function to import a URDF model named `sm_humanoid`. The command

```
smimport('sm_humanoid')
```

tells the function to import a multibody model from an intermediate XML file named `sm_humanoid`. If the function finds no XML file with the specified name, it returns an error—even if there is a URDF file with the same name in the same folder. As a rule, you must include the URDF file extension explicitly whenever attempting to import URDF models.

URDF Entities That You Can Import

Simscape Multibody software supports only a subset of all the available URDF elements and attributes. You can import a URDF model with unsupported elements or attributes, but all such entities are ignored. The following code shows the elements and attributes

that you can and cannot import. Elements are shown in bold font and attributes in regular font. Unsupported elements and attributes are shown in red.

```
<robot name>
    <link name>
        <inertial>
            <origin xyz rpy />
            <mass value />
            <inertia ixx iyy izz ixy ixz iyz />
        </inertial>
        <visual name>
            <origin xyz rpy />
            <geometry>
                <box size />
                <cylinder radius length />
                <sphere radius />
                <mesh filename scale />
            </geometry>
            <material name>
                <color rgba />
                <texture filename />
            </material>
        </visual>
        <collision name>
            <origin xyz rpy />
            <geometry>
                <box size />
                <cylinder radius length />
                <sphere radius />
                <mesh filename scale />
            </geometry>
        </collision>
    </link>
    <joint name type>
        <origin xyz rpy />
        <parent link />
        <child link />
        <axis xyz />
        <calibration rising />
        <calibration falling />
        <dynamics damping friction />
        <limit lower upper effort velocity />
        <mimic joint multiplier offset />
        <safety_controller soft_lower_limit ...
        ... soft_upper_limit k_position k_velocity />
    </joint>
</robot>
```

Mapping to Simscape Multibody Blocks

The URDF `<robot>` element maps into a Simscape Multibody model. The `<link>` elements nested inside the `<robot>` element map into Simulink Subsystem blocks representing the links or, in Simscape Multibody nomenclature, bodies. The `<joint>` elements map into equivalent Simscape Multibody joint blocks. The `name` attributes of these elements map into the model name, the Subsystem block names, and the joint block names, respectively.

The Subsystem blocks comprise Solid, Rigid Transform, and Reference Frame blocks. The Solid blocks provide the solid properties—geometry, inertia, color—of the body. Each Subsystem block contains two types of Solid blocks. One is named `Inertial` and contains the inertia data from the `<inertial>` element. `Inertial` Solid block impact only model dynamics. The other is named `Visual` and contains the geometry and color data from the `<visual>` element. `Visual` Solid blocks impact only model visualization.

The Rigid Transform blocks provide the translational and rotational offsets from the local reference frame of the body to the Inertial and Visual elements. These transforms are derived from the `<origin>` elements of the `<inertial>` and `<visual>` elements of links, as well as from the `<origin>` and `<axis>` elements of joints. The Reference Frame block identifies the local reference frame of the body.

The type of joint block used depends on the `<type>` attribute of the `<joint>` element. The joint mapping between URDF and Simscape Multibody software is largely intuitive. A `<joint>` element of `type` prismatic maps into a Prismatic Joint block. A `<joint>` element of `type` fixed maps into a Weld Joint block. The table shows the mappings for the remaining URDF `<joint>` elements.

Correspondence Between URDF and Simscape Multibody Joints

URDF <joint type> Attribute	Simscape Multibody Joint Block	Degrees of Freedom
`revolute`	Revolute Joint†	One rotational with joint limits†
`continuous`	Revolute Joint	One rotational (without joint limits)
`prismatic`	Prismatic Joint	One translational with joint limits†
`fixed`	Weld Joint	Zero (rigid connection)
`floating`	6-DOF Joint	Three rotational and three translational
`planar`	Planar Joint	Two rotational and one translational

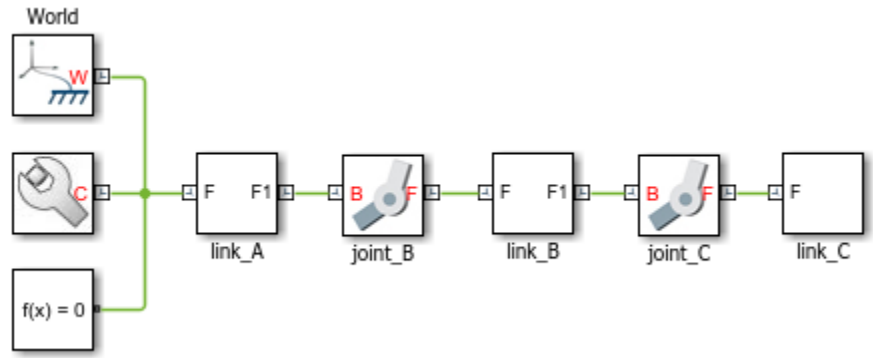
†Joint limits are ignored

Import a Simple URDF Model

As an example, import the double-pendulum URDF model described in the "Create a Simple URDF Model" on page 6-37 section. Create the URDF model if you have not yet done so before proceeding. To import the model, navigate to the folder in which you saved your double-pendulum URDF model. Then, at the MATLAB command prompt, enter the command

```
smimport('double_pendulum.urdf')
```

If you saved your URDF model under a different name, use that name instead. The function imports the URDF model and generates an equivalent Simscape Multibody model. The figure shows the resulting model with the blocks and their connection lines slightly rearranged.



About the Imported Model

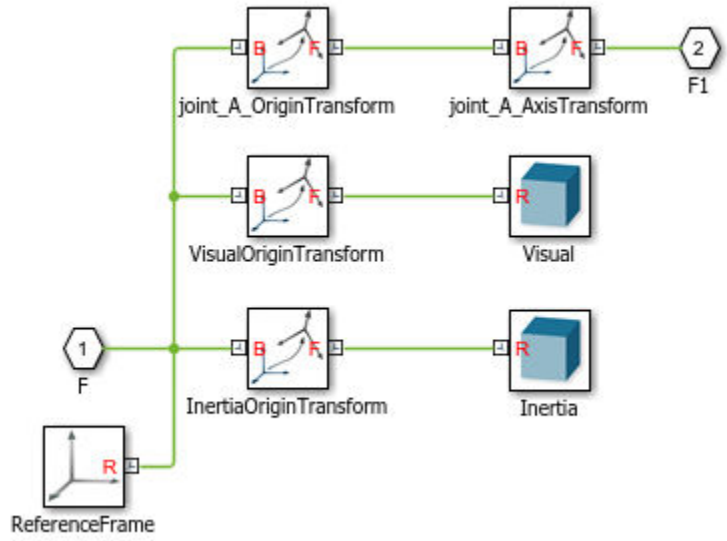
The `<link>` elements named `link A`, `link B`, and `link C` in the URDF model map into Simulink Subsystem blocks also named `link A`, `link B`, and `link C`. The `<joint>` elements named `joint A` and `joint B`—each with `type` set to `continuous`—map into Simscape Multibody Revolute Joint blocks also named `joint A` and `joint B`.

The block diagram reflects the topology of the URDF model—an unbranched kinematic tree. `link C` connects to `link B` as a child of that element. `link B` in turn connects to `link A` as a child of that element. `link A` is the root link and is therefore grounded—a condition reflected in the rigid connection between the `link A` and World Frame blocks.

URDF Model Topology

The Subsystem blocks representing the URDF `<link>` elements each comprise a small block diagram with one Reference Frame block, two Solid blocks, and a variable number of Rigid Transform blocks. The Subsystem blocks are not masked and can be opened directly with a double click. The figure shows the block diagram of the `link A` Subsystem block.

One Solid block is the translated equivalent of the `<visual>` URDF element and is named Visual. This block contains the relevant parameters of the `<visual>` element, including link geometry and color. The second Solid block is the translated equivalent of the `<inertial>` URDF element and is named Inertial. This block contains the relevant parameters of the `<inertial>` element, including link mass, moments of inertia, and products of inertia.



`link A` Subsystem

The Reference Frame block identifies the local reference frame of the `<link>` URDF element. This frame coincides with the joint connection frame to the parent link or, as in this case of a root link, to the World Frame block. The Rigid Transform blocks specify the translational and rotational transforms to the reference frames of the `<inertial>`, `<visual>`, and `<joint>` URDF elements. An additional Rigid Transform block specifies the rotation transform needed to align the Simscape Multibody joint axis with the URDF joint axis.

Assemble and Simulate the Imported Model

Build on the model to obtain a meaningful simulation. You can, for example, use joint state targets to assemble the double pendulum in an unstable configuration and simulate its fall under gravity:

1. In the dialog box of the joint_A block, select the **State Targets > Specify Position Target** checkbox and set the **Value** parameter to `30` deg. This parameter sets the starting angle of the upper joint.
2. In the **Solver** pane of the Configuration Parameters window, click **Additional options** and set the **Max step size** parameter to `0.01`. This value keeps the solver

step size small enough to produce a smooth animation during simulation. Increase the value if simulation proceeds slowly.

3 Update the block diagram and run the simulation. You can update the block diagram by selecting **Simulation > Update Diagram**. You can simulate the model by selecting **Simulation > Run**. Mechanics Explorer shows an animation of the double pendulum fall under gravity.

URDF Import Limitations

URDF models allow only tree topologies. Some URDF variants, such as SDF (Simulation Description Format) and DrakeURDF, add support for kinematic loops. You cannot import models in these alternative formats. You can, however, add kinematic loops to an imported model once in the Simscape Multibody environment.

URDF elements and attributes derived from URDF extensions are not supported. Extensions include `<transmission>` elements used to model gear trains and parallel linkages, `<gazebo>` elements used to simulate URDF models in the Gazebo environment, and the less commonly used `<model_state>` and `<sensor>` elements.

Some core URDF elements and attributes are also not supported. Unsupported elements include `<collision>`, used for link collision detection, and `<limit>`, used for joint motion limits. Unsupported attributes include `scale` for mesh file scaling, `friction` for joint internal mechanics, and more on page 6-41. Unsupported URDF elements and attributes are ignored during import.

Mesh Geometries

You can import URDF models with references to external geometry files, but the files must be in a format compatible with Simscape Multibody software. Valid file formats include STL (Standard Tesselation Format) and STEP (Standard for the Exchange of Product data), sometimes referred to as STP.

Collada, or DAE, geometry files, common in URDF models, are not supported in Simscape Multibody software. If you import a model with references to DAE files, the geometries derived from those files are not rendered. The lack of visualization may limit your ability to analyze a model but has no impact on the model dynamics.

While not natively supported in URDF, any STEP file references that you include in your URDF model import without issue. If necessary, you can convert any DAE files referenced in your URDF model to STEP or STL format and update the file references—found in the `<mesh>` elements nested inside the link `<geometry>` elements—to import the model with full visualization.

Physical Units

The block parameters of the imported model are automatically set to the URDF default units, SI. You can modify the units after import, but you must do so on a black-by-block basis. A variety of other units are supported, including those at the core of MKS (Meter-Kilogram-Second), USC (US Customary), and other units systems.

Differences from CAD Import

Despite their similarities, including their mutual reliance on the `smimport` function, CAD and URDF import differ in some important aspects:

- CAD models are imported in an intermediate XML format. URDF models are imported directly in URDF format.

  The intermediate XML files on page 6-2 provide the information needed to recreate the CAD models in the Simscape Multibody environment. The same information is provided directly in URDF files when importing URDF models. XML multibody description files must conform to the Simscape Multibody XML schema. See “Exporting a CAD Model” on page 6-7 for ways to generate a valid XML file.

- Imported CAD models have their numerical parameters defined in MATLAB files. Imported URDF models have their numerical parameters hardcoded into the block dialog boxes.

  CAD import uses a detached data framework that places all block parameter values in a cell structure defined in a separate MATLAB data file on page 6-6. The detached data framework enables you to update a previously imported model on page 6-10 when you modify the source CAD model. URDF import lacks this feature and does not support model update.

See Also

`smimport`

More About

- "URDF Primer" on page 6-31
- "Import a URDF Humanoid Model" on page 6-50
- "CAD Translation" on page 6-2

Import a URDF Humanoid Model

In this section...
“URDF Import” on page 6-50 “Example Overview” on page 6-50 “Import the Model” on page 6-51

URDF Import

You can import a URDF model into the Simscape Multibody environment. The import process occurs in a single step based on the `smimport` function. The `smimport` function converts the URDF model directly into an equivalent Simscape Multibody model.

Example Overview

This example shows how to import an Onshape model of a humanoid robot assembly. The model comprises various parts (”links” in URDF jargon) representing the torso, head, and limbs of the robot. The parts connect through revolute and weld joints (”continuous” and “fixed” respectively). This model is identical to that shown in “Import an Onshape Humanoid Model” on page 6-27. You can open the model from the MATLAB command prompt by entering the command:

```
open sm_humanoid.urdf
```

Model Schematic

Import the Model

Use the `smimport` function to import the URDF model:

```
urdfModel = 'sm_humanoid.urdf';
smimport(urdfModel);
```

The function generates a Simscape Multibody model of the humanoid robot. The file extension is required to identify the import file as URDF. Update the imported model (**Simulation > Update Diagram**) to open a static visualization in the initial state. The figure shows the results.

Build on the model, for example, by adding control systems to actuate the various joints. For a controlled example, at the MATLAB command prompt enter `sm_import_humanoid_urdf`. Simulate the model to view a simple animation.

See Also

`smimport`

More About

- “URDF Primer” on page 6-31
- “URDF Import” on page 6-41
- “CAD Translation” on page 6-2

Deployment

7

Code Generation

Code Generation Applications

In this section...
“Code Generation Overview” on page 7-2 “Simulation Acceleration” on page 7-2 “Model Deployment” on page 7-3

Code Generation Overview

Simscape Multibody supports code generation through Simulink Coder. You can generate C/C++ code from a Simscape Multibody model to accelerate simulation in the Simulink environment or to deploy a model onto external hardware. Model deployment requires an active Simulink Coder license while simulation acceleration does not.

Code Generation Applications

Simulation Acceleration

Simulink can generate C/C++ executable code to shorten simulation time. Two simulation modes rely on code generated from a model:

- Accelerator
- Rapid Accelerator

Simscape Multibody supports the two accelerator modes. You can access the simulation accelerator modes in the Simulink Editor window for your model. Click **Simulation** >

Mode, and select `Accelerator` or `Rapid Accelerator`. Accelerator modes do not require additional Simulink code generation products.

Note Simulation accelerator modes do not support model visualization. When you simulate a Simscape Multibody model in `Accelerator` or `Rapid Accelerator` modes, Mechanics Explorer does not open with a 3-D display of your model.

Model Deployment

With Simulink Coder, you can generate standalone C/C++ code for deployment outside the Simulink environment. The code replicates the source Simscape Multibody model. You can use the stand-alone code for applications that include:

- Hardware-In-Loop (HIL) testing
- Software-In-Loop (SIL) testing
- Rapid prototyping

Note Simscape Multibody supports, but does not perform, code generation for model deployment. Code generation for model deployment requires the Simulink Coder product.

See Also

Related Examples

- “Generate Code for a Multibody Model” on page 7-7
- “Configure Model for Rapid Accelerator Mode” on page 7-9

Code Generation Setup

In this section...
“Before You Begin” on page 7-4
“Solver Selection” on page 7-4
“Target Selection” on page 7-4
“Run-Time Parameters” on page 7-4
“Compiler Optimization” on page 7-5

Before You Begin

Simscape Multibody software supports code generation for fast simulation in the Simulink environment or for model deployment onto external targets. If your goal is to obtain standalone C/C++ code for real-time simulation on an external target, you must have an active Simulink Coder installation.

Solver Selection

Simscape Multibody models have continuous states and require a continuous or hybrid Simulink solver. You can change solvers from the **Solver** pane of the **Model Configuration Parameters** window. Select any solver but that marked `discrete (no continuous states)`. Consider the `ODE1` fixed-step solver if you need to approximate the behavior of a discrete solver.

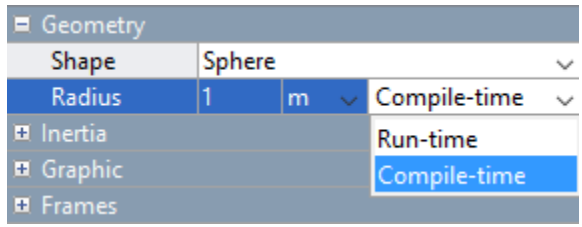
Target Selection

The choice of code generation target depends on the Simulink solver used. If you select a variable-step solver, you must set `rsim.tlc` as the system target file. You can specify the system target file from the **Model Configuration Parameters** window. Look for the **System target file** parameter in the **Target selection** area of the **Code Generation** pane.

Run-Time Parameters

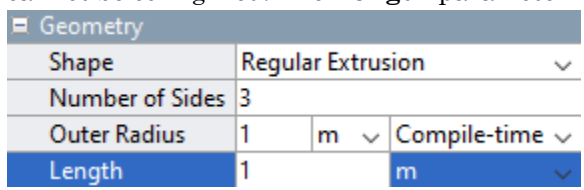
You can configure most numerical block parameters as `Compile-time` (default) or `Run-time` using a drop-down list that appears beside configurable parameters. The figure

shows the run-time drop-down list in a Solid block dialog box. All parameters are by default `Compile-time`. The drop-down list is disabled when the model is in Fast Restart mode.



`Compile-time` parameters update in value when you recompile the model. Leave parameters as `Compile-time` when performing tasks that rely on inlined parameters such as model optimization. `Run-time` parameters update in value without the need for extra compilations. Set parameters to `Run-time` when tuning their values in Fast Restart mode or when simulating models that rely at least in part on generated C code.

To set a parameter as `Run-time` from the block dialog box, you must configure your Simscape preferences. Open the MATLAB Preferences window, select the **Simscape** node, and check the **Show run-time parameter settings** check box. Parameters without a run-time option or with a run-time option that is inactive (i.e., "grayed out") cannot be configured. The **Length** parameter in the figure is an example.



For more information about Simscape run-time parameters, see "About Simscape Run-Time Parameters" (Simscape).

Compiler Optimization

You can set your C/C++ compiler to optimize generated code. Optimized code runs faster but compiles slower. Compilation can be especially slow in large models with many bodies. The choice of compiler can exacerbate the slow compilation times. With certain versions of Microsoft Visual C++, Simulink software may appear to hang as the model is compiled.

If a model takes unusually long to compile, consider switching to a different installed compiler or disabling compiler optimization for your model. The Clang compiler provides a suitable alternative to Microsoft Visual C++. You can perform both tasks from the **Code Generation** menu of the **Model Configuration Parameters** window.

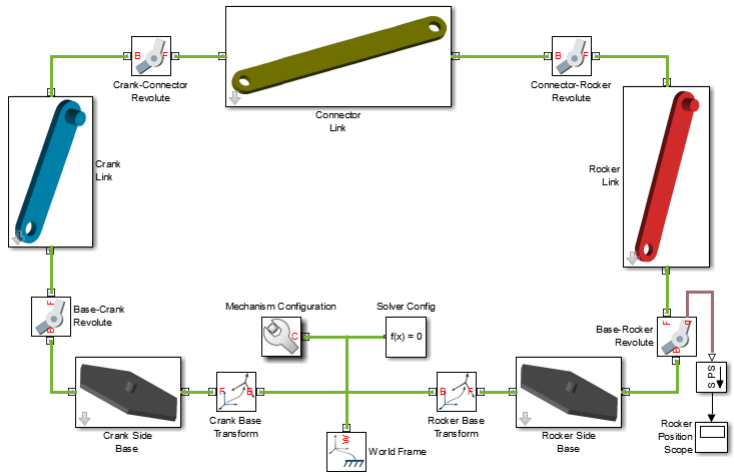
To switch compilers, in the **Toolchain settings** area of the **Code Generation** menu, set the **Toolchain** parameter to a different compiler. To disable code optimization, set the **Build configuration** parameter to `Faster Builds`.

Generate Code for a Multibody Model

This example shows how to configure and generate C code for a simple Simscape Multibody model. The example is based on a four-bar model named `sm_four_bar`. The model uses a variable-step solver, `ode45 (Dormand-Prince)`, and therefore requires the `rsim` target to generate code.

1 At the MATLAB command prompt, enter `sm_four_bar`.

MATLAB software opens the four-bar example model. Save the model with a different name in a convenient folder.



2 In the Simulink menu, select **Simulation** > **Model Configuration Parameters**.

The Model Configuration Parameters window enables you to specify a code generation target and set the code generation report options for your model.

3 In the **Code Generation** node of the Model Configuration Parameters window, set the **System target file** parameter to `rsim.tlc`.

The `rsim.tlc` target file is compatible with Simscape Multibody models that have variable-step solvers.

4 In the **Code Generation** > **Report** node of the Model Configuration Parameters window, check the **Create code generation report** check box and click **OK**.

MATLAB software creates and opens a code generation report when you build your model.

5 In the Simulink menu bar, select **Code > C/C++ Code > Build Model**.

Simulink Coder software generates C code for the four-bar model. The code generation report for your model opens with a list of generated code and data files.

See Also

More About

- "Code Generation Applications" on page 7-2

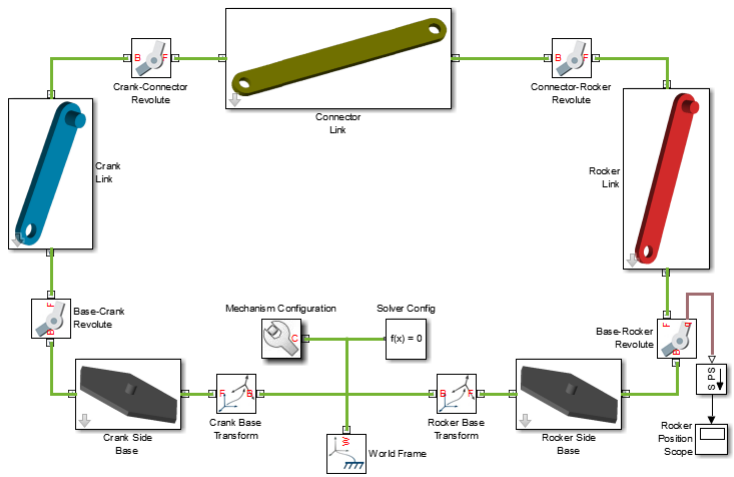
Configure Model for Rapid Accelerator Mode

Model Overview

This example shows how to simulate a simple Simscape Multibody model in Accelerator mode. This simulation mode relies partly on generated C code that you generally must compile only once. An exception arises when modifying `Compile-time` parameters between simulations, as these require recompilation in order to update in value. You can rapidly update parameter values without model recompilation by configuring the parameters as `Run-time`.

1 At the MATLAB command prompt, enter `sm_four_bar`.

MATLAB software opens the four-bar example model. Save the model with a different name in a convenient folder.



2 In the Simulink menu bar, select **Simulation** > **Mode** > **Accelerator**. Simulink converts the model to C code and compiles this code when you next simulate the model.

3 In the Simulink menu bar, select **Simulation** > **Model Configuration Parameters** and in the tree view pane select the **Code Generation** node.

4 In the **Target selection** area, set the **System target file** parameter to `rsim.tlc`. The `rsim.tlc` target file is compatible with Simscape Multibody models that have variable-step solvers.

5 Run the simulation.

See Also

More About

- “Code Generation Applications” on page 7-2