# SimMechanics™

## User's Guide

MATLAB® & SIMULINK®

R2015b

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

*SimMechanics™ User's Guide*

### 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) |

## Multibody Modeling

## 1 Spatial Relationships

## 2 Rigid Bodies

## Multibody Systems

## 3

# 4 Internal Mechanics, Actuation and Sensing

## Simulation and Analysis

## 5 Simulation

## 6 Visualization and Animation

# CAD Import

## 7 About CAD Import

## Deployment

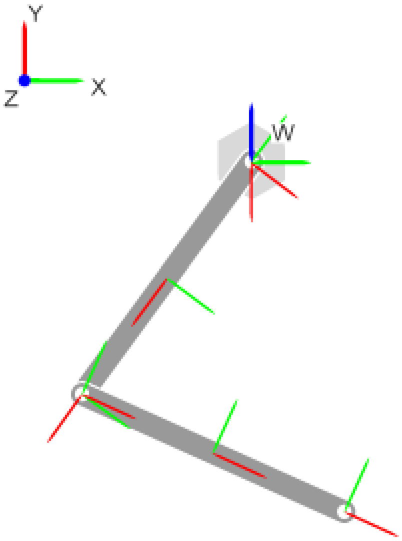
## 8 Code Generation

# Multibody Modeling

# Spatial Relationships

# Working with Frames

Frames form the foundation of multibody modeling. These constructs encode the relative position and orientation of one rigid body with respect to another. In SimMechanics, every rigid body contains at least one frame.

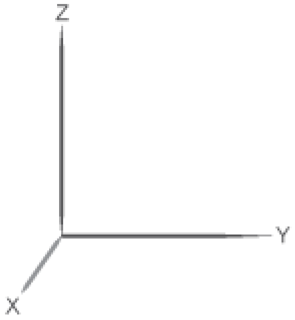


Consider a double pendulum with two links. Each link has a set of physical properties that affect its dynamic behavior and appearance—geometry, inertia, and color. Yet, none of these properties contain information about the spatial arrangement of the links. To position and orient one link with respect to another, you need frames.

You relate two rigid bodies in space by connecting two frames together. In the double pendulum, you connect the end frame of one link to the end frame of another link using a joint. In turn, each link contains a local reference frame against which you define the two end frames. You can make two frames coincident, translate them, or rotate them with respect to each other.

## Frames

Frames have one origin and three axes. The origin defines the local zero coordinate of the frame. This is the point with respect to which you measure the translational distance between two frames. The axes define the directions in which the components of a 3-D vector are resolved. For example, if you measure the translation vector between two frame origins, you can resolve the vector components along the axes of the base frame. For more information, see “Measurement Frames” on page 4-62.



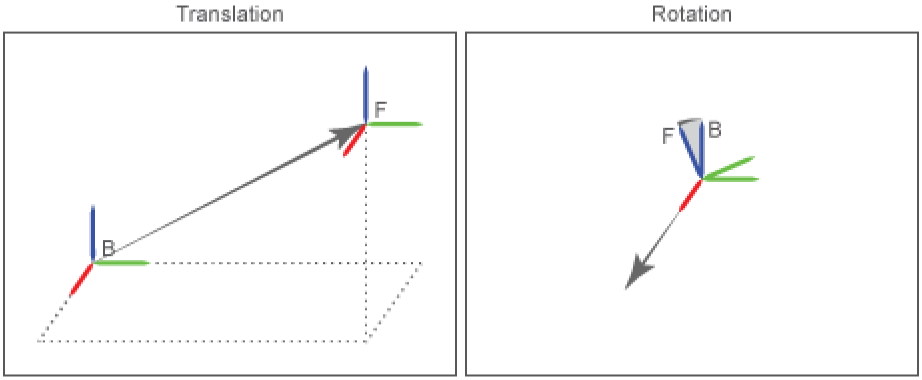
## Frame Types

A multibody model generally contains two frame types: global and local. The global frame represents the world. It is inertial and defines absolute rest in a model. In SimMechanics, you represent the global frame with the `World Frame` block. This block is available in the Frames and Transforms library. The World frame is uniquely defined in every model. You can add multiple World Frame blocks to a model, but they all represent the same frame.

A local frame represents a position and orientation in a rigid body. It can move with respect to the World frame, but not with respect to the rigid body itself. Because it can move with respect to the World frame, a local frame is generally non-inertial. To add a local frame to a rigid body, you use the `Rigid Transform` block. You can add multiple local frames to a rigid body—to define the position and orientation of joints, to apply an external force or torque, or to sense motion. For more information, see "Frame Transformations" on page 1-17.

## Frame Transforms

To separate two frames in space, you apply a frame transformation between them. In SimMechanics, two frame transformations are possible: rotation and translation. Rotation changes the relative orientation of two frames. Translation changes their relative position.



Rigid transformations fix spatial relationships for all time. When you rigidly connect two frames, they move as a single unit. They cannot move with respect to each other. In SimMechanics, you apply a rigid transformation with the `Rigid Transform` block.

---

**Note:** Frame transformations are important in multibody models. The Rigid Transform block is among the most commonly used in SimMechanics.

---

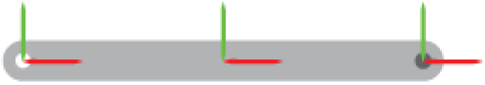
You can also relate to frames with a time-varying transformation. In this case, the rotation, translation, or both, can vary as a function of time. One example is the connection between two links in a double-pendulum. Two frames, one on each link, connect with a joint that allows their spatial relationship to vary with time.

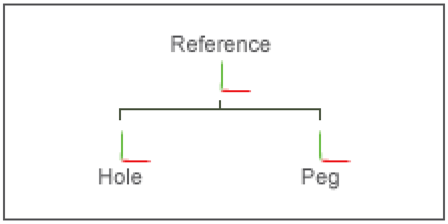
To add a time-varying transformation, you use joint blocks. These blocks allow frame transformations to vary with time. The transformations can arise from model dynamics and joint actuation inputs, the latter of which include force, torque, and motion variables.

## Frame Networks

Rigid body subsystems generally have multiple frames. For example, a binary link—one with two connection points—contains two frames, each identifying a connection point. More complex rigid bodies may have yet more frames. In fact, SimMechanics imposes no limit on the number of frames a rigid body can have. You can add as many frames as your application requires.

The set of frames that belong to a rigid body form a *frame network*. Like other networks, it is often convenient to organize frames hierarchically. You can, for example, organize the frames of a binary link such that its two joint frames are defined with respect to the geometry center frame. In this simple example, the frame network contains two hierarchical levels: a top level containing the geometry center frame, and a lower level containing two joint frames. More complex rigid bodies generally have more hierarchical levels.





The top hierarchical level contains the parent frame. Lower hierarchical levels contain children frames. Children frames can in turn contain their own children frames. All frames in a frame network depend, directly or indirectly, on the parent frame. This dependence exists because the sequence of frame transformations used to define a frame must ultimately reference the parent frame.

## More About

- “Frame Transformations” on page 1-17
- “Representing Frames” on page 1-6
- “Motion Sensing” on page 4-44

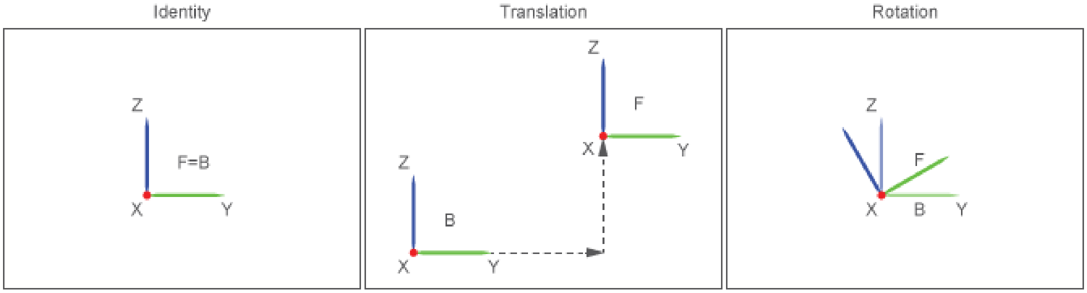
# Representing Frames

**In this section...**

You represent frames with frame ports, lines, and nodes. Each of these frame entities represents one frame. You connect one frame entity to any other using a connection line. When you do so, you apply a spatial relationship between the two frames. Spatial relationships that you can specify include:

- Identity — Make two frames coincident with each other.
- Translation — Maintain an offset distance between two frame origins.
- Rotation — Maintain an angle between two frames.

The figure illustrates these spatial relationships. Letters B and F represent the two frames between which you apply a spatial relationship.



Identity

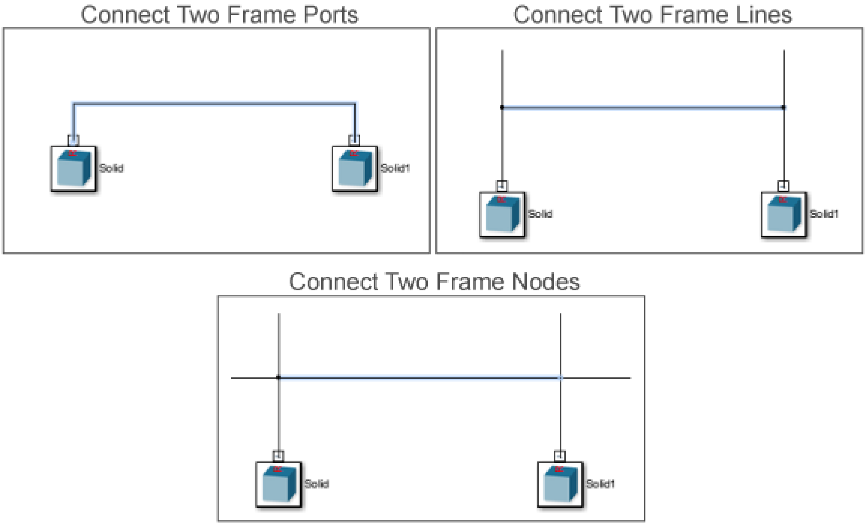
Translation

Rotation

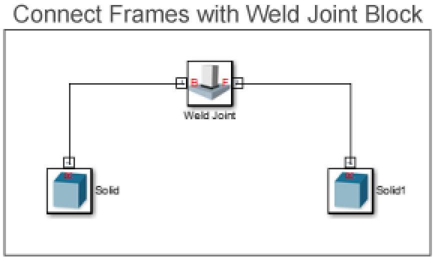
A frame port is any port with the frame icon □. A frame line is any connection line that joins two frame ports. A frame node is the junction point between two or more frame lines. You can connect one frame entity only to another frame entity. Connecting frame ports, lines, or nodes to other types of ports, lines, or nodes is invalid. For example, you cannot connect a frame port to a physical signal port.

## Identity Relationships

To make two frames coincident in space, connect the corresponding frame entities with a frame line. The frame line applies a rigid identity relationship between the two frames. During simulation, the two frames can move only as a single unit. They cannot move with respect to each other. The figure shows three ways to make two frames coincident.


Connect Two Frame Ports
Connect Two Frame Lines
Connect Two Frame Nodes

Alternatively, use the `Weld Joint` block to make two frames coincident for all time. The Weld Joint block fixes the relative positions and orientations of frames belonging to different rigid bodies.


Connect Frames with Weld Joint Block

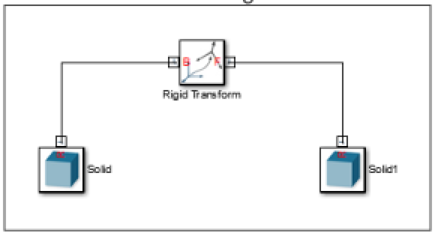
**Note:** Ensure each joint frame port connects rigidly to a Solid or Inertia block. The connection can be direct, through a connection line, or indirect, through one or more Rigid Transform blocks. Joint frame ports not rigidly connected to components with inertia (those containing at least one Solid or Inertia block) can cause a degenerate-mass error during simulation.

## Translation and Rotation

To separate two frames in space, you use the `Rigid Transform` block. By connecting two frame entities to the base and follower frame ports of this block, you apply the rigid transformation that the block specifies. Rigid transformations include translation and rotation.

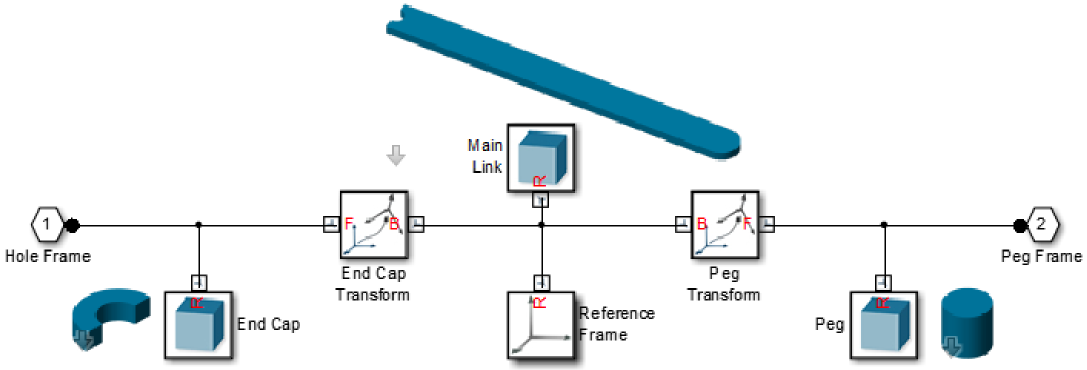
You can apply an offset distance between two frame origins, a rotation angle between the frame axes, or both. Two frames that you connect using a Rigid Transform block behave as a single entity. If you specify neither translation or rotation, the Rigid Transform block represents the identity relationship. The two frames become coincident in space. In the figure, a Rigid Transform block applies a rigid transformation between two solid reference frames.

Connect Frames with Rigid Transform Block

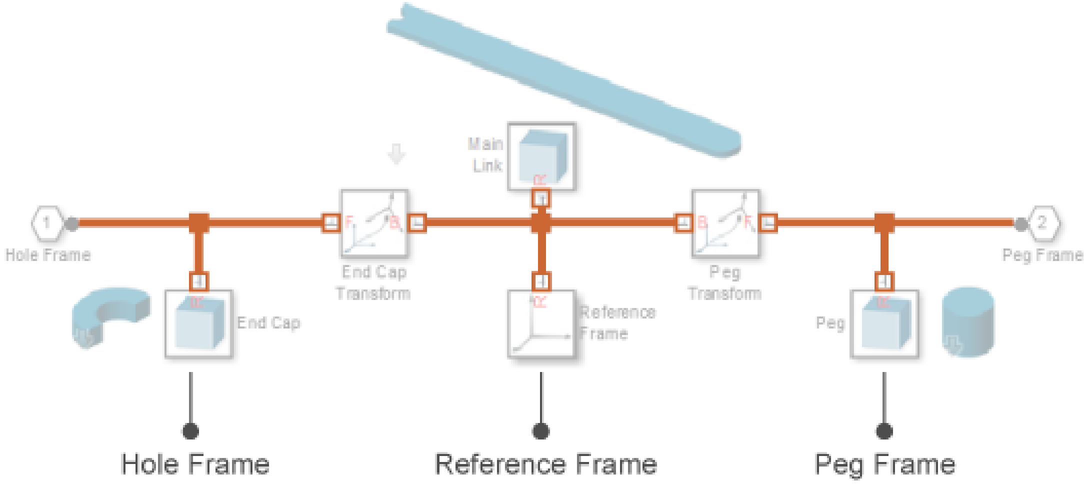


## Interpreting a Frame Network

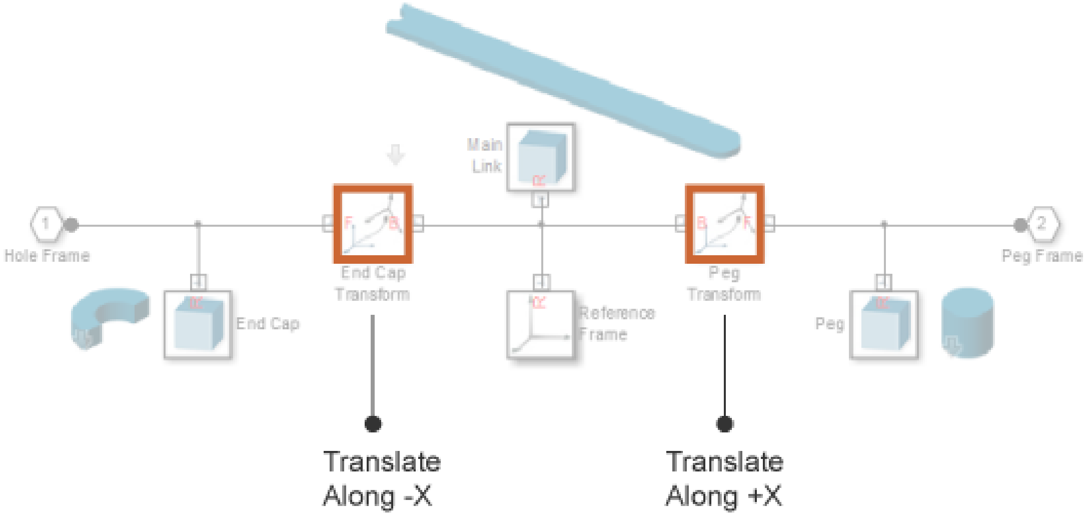
As an example, consider the frame network of a binary link. SimMechanics provides a model of this rigid body. To open it, at the MATLAB® command prompt, enter `sm_compound_body`. Double-click the Compound Body subsystem block to view the underlying block diagram. The figure shows this diagram.



To represent the binary link, the Compound Body subsystem contains three Solid blocks. The blocks represent the main, peg, and hole sections. Three frames provide the position and orientation of the three solids according to the guidelines that section “Identity Relationships” on page 1-7 introduces. Each group of frame ports, lines, and nodes that directly connect to each other represents one frame. The figure shows the three frames in the block diagram.



Two Rigid Transform blocks represent the spatial relationships between the three frames. One block translates the hole frame with respect to the reference frame along the common -X axis. The other block translates the peg frame with respect to the reference frame along the common +X axis. The figure shows these two blocks.



## Related Examples

- “Represent Box Frame Tree” on page 1-37
- “Represent Binary Link Frame Tree” on page 1-33

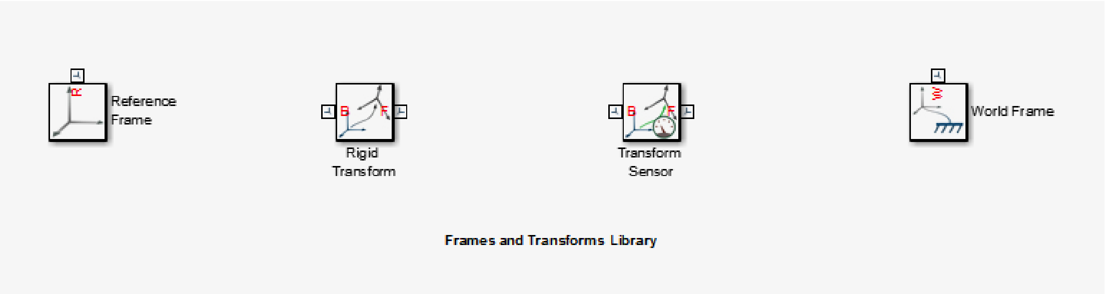
## More About

- “Working with Frames” on page 1-2
- “Frame Transformations” on page 1-17
- “World and Reference Frames” on page 1-11
- “Find and Fix Frame Issues” on page 1-56

# World and Reference Frames

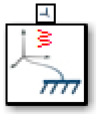
**In this section...**

Two preset frames are available in SimMechanics: World and Reference. These are standalone frames with respect to which you can define other frames in a model. New frames can in turn serve as the basis to define yet other frames. However, directly or indirectly, all frames depend on either World or Reference frames. Both frames are available as blocks in the Frames and Transforms library.

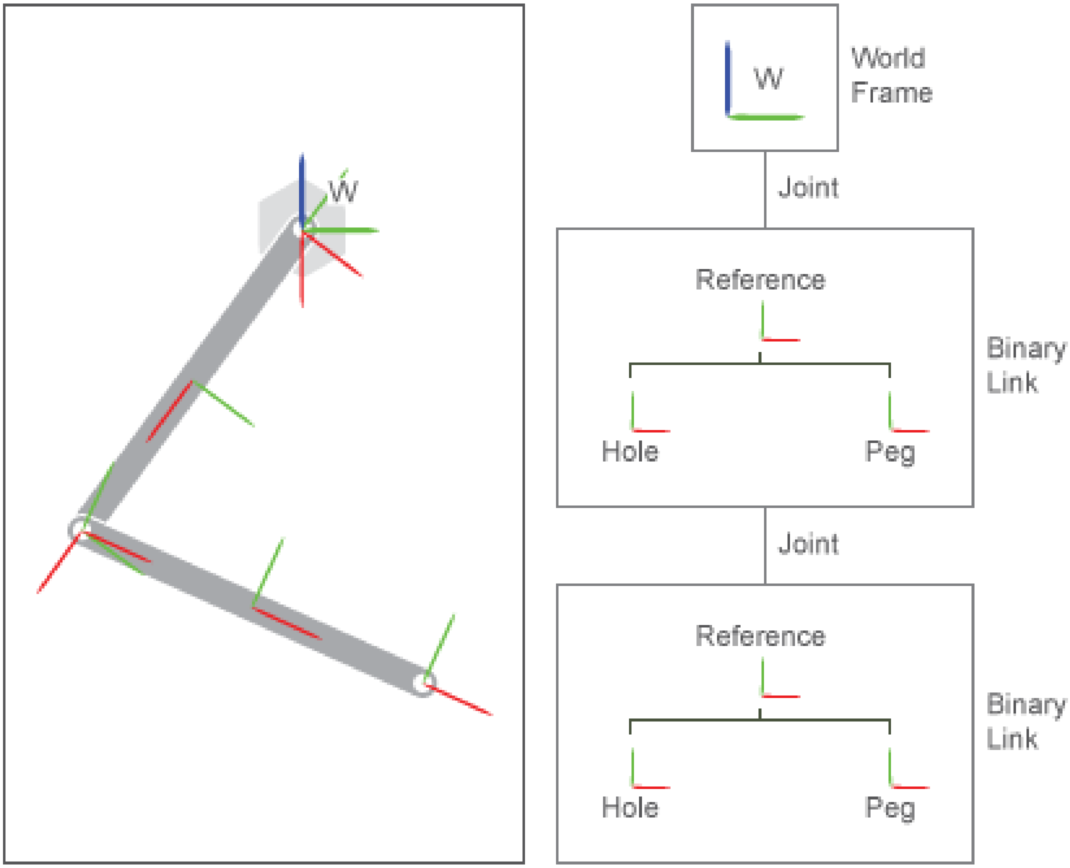


## World Frame

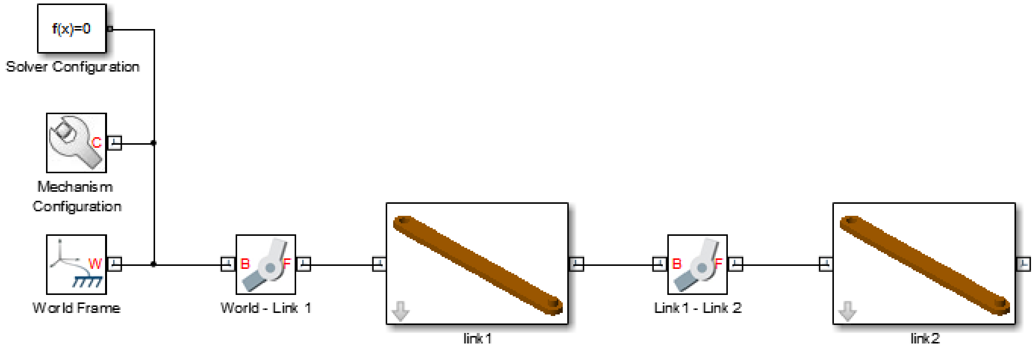
The World frame represents the external environment of a mechanical system. It is always at absolute rest, and therefore experiences zero acceleration. As a consequence, centripetal and other pseudo-forces are not present in the world frame, and it is said to be inertial. Rigidly connecting any frame to the World frame makes that frame also inertial. To add the World frame to a model, use the `World Frame` block.



The World frame is the ultimate reference frame. Its position and orientation are predefined and do not depend on any other frame. This property makes the World frame invaluable. You can always apply a transform to the World frame and obtain a new frame. Applying a transform to the resulting frame in turn yields more new frames, all indirectly related to the World frame. The result is a frame tree with the World frame at the root. The figure shows such a frame tree for a double-pendulum system.

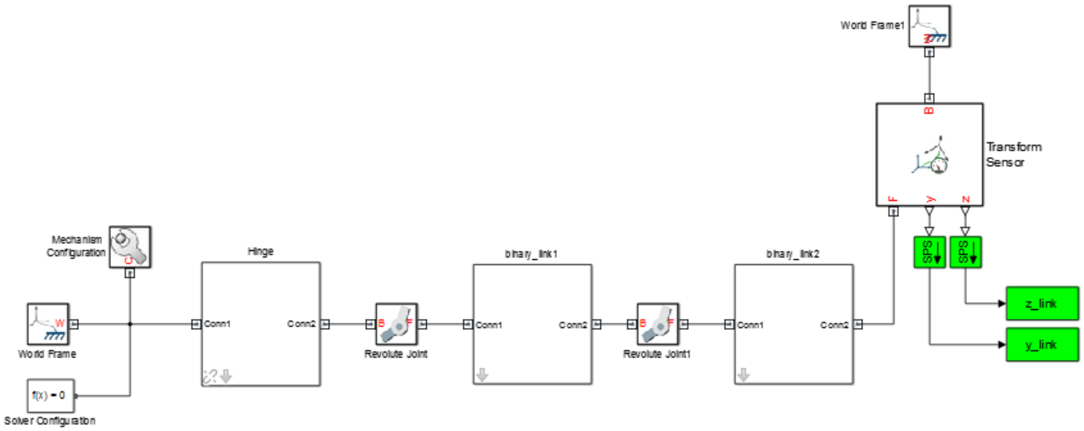


The double-pendulum block diagram is based on this frame tree. The World Frame block identifies the root of the frame tree. A Revolute Joint block applies the variable transform that relates the World frame to the binary link peg frame. A second Revolute Joint block applies a similar variable transform between the hole and peg frames of adjoining binary links. The figure shows this block diagram.



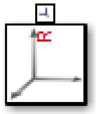
The World frame is present in every model. However, the World Frame block is strictly optional. If you do not add this block to a model, SimMechanics assigns one of the existing frames as the World frame. This implicit World frame connects to the rest of the model via an implicit 6-DOF joint, which in the absence of counteracting forces allows a machine to fall under gravity.

You can connect multiple World Frame blocks to a model. However, all World Frame blocks represent the same frame. In this sense, the World frame is unique. You can add multiple World Frame blocks to simplify modeling tasks, e.g., sensing motion with respect to the World frame. The figure shows the model of a double-pendulum with two World Frame blocks. Both World Frame blocks represent the same frame.

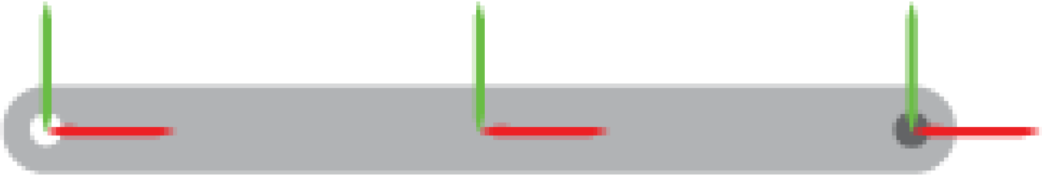


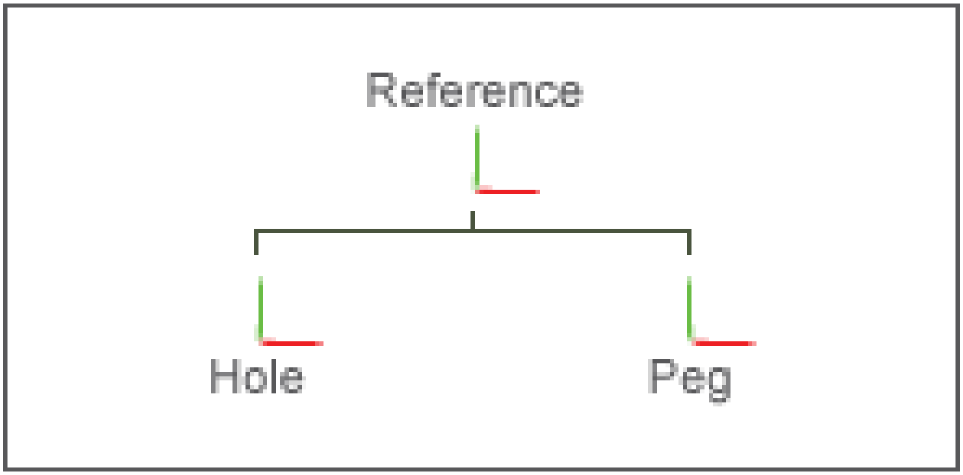
## Reference Frame

The Reference frame represents the root of a rigid body or multibody subsystem. Within a subsystem, it denotes the frame against which all remaining frames are defined. To add a Reference frame, use the `Reference Frame` block. Use this block to mark the top level of a subsystem frame tree.

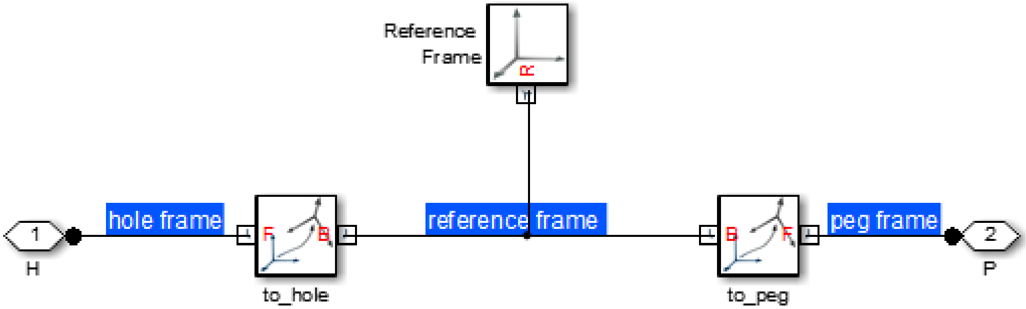


Applying a transform to the Reference frame yields other frames. Applying transforms to these other frames yields still more frames. The overall set of frames forms a frame tree with the Reference frame at the root. The figure shows such a frame tree for one of the binary links used in the double-pendulum system.





The block diagram of the binary link subsystem is based on this frame tree. The following figure shows the binary link block diagram. The Reference Frame block identifies the root of the frame tree. Rigid Transform block to_hole adds the hole frame. Rigid Transform block to_peg adds the peg frame. It is a simple task to add the main, peg, and hole solids once these frames are defined.



The distinguishing feature of the Reference frame is that it can move with respect to other frames. Depending on the dynamics of a model, a Reference frame can accelerate, giving rise to pseudo-forces that render this frame non-inertial. Rigidly connecting any frame to a non-inertial Reference frame makes that frame also non-inertial.

The Reference frame is present in every subsystem. However, the Reference Frame block is strictly optional. If you do not add this block to a subsystem, SimMechanics assigns one of the existing frames as the Reference frame.

## More About

- "Working with Frames" on page 1-2
- "Frame Transformations" on page 1-17
- "Representing Frames" on page 1-6

# Frame Transformations

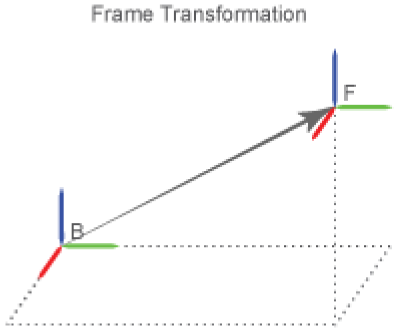
**In this section...**

To place a solid in space, with a given position and orientation, you use frames. By connecting the solid reference frame to another frame, you resolve its position and orientation within the model. For example, connecting the solid reference frame directly to the World frame causes their origins and axes to coincide. However, if the model does not yet contain the desired frame, you must first add it.

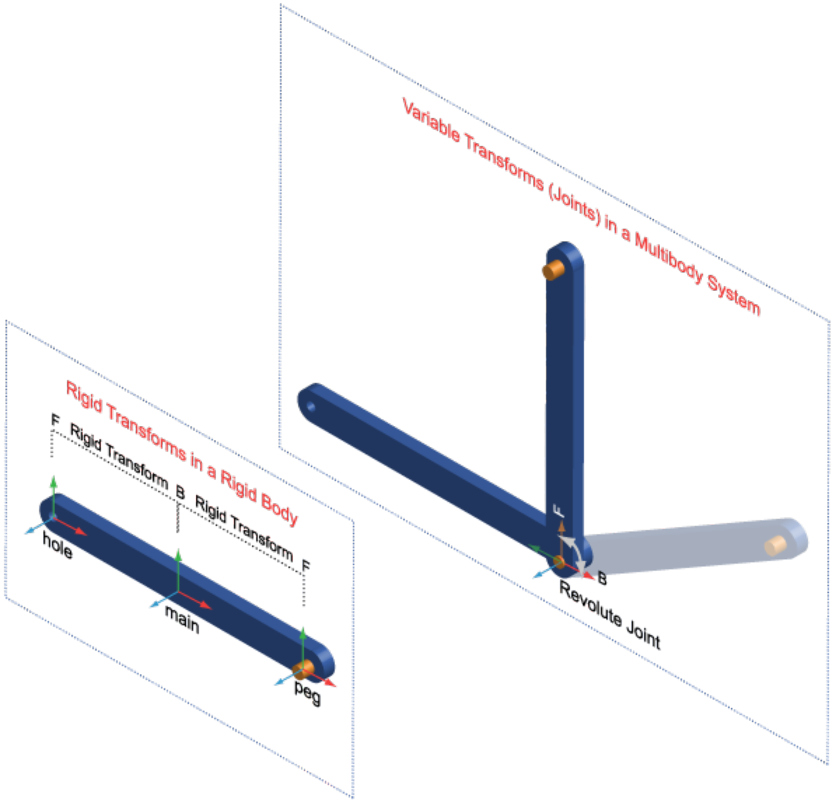
Adding a frame is the act of defining its position and orientation. Because these properties are relative, you must always define a frame with respect to another frame. Every model starts with one of two frame blocks you can use as reference: World Frame or Reference Frame. As a model grows, so does the number of frames that you can use as a reference.

## Rigid and Time-Varying Transformations

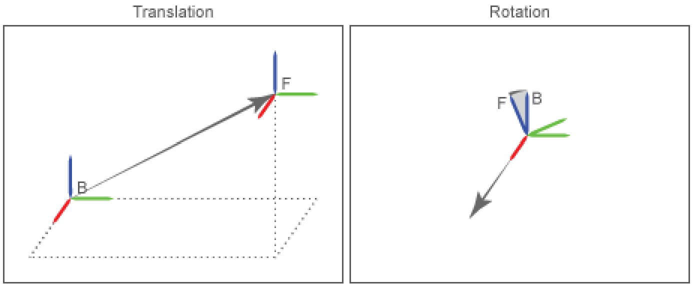
The spatial relationship between the two frames, the existing and the new, is called a frame transformation. When the transformation is fixed for all time, it is *rigid*. Two frames related by a rigid transformation can move with respect to the world, but never with respect to each other. In SimMechanics, you add a frame by applying a rigid transformation to an existing frame. The block you use for this task is the `Rigid Transform` block.



Frame transformations can also vary with time. In this case, the two frames that the transformation applies to can move with respect to each other. In SimMechanics, joint blocks provide the degrees of freedom that allow motion between two frames. Depending on the joint block, frames can move along or about an axis. For example, the `Revolute Joint` block allows two frames to rotate with respect to each other about a common +Z axis. Likewise, the `Prismatic Joint` block allows two frames to rotate with respect to each other along a common +Z axis. For more information about joints, see “Modeling Joint Connections” on page 3-2.

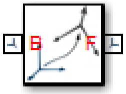

Variable Transforms (Joints) in a Multibody System
Rigid Transforms in a Rigid Body
F
Rigid Transform
B
Rigid Transform
F
hole
main
peg
F
B
Revolute Joint

You can apply two rigid transformations: rotation and translation. Rotation changes the orientation of the follower frame with respect to the base frame. Translation changes the position of the follower frame with respect to the base frame. A third, implicit, transformation is available—identity. This transformation is marked by the absence of both frame rotation and translation, making base and follower frames coincident in space.


Translation

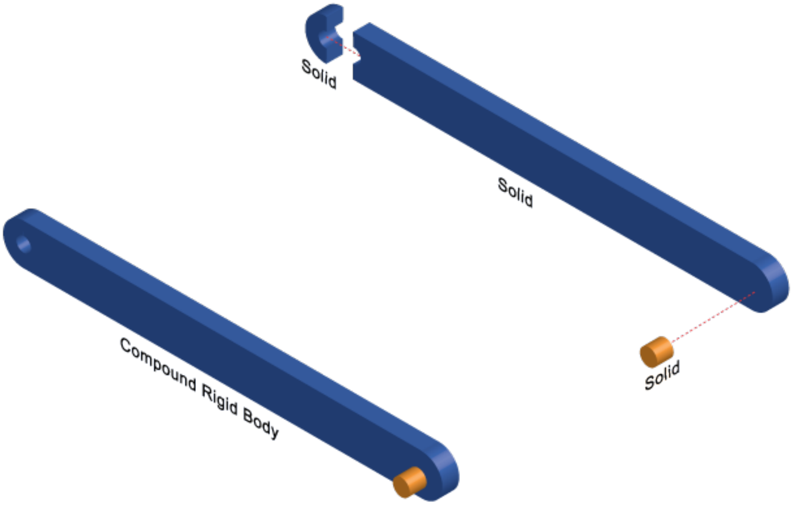
Rotation

Every rigid transformation involves two frames: a base and a follower. The base frame is a reference, the starting point against which you define the new frame. Any frame can act as the base frame. When you apply a rigid transformation, you do so directly *to* the base frame. The follower frame is the new frame — the transformed version of the base frame. The Rigid Transform block identifies base and follower frames with frame ports B and F, respectively.



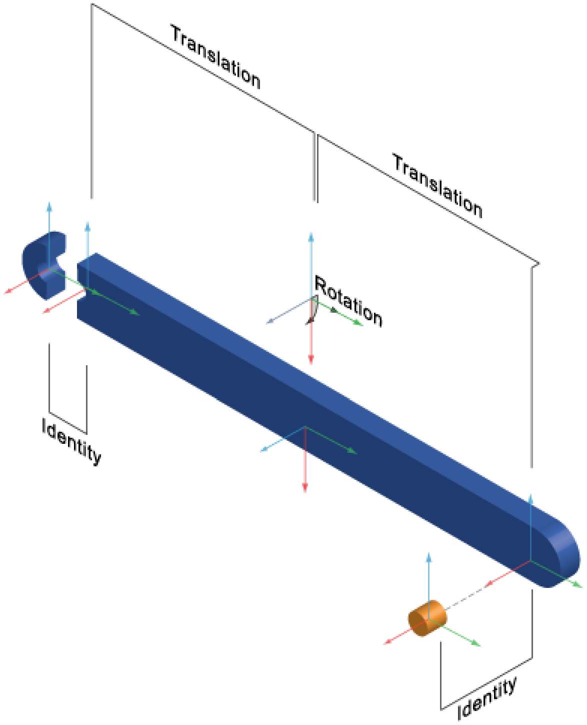
## Rigid Transformation Example

As an example, consider a binary link. You can model this rigid body with three elementary solids: main body, peg, and hole sections. This type of rigid body is known as *compound*. Each solid has a local reference frame, which is fixed with respect to the solid, but which can move with respect to the world. The figure shows the binary link compound rigid body and the three solids that comprise it.

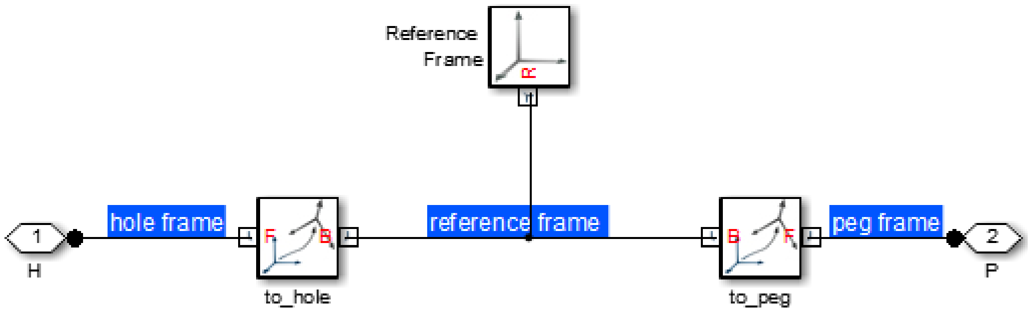


When modeling the binary link, the goal is to place the peg at one end of the link, and the hole section at the other end. The proper approach is to apply a rigid transformation between the main peg and peg reference frames, and main body and hole section reference frames. The transformations specify the separation distance and rotation angle, if any, between each pair of frames. Because the transformations are rigid, they constrain the solids to move as a single unit — a *rigid body*. The rigidly connected solids can move together with respect to the World frame, but never with respect to each other.

The figure shows the set of transformations used to model the binary link. These include translation, rotation, and identity. No Rigid Transform block is required to apply an identity transformation. See “Representing Frames” on page 1-6.



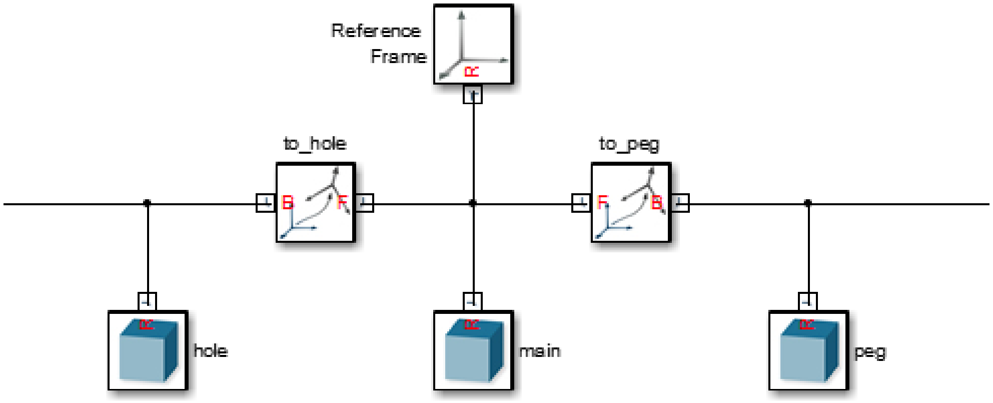
The block diagram, shown in the following figure, reflects the structure of the binary link. Three Solid blocks represent the main body, peg, and hole sections. Their R ports identify the respective reference frames. Two Rigid Transform blocks, named to_hole and to_peg apply the rigid transformations that relate the solid pairs main–hole and main–peg.



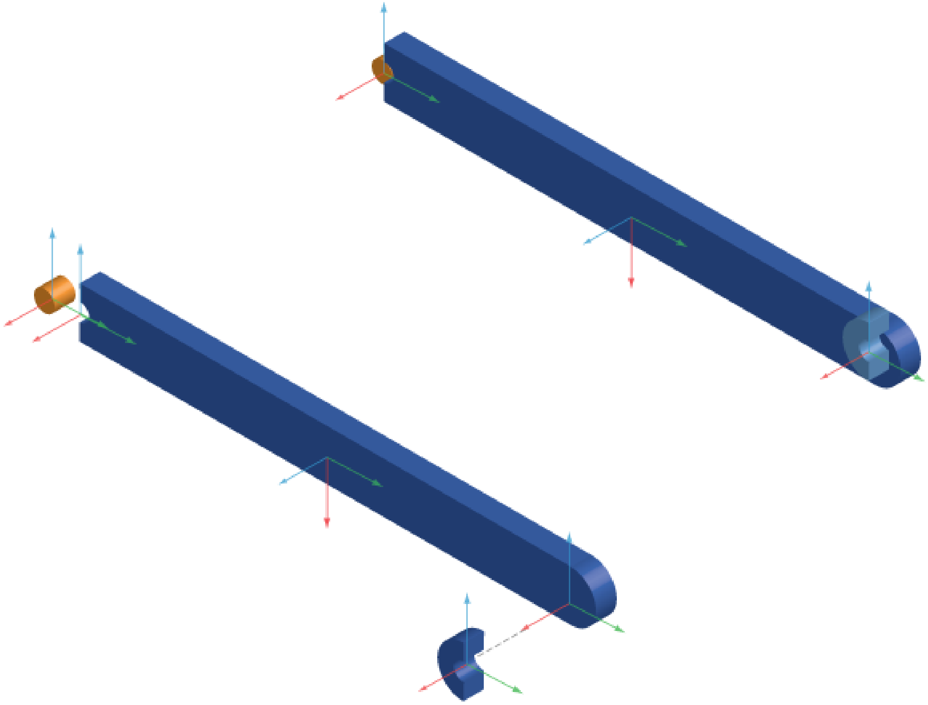
## Reversing Rigid Transformations

Rigid transformations describe the operation that takes the base frame into coincidence with the follower frame. In this sense, the transformation *acts on* the base frame. Switching base and follower port frames causes the transformation to act on a different frame, changing the relationship between the two frames. The result is a follower frame with different position and orientation and, as a consequence, a different rigid body subsystem.

Consider the binary link system. In the original configuration, rigid transformations translate the peg to the right of the main body and the hole to the left. To accomplish this, the main body frame connects to the base port frame of the corresponding Rigid Transform blocks, while the hole and peg frames connect to the follower port frames. When you switch base and follower frame ports, the transformations instead translate the main body to the right of the peg and to the left of the hole.



While in the first case the peg translated to the right of the main body, in the second case the peg translated to the left. The same principle applies to the hole. The figure shows the effect of switching base and follower frames in both Rigid Transform blocks of the binary link block diagram.



## Related Examples

- "Represent Binary Link Frame Tree" on page 1-33

# Rotation Methods

**In this section…**

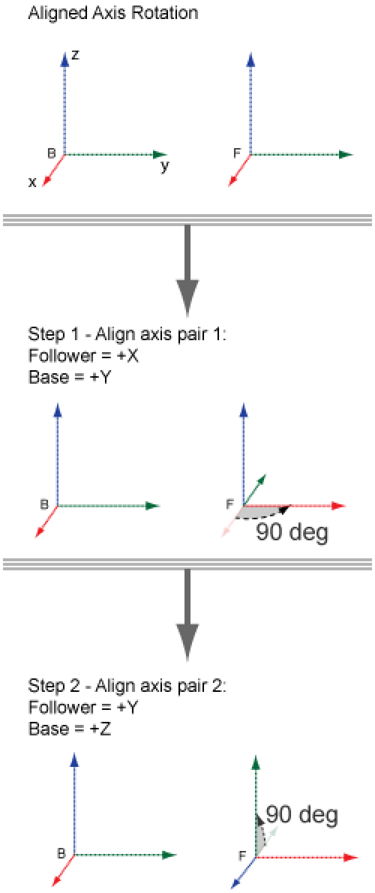
You can specify frame rotation using different methods. These include aligned axes, standard axis, and arbitrary axis. The different methods are available through the Rigid Transform block. The choice of method depends on the model. Select the method that is most convenient for the application.

## Specifying Rotation

Rotation is a relative quantity. The rotation of one frame is meaningful only with respect to another frame. As such, the Rigid Transform block requires two frames to specify a transformation: base and follower. The transformation operates on the base frame. For example, a translation along the +Z axis places the follower frame along the +Z axis from the base frame. Reversing frame ports is allowed, but the transformation is reversed: the base frame is now placed along the +Z axis from the follower frame.

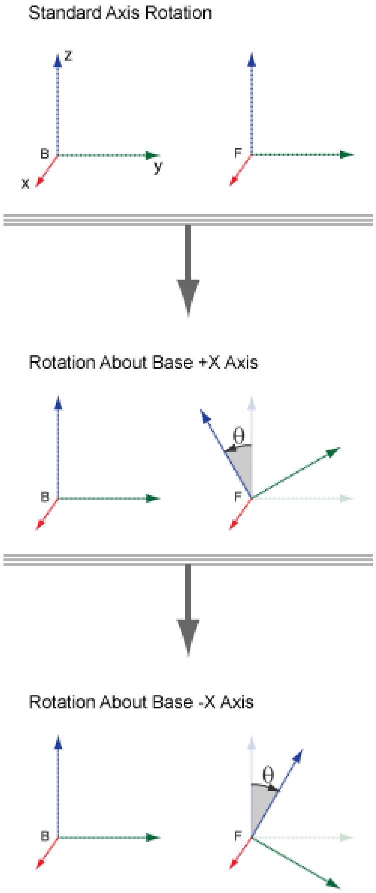
## Aligned Axes

Rotate two frames with respect to each other by aligning any two axes of one with any two axes of the other. The figure illustrates the aligned axes method.



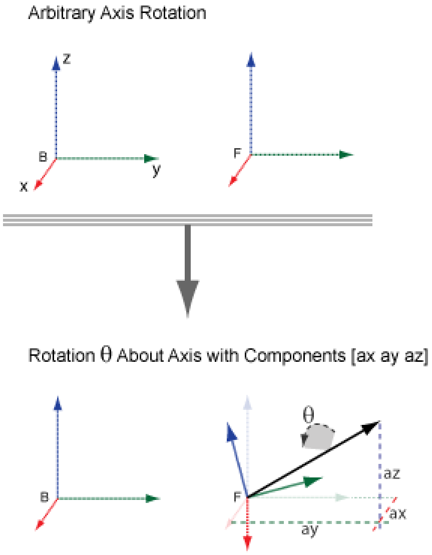
## Standard Axis

Rotate frames with respect to each other about one of the three base frame axes: X, Y, or Z.



## Arbitrary Axis

Rotate two frames with respect to each other about an arbitrary axis resolved in the base frame.


Arbitrary Axis Rotation
z
B
y
x
F
Rotation θ About Axis with Components [ax ay az]
B
θ
az
F
ax
ay

# Translation Methods

**In this section...**

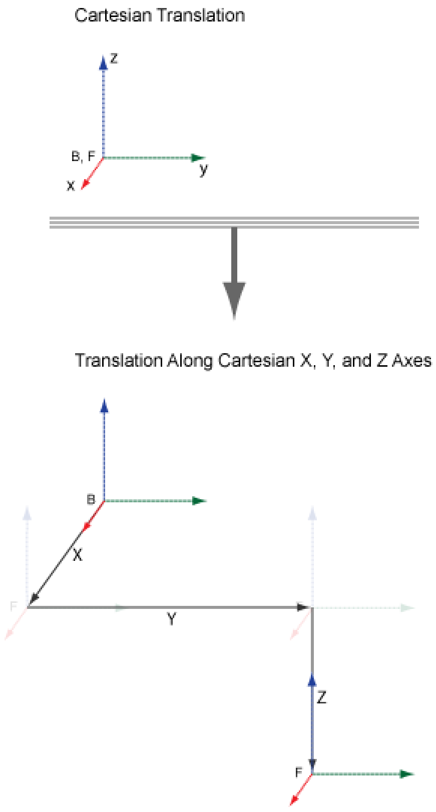
You can specify frame translation using different methods. These include Cartesian, standard axis, and cylindrical. The different methods are available through the `Rigid Transform` block. The choice of method depends on the model. Select the method that is most convenient for the application.

## Specifying Translation

Translation is a relative quantity. The translation of one frame is meaningful only with respect to another frame. As such, the Rigid Transform block requires two frames to specify a translation: base and follower. The transformation operates on the base frame. For example, a translation along the +Z axis places the follower frame along the +Z axis from the base frame. Reversing frame ports is allowed, but the transformation is reversed: the base frame is now placed along the +Z axis from the follower frame.

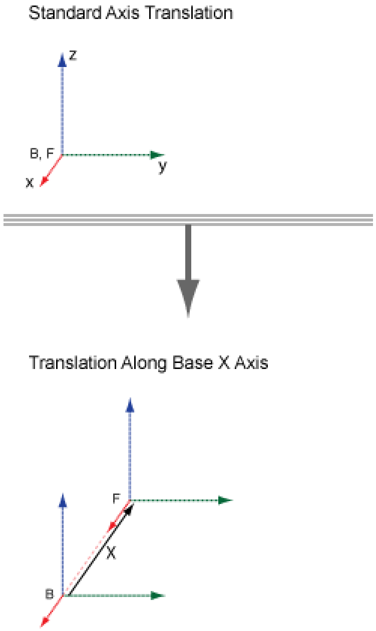
## Cartesian

Translate follower frame along arbitrary Cartesian vector resolved in the base frame.



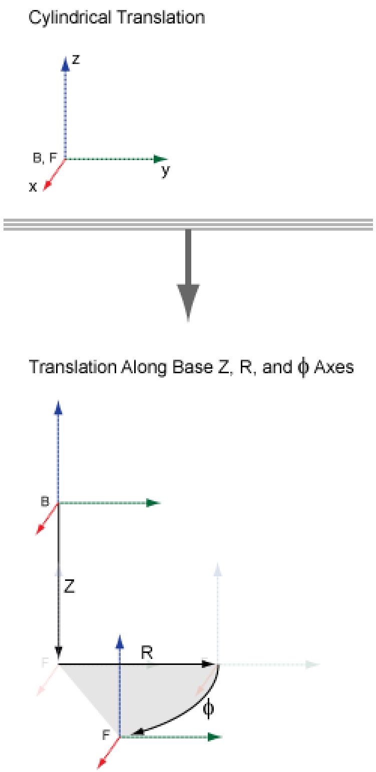
## Standard Axis

Translate follower frame along one of the three axes of the base frame.



## Cylindrical

Translate follower frame along cylindrical axes resolved in the base frame.



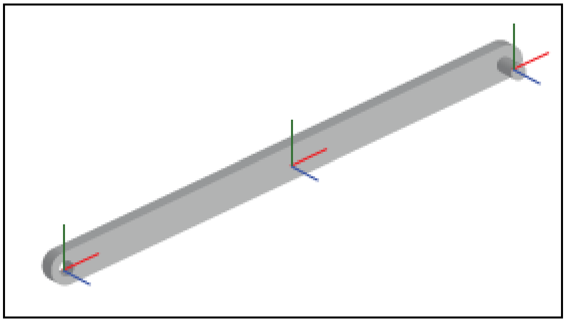
Cylindrical Translation

Translation Along Base Z, R, and ϕ Axes

# Represent Binary Link Frame Tree

## Model Overview

In this example, you model the frame tree of a binary link. This tree contains three frames to which you later connect solid elements. You specify these frames using the Rigid Transform block.



You can promote subsystem reusability by parameterizing link dimensions in terms of MATLAB variables. In this example, you initialize the variables in a subsystem mask. You then specify their numerical values in the subsystem dialog box. The table shows the variables used in this example.

| Dimension | MATLAB Variable |
|---|---|
| Length | L |
| Width | W |

| Dimension | MATLAB Variable |
|---|---|
| Thickness | T |

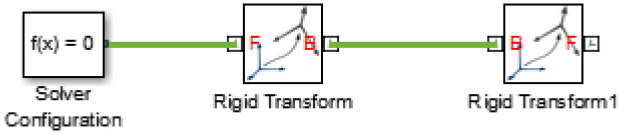
## Build Model

1 Drag these blocks into a new model.

| Block | Quantity | Library |
|---|---|---|
| `Rigid Transform` | 2 | Frames and Transforms |
| `Solver Configuration` | 1 | Simscape™ Utilities |

2 Connect and name the blocks as shown in the figure.

**Note:** You must connect the frame ports exactly as shown in the figure. Ensure the base frame ports connect directly to each other. If they do not, the frames created will differ from this example.



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

| Parameter | Setting |
|---|---|
| **Translation > Method** | Select `Standard Axis`. |
| **Translation > Axis** | Select `-X`. |
| **Translation > Offset** | Enter `L/2`. Select units of `cm`. |

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

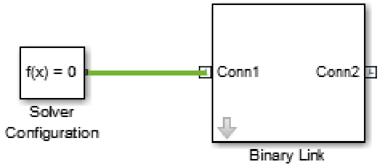
| Parameter | Setting |
|---|---|
| **Translation** > **Method** | Select `Cartesian`. |
| **Translation** > **Offset** | Enter `[L/2 0 3/2*T]`. Select units of `cm`. |

## Generate Binary Link Subsystem

To initialize the MATLAB dimension variables used to specify the frame transforms, convert the binary link block diagram into a subsystem and use the subsystem mask:

1 Select all the blocks excluding Solver Configuration.

2 Generate a new subsystem, e.g., by pressing **Ctrl+G**.



3 Create a subsystem mask, e.g., by selecting the Subsystem block and pressing **Ctrl +M**.

4 In the **Parameters & Dialog** tab, add three Edit fields to the **Parameters** folder. Then, specify the following parameters and click **OK**.

| Prompt | Name |
|---|---|
| `Length` | `L` |
| `Width` | `W` |
| `Thickness` | `T` |

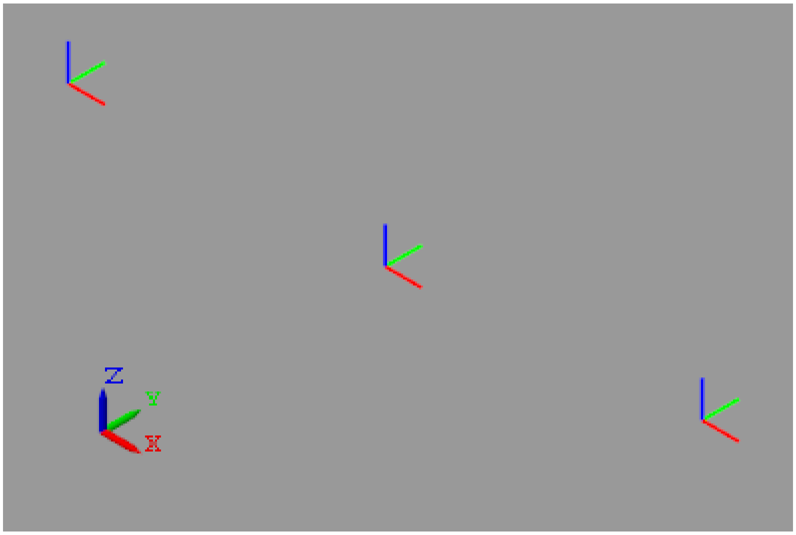
5 In the Subsystem dialog box, specify these parameters.

| Parameter | Value |
|---|---|
| Length | `30` |

| Parameter | Value |
|---|---|
| Width | 2 |
| Thickness | 0.8 |

## Visualize Model

Update the block diagram. You can perform this task from the Simulink® Editor menu bar, by selecting **Simulation** > **Update Diagram**. Then, in the Mechanics Explorer menu bar, select **View** > **Show Frames**. The visualization pane shows the binary link frame tree.



## Open Reference Model

To see a completed version of the frame tree model, at the MATLAB command prompt enter `smdoc_binary_link_frames`.

## Related Examples

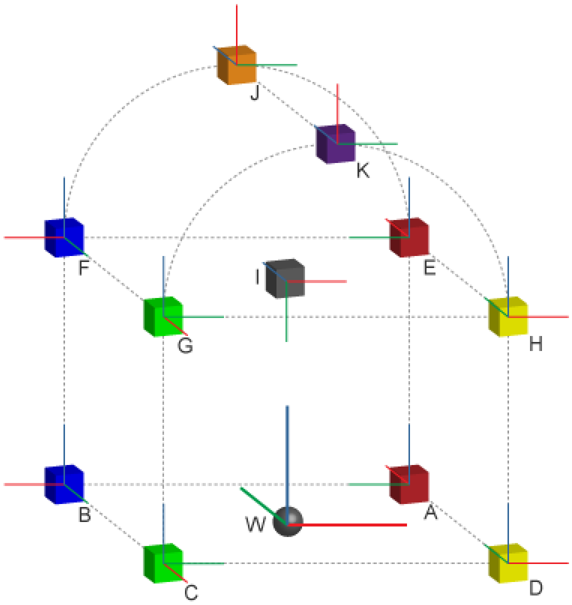
- "Model Binary Link" on page 2-74

# Represent Box Frame Tree

**In this section...**

## Model Overview

In SimMechanics, you can rigidly connect multiple Solid blocks to represent a complex rigid body. To position and orient different solids with respect to each other, you create a frame network that you can connect the solids to. The frame network contains `Rigid Transform` blocks that specify the spatial relationships between the different frames. In this example, you represent the frame tree for a box shape.



The example highlights the Rigid Transform block as the basic tool that you use to specify spatial relationships between frames and the solids that connect to them. The complete frame network is complex. It highlights nearly every type of rigid transformation that you can apply between two frames.

The modeling process in this example contains four stages:

1. Add World Frame (W).

   This is the ultimate reference frame against which you define all other frames.
2. Add the frames of the box bottom plane (frames A-D in the figure).

   You define these frames directly with respect to the World frame.
3. Add the frames of the box top plane (frames E-I in the figure).

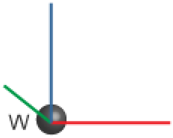
   You define these frames directly with respect to the box bottom plane frames.
4. Add the frames of the box arch (frames K and J in the figure).

   You define these frames directly with respect to the center frame of the box top plane.

This example is based on model `sm_frame_tree`, which accompanies your SimMechanics installation. To open this model, at the MATLAB command line, enter `sm_frame_tree`.

## Start Model

Start a new model. Then, add a global reference frame that you can use to define other frames.



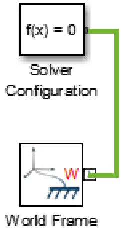
Use the World Frame block to represent the World frame:

1 Start a new model.

2 Drag the following blocks into the model.

| Library | Block | Quantity |
|---|---|---|
| Frames and Transforms | `World Frame` | 1 |
| Simscape Utilities | `Solver Configuration` | 1 |

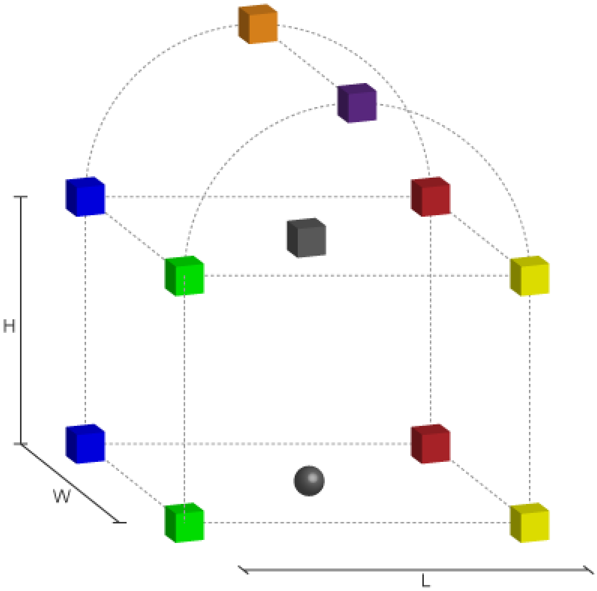
3 Connect the blocks as they appear in the figure.



## Initialize Model Workspace Parameters

To specify the distance offsets between frames, you use Rigid Transform blocks. In this example, you specify the distance offsets in terms of MATLAB variables that you initialize in the model workspace. The table lists these variables.

| Dimension | Variable |
|---|---|
| Length | L |
| Width | W |
| Height | H |



To initialize the MATLAB variables:

1 On the Simulink menu bar, click **Tools** > **Model Explorer**.

2 On the Model Hierarchy pane, double-click the name of your model (e.g. **frame_tree**).

3 Click **Model Workspace**.

4 On the **Model Workspace** pane, in the **Data Source** drop-down list, select `MATLAB Code`.

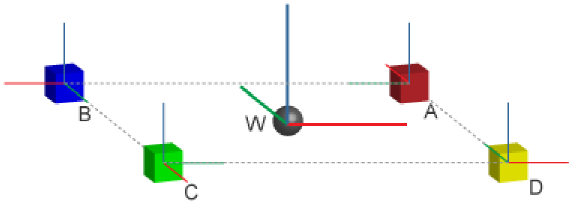
5 In the **MATLAB Code** section that appears, enter the following code:

```
% Size of Cube
L = 12;
W = 10;
H = 8;
```

6 Click **Reinitialize from Source**.

## Add Bottom Plane Frames

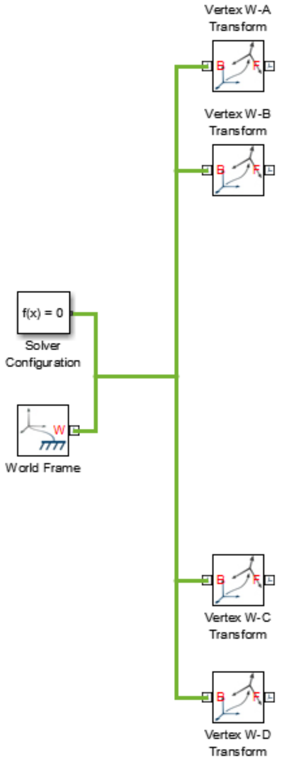
The World frame is the ultimate reference frame in a model. Now that you added the World frame to your model, you can define other frames with respect to it. You do this using the Rigid Transform block.



To define the four corner frames of the bottom box plane:

1 From the Frames and Transforms library, drag four Rigid Transform blocks to the model.

2 Connect and name the blocks as they appear in the figure.



3 Double-click the Vertex W-A Transform block and, in the dialog box, specify the parameters that the table provides.

| Parameter Section | Parameter | Value |
|---|---|---|
| **Rotation** | **Method** | Select `Standard Axis` |
|  | **Axis** | Select `+Z` |
|  | **Angle** | Enter `90` (`deg`) |
| **Translation** | **Method** | Select `Cartesian` |
|  | **Offset** | Enter `[L/2 W/2 0]` (`cm`) |

4 Double-click the Vertex W-B Transform block and, in the dialog box, specify the parameters that the table provides.

| Parameter Section | Parameter | Value |
|---|---|---|
| **Rotation** | **Method** | Select `Aligned Axis` |
| | **Pair 1 > Follower/Base** | Select `+X/-X` |
| | **Pair 2 > Follower/Base** | Select `+Y/-Y` |
| **Translation** | **Method** | Select `Cartesian` |
| | **Offset** | Enter `[-L/2 W/2 0]` (`cm`) |

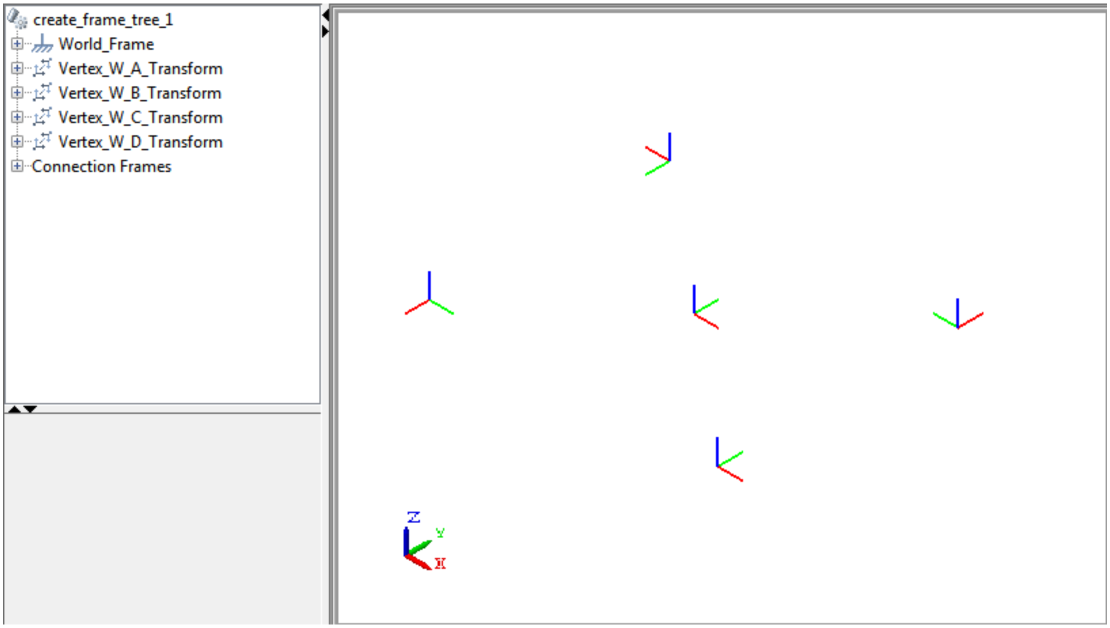
5 Double-click the Vertex W-C Transform block and, in the dialog box, specify the parameters that the table provides.

| Parameter Section | Parameter | Value |
|---|---|---|
| **Rotation** | **Method** | Select `Standard Axis` |
| | **Axis** | Select `+Z` |
| | **Angle** | Enter `270` (`deg`) |
| **Translation** | **Method** | Select `Cartesian` |
| | **Offset** | Enter `[-L/2 -W/2 0]` (`cm`) |

6 Double-click the Vertex W-D Transform block and, in the dialog box, specify the parameters that the table provides.

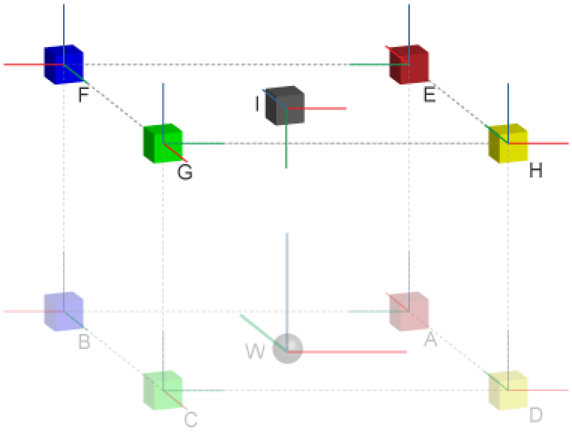
| Parameter Section | Parameter | Value |
|---|---|---|
| **Rotation** | **Method** | Select `None` |
| **Translation** | **Method** | Select `Cartesian` |
| | **Offset** | Enter `[L/2 -W/2 0]` (`cm`) |

To visualize the frames that you just added, on the Simulink menu bar, select **Simulation** > **Update Diagram**. Mechanics Explorer opens with a static 3-D display of your model. To view the position and orientation of each frame, in the Mechanics Explorer menu bar, select **View** > **Show Frames**.



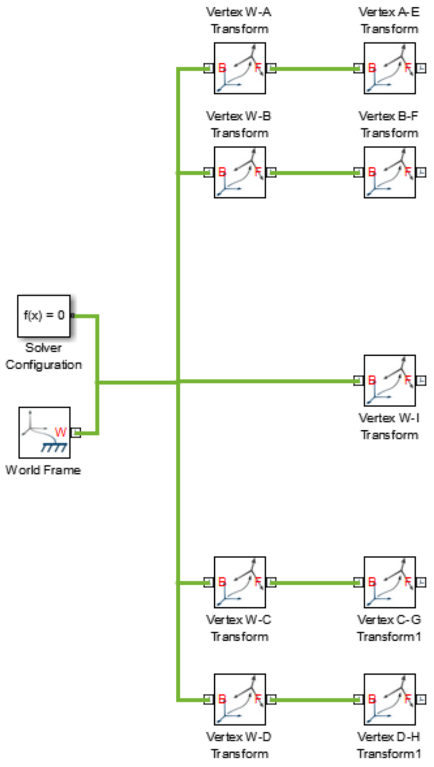
## Add Top Plane Frames

You can now define the top plane frames with respect to the bottom plane frames.



To add the top plane frames:

1 From the Frames and Transforms library, drag five Rigid Transform blocks.
2 Connect and name the blocks as they appear in the figure.



3 Double-click the following blocks:

   - Vertex A-E Transform
   - Vertex B-F Transform
   - Vertex C-G Transform
   - Vertex D-H Transform

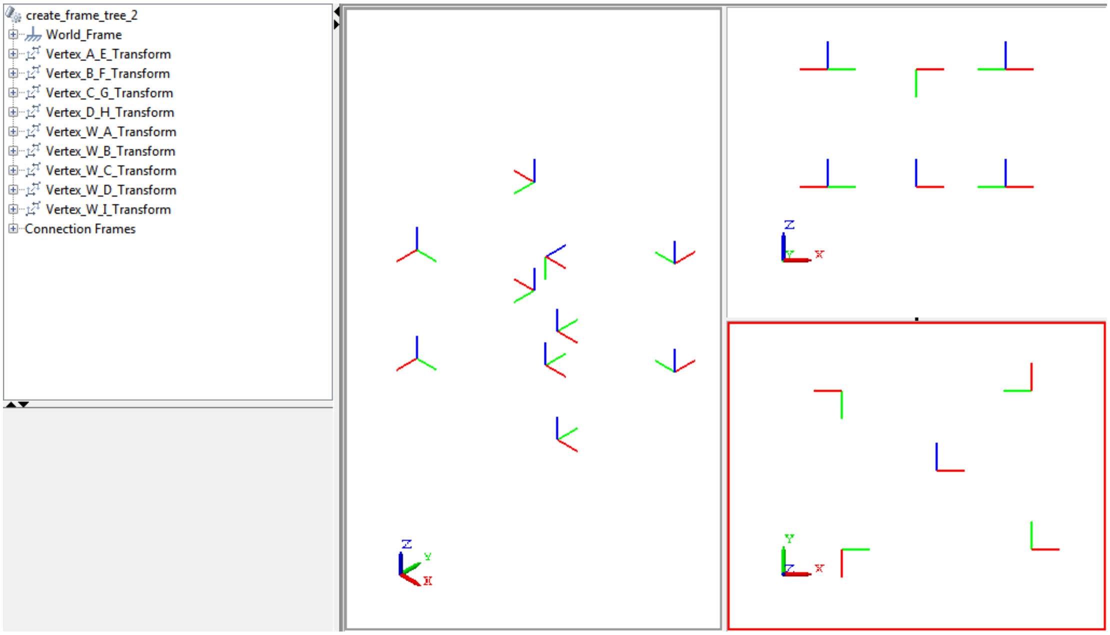
4 In each block dialog box, specify the following parameters.

| Parameter Section | Parameter | Value |
|---|---|---|
| **Rotation** | **Method** | Select `None` |
| **Translation** | **Method** | Select `Standard Axis` |
| | **Axis** | `+Z` |
| | **Offset** | Enter `H` (`cm`) |

5 Double-click the Vertex W-I Transform block and, in the dialog box, specify the following parameters.

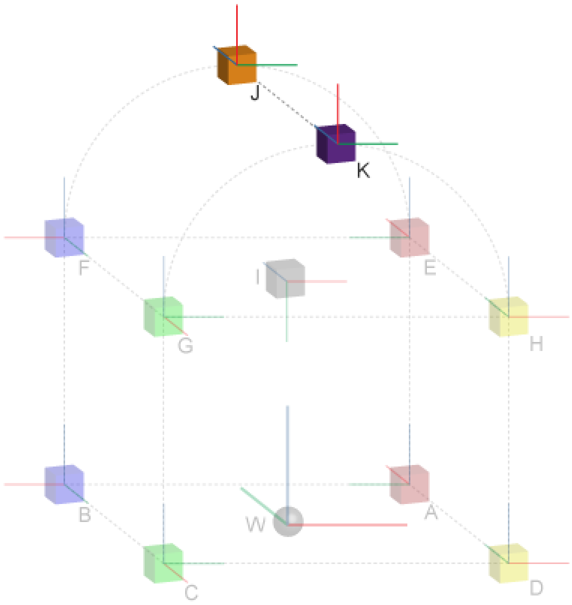
| Parameter Section | Parameter | Value |
|---|---|---|
| **Rotation** | **Method** | Select `Aligned Axes` |
| | **Pair 1 > Follower/Base** | Select `+Y`/`-Z` |
| | **Pair 2 > Follower/Base** | Select `+Z`/`+Y` |
| **Translation** | **Method** | Select `Standard Axis` |
| | **Axis** | `+Z` |
| | **Offset** | Enter `H` (`cm`) |

To visualize the frames that you just added, on the Simulink menu bar, select **Simulation > Update Diagram**. Mechanics Explorer updates the 3-D view of the box frame tree.



## Add Arch Frames

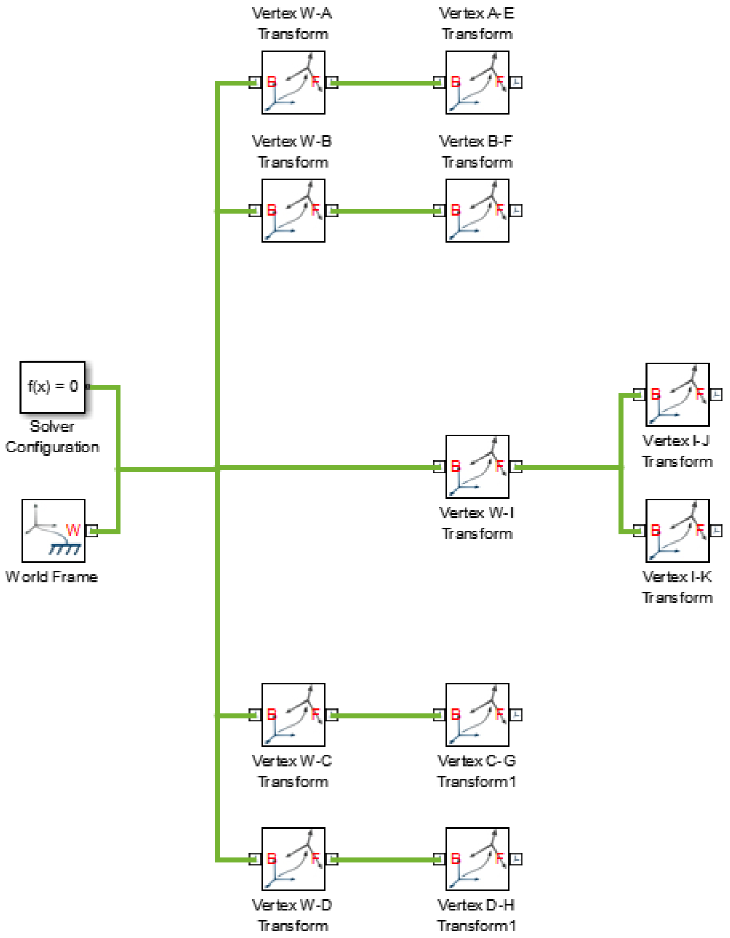
Finally, add the two arch frames. As before, use the Rigid Transform block to define these frames. Define them with respect to the center frame of the top plane (frame I).



To define the arch frames:

1 From the Frames and Transforms library, drag two Rigid Transform blocks.

2 Connect and name the blocks as they appear in the figure.



3 Double-click the Vertex I-J Transform block and, in the dialog box, specify the parameters that the table provides.

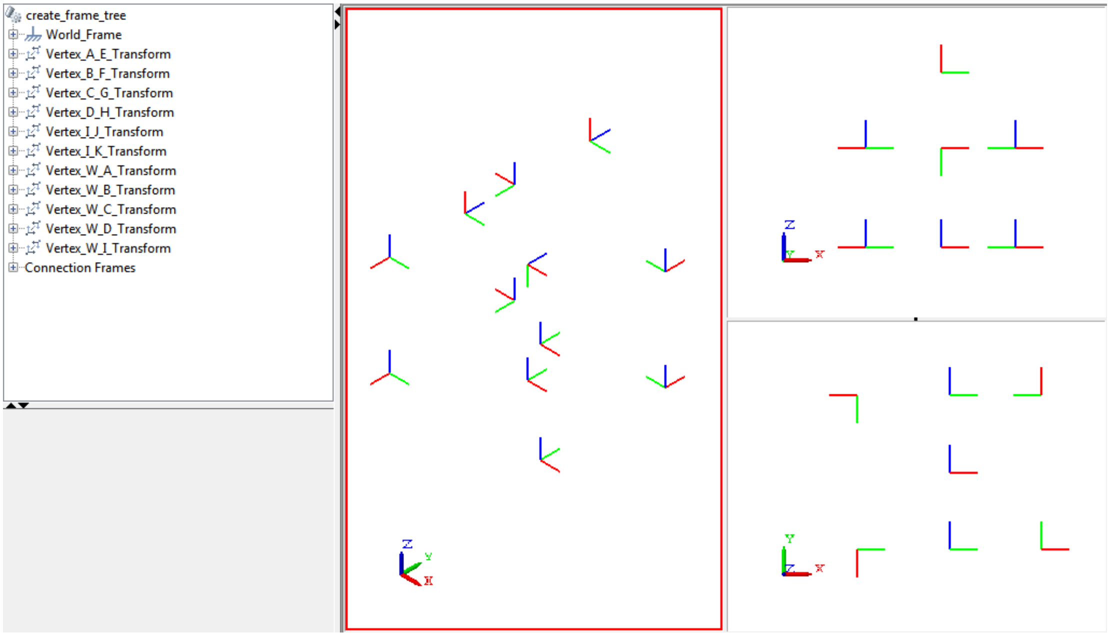
| Parameter Section | Parameter | Value |
|---|---|---|
| **Rotation** | **Method** | Select `Standard Axis` |
| | **Axis** | Select `+Z` |
| | **Angle** | Enter `-90` (`deg`) |

| Parameter Section | Parameter | Value |
|---|---|---|
| **Translation** | **Method** | Select `Cylindrical` |
| | **Radius** | Enter `L/2` (`cm`) |
| | **Theta** | Enter `-90` (`deg`) |
| | **Z Offset** | Enter `W/2` (`cm`) |

4 Double-click the Vertex I-K Transform block and, in the dialog box, specify the parameters that the table provides.

| Parameter Section | Parameter | Value |
|---|---|---|
| **Rotation** | **Method** | Select `Standard Axis` |
| | **Axis** | Select `+Z` |
| | **Angle** | Enter `-90` (`deg`) |
| **Translation** | **Method** | Select `Cylindrical` |
| | **Radius** | Enter `L/2` (`cm`) |
| | **Theta** | Enter `-90` (`deg`) |
| | **Z Offset** | Enter `-W/2` (`cm`) |

To visualize the frames that you just added, on the Simulink menu bar, select **Simulation** > **Update Diagram**. Mechanics Explorer opens with a static 3-D display of your model. To view the position and orientation of each frame, on the Mechanics Explorer tool bar, check that the frame visibility icon is toggled on.



## Save Model

Save the model as `frame_tree` in a convenient folder. In a subsequent example, you use Graphic blocks to represent each frame with a graphic icon. See “Visualize Box Frame Tree” on page 1-52

## Related Examples

- “Visualize Box Frame Tree” on page 1-52
- “Represent Binary Link Frame Tree” on page 1-33

## More About

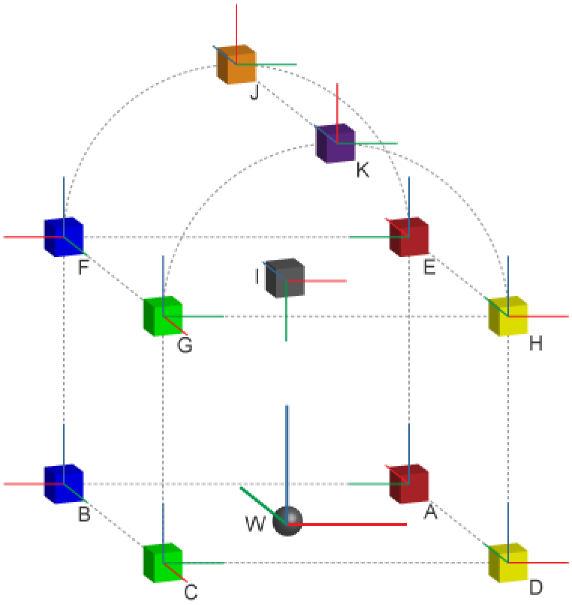
- “Representing Frames” on page 1-6
- “Frame Transformations” on page 1-17
- “World and Reference Frames” on page 1-11

# Visualize Box Frame Tree

**In this section...**

## Model Overview

To visualize a frame or frame network, you can use the `Graphic` block. By connecting this block to a frame, you add a graphic icon to that frame. The graphic icon has zero inertia and it does not affect model dynamics during simulation. In this example, you use Graphic blocks to add graphic icons to the box frame tree that you modeled in a previous example. See "Represent Box Frame Tree" on page 1-37.



## Build Model

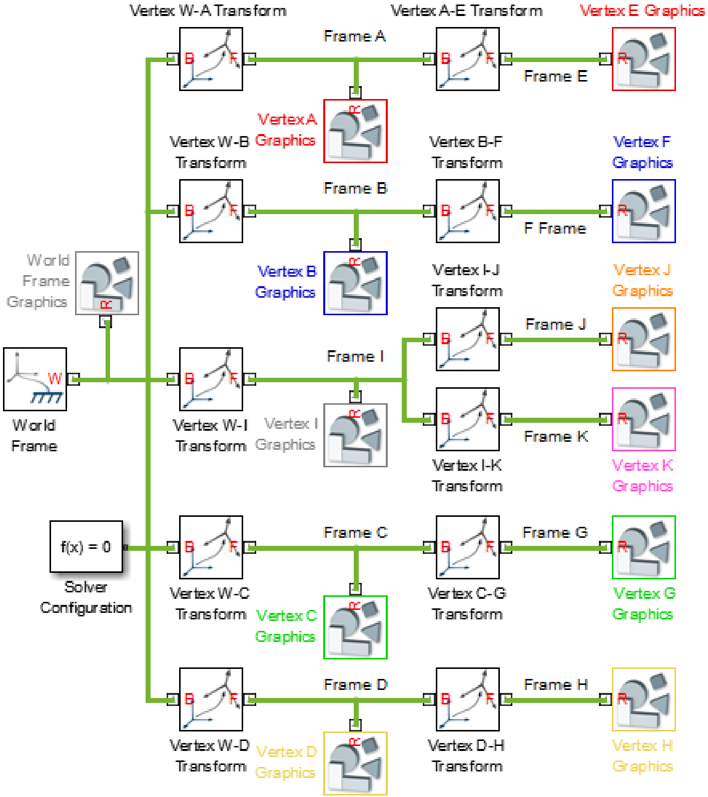
To add a graphic icon to each frame in your model:

1. Open model `frame_tree`.

   This is the model that you created in example “Represent Box Frame Tree” on page 1-37.

2. From the Body Elements library, drag 12 Graphic blocks to that model.

3. Connect and name the blocks as they appear in the figure.



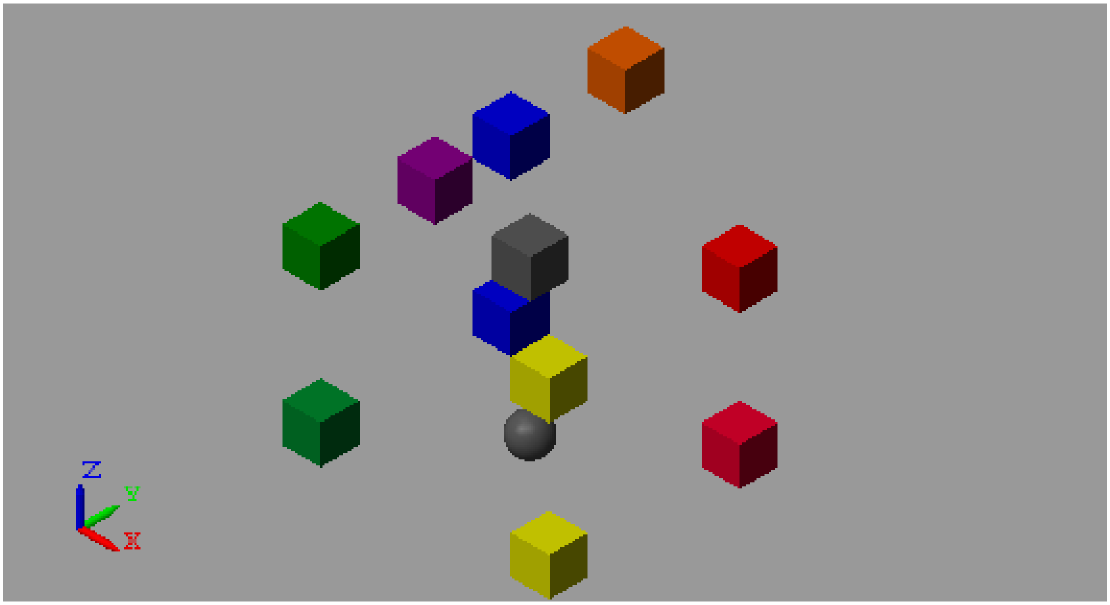
4. Double-click each Graphic block.

5. In the dialog box, specify parameters according to the following table.

| Graphic Block | Color | Shape | Size |
|---|---|---|---|
| World Frame Graphics | `[0.4 0.4 0.4]` | `Sphere` | `25` |
| Vertex I Graphics | | `Cube` | |
| Vertex A Graphics | `[1.0 0.0 0.0]` | | |
| Vertex E Graphics | | | |
| Vertex B Graphics | `[0.0 0.0 1.0]` | | |
| Vertex F Graphics | | | |
| Vertex C Graphics | `[0.0 0.6 0.2]` | | |
| Vertex G Graphics | | | |
| Vertex D Graphics | `[1.0 1.0 0.0]` | | |
| Vertex H Graphics | | | |
| Vertex J Graphics | `[1.0 0.4 0.0]` | | |
| Vertex K Graphics | `[0.6 0.0 0.6]` | | |

## Visualize Model

You can now visualize your model in Mechanics Explorer. To do this, on the Simulink menu bar, select **Simulation** > **Update Diagram**. Mechanics Explorer opens with a static 3-D display of your model. Rotate, pan, and zoom to explore.



## Related Examples

- "Represent Box Frame Tree" on page 1-37
- "Represent Binary Link Frame Tree" on page 1-33

## More About

- "Representing Frames" on page 1-6
- "Frame Transformations" on page 1-17

# Find and Fix Frame Issues

| In this section... |
|---|
| "Rigidity Loops" on page 1-56 |
| "Shorted Rigid Transform Blocks" on page 1-57 |

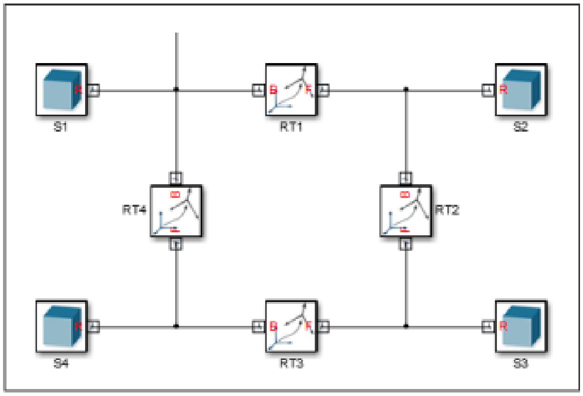
If your model contains an invalid frame connection, SimMechanics issues an error and the model does not simulate. Possible error sources include:

- Rigidity loops — Rigidly connecting multiple frames in a closed loop
- Shorted Rigid Transform Blocks — Rigidly connecting base and follower frame ports of a Rigid Transform block

## Rigidity Loops

A rigidity loop is a closed loop of Rigid Transform blocks. The loop contains one redundant Rigid Transform block that over-constrains the subsystem. If a rigidity loop is present, SimMechanics issues an error and the model does not simulate.

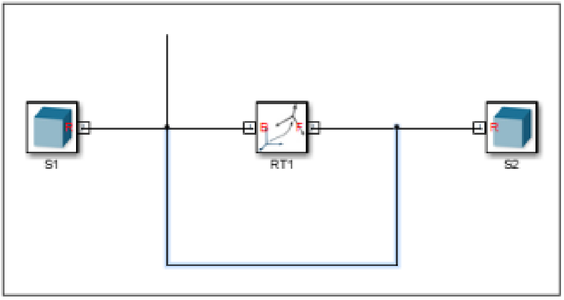
To remove the simulation error, disconnect one Rigid Transform block. This step removes the redundant constraint, and allows the model to simulate. The following figure shows a rigidity loop. The loop contains four Rigid Transform blocks directly connected to each other.



## Shorted Rigid Transform Blocks

A shorted Rigid Transform block contains a direct connection line between base (B) and follower frames (F). The connection line makes the two port frames coincident in space. However, the Rigid Transform block enforces a spatial transformation that translates or rotates one port frame relative to the other. The result is a conflict in the frame definition.

If a shorted Rigid Transform block is present, SimMechanics issues an error and the model does not simulate. The error remains even if the Rigid Transform block specifies no rotation and no translation. To remove the simulation error, delete the direct connection line between base and follower frame ports of the Rigid Transform block. The following figure shows a shorted Rigid Transform block.



## Related Examples

- “Represent Box Frame Tree” on page 1-37
- “Represent Binary Link Frame Tree” on page 1-33

## More About

- “Representing Frames” on page 1-6
- “Frame Transformations” on page 1-17

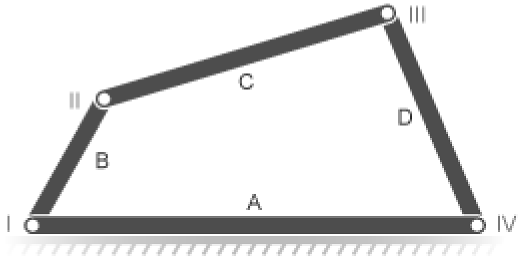
# Rigid Bodies

# SimMechanics Bodies

**In this section...**

## Rigid Body Essentials

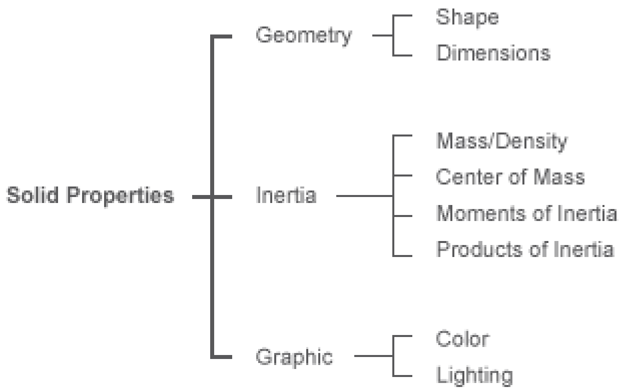
Bodies are the basic components of a SimMechanics model. They are the parts that you interconnect with joints and constraints to model an articulated mechanism or machine. As an example, a four-bar linkage contains four bodies, each a binary link, which interconnect via four revolute joints. The figure shows the four bodies, labeled A–D.



In a SimMechanics model, all bodies are rigid. They are idealizations in which internal strains always equal zero. True rigid bodies do not exist in nature but, under normal operating conditions, many engineered components behave as approximately rigid bodies —that is, with negligible deformation. In general, the rigid-body approximation provides accurate modeling results whenever the expected deformation is much smaller than the characteristic length of the modeled system.

## Rigid Body Properties

Solid properties determine the appearance and behavior of a rigid body. For example, the moments and products of inertia determine the angular acceleration of a free rigid body in response to an applied torque. In SimMechanics, solid properties fall into three groups —geometry, inertia, and graphic—each with group-specific parameters. The figure lists these properties.



To specify the solid properties of a rigid body, you use the blocks in the Body Elements library. The library contains three blocks, of which Solid is the most frequently used. This block enables you to specify all the solid properties of a rigid body in a single place. The remaining blocks, Inertia and Graphic, serve special cases, such as visualizing certain frames and modeling mass disturbances.

The table summarizes the primary purposes of the Body Elements blocks.

| Block | Purpose |
|---|---|
| `Solid` | Specify the solid properties—geometry, inertia, graphic—of a simple rigid body or of part of a compound rigid body. |
| `Inertia` | Specify the inertial properties of a mass element, such as a mass disturbance present in rigid bodies. |
| `Graphic` | Select a graphical icon to visualize any SimMechanics frame in a model. |

## Rigid Body Frames

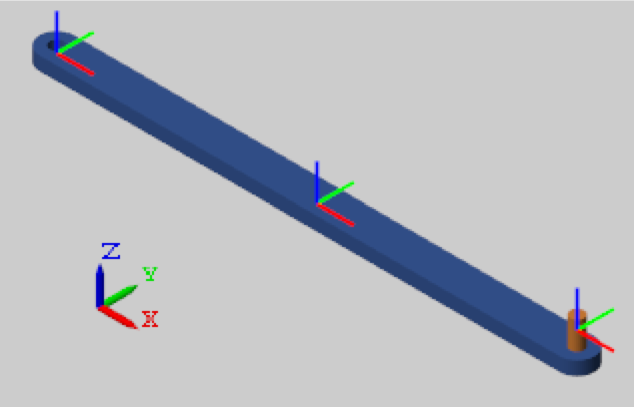
In SimMechanics, rigid bodies have frames, each identifying a position and orientation in 3-D space. These frames are important to the SimMechanics modeling workflow. They enable you to specify the correct position and orientation for each of the following tasks:

- Connect joints and constraints between rigid bodies. For example, you always connect a revolute joint between two frames in separate rigid bodies (or, alternatively, between a rigid body frame and the world frame).
- Apply forces and torques to or between rigid bodies. For example, you always apply an external force and torque to a single frame in a rigid body.
- Sense motion, forces, and torques between rigid bodies. For example, you always sense the relative position coordinates between two frames on different rigid bodies (or, alternatively, between a rigid body frame and the world frame).

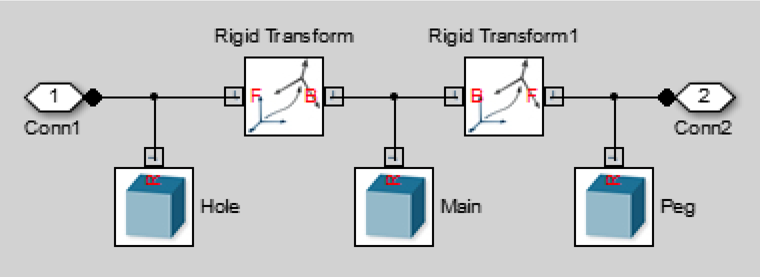
The Solid block, the main component of any body subsystem, provides a reference frame through frame port R. You can create additional frames in the Solid block dialog box using its frame-creation interface. This interface is accessible from the **Frames** area of the Solid block dialog box. The Solid block adds a frame port for every frame that you create.

Drawing a frame connection line between frame ports on different Solid blocks makes the port frames coincident in space. You can translate and rotate these frames relative to each other by adding a Rigid Transform block to the connection line. This block enables you to specify the pose of the follower frame relative to the base frame.

The figure shows an example of a rigid body subsystem in Mechanics Explorer. The rigid body is a binary link with three frames, each associated with a solid section of the link—hole, main, and peg. Rigid transforms specify the translational offset between each pair of frames.



In the binary-link block diagram, Rigid Transform blocks specify the translation transforms between the three frames. A total of two such blocks are needed, one between each pair of frames. The following figure shows the binary link block diagram.



For more information on this rigid body subsystem, see "Model Binary Link" on page 2-74

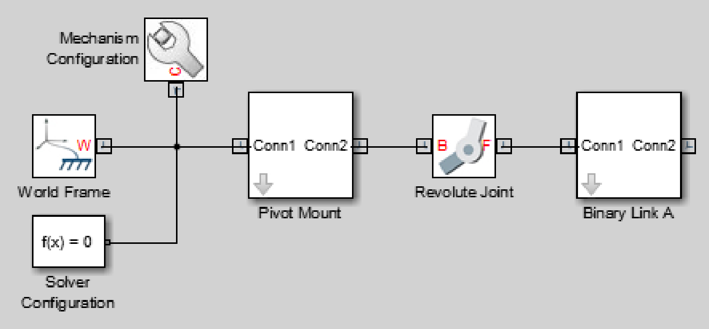
## Rigid Body Delimitation

In a SimMechanics model, a set of Solid and Rigid Transform blocks between two joint blocks or between one joint block and the World Frame block constitutes a rigid body. During simulation, SimMechanics software computes the center of mass for each such block subset. Gravitational Field blocks in your model, if any, then apply a gravitational force at the calculated centers of mass.

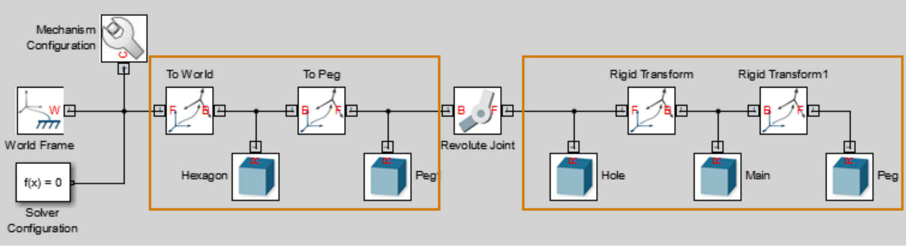
If you connect two halves of a rigid body using a Weld Joint block, the SimMechanics model treats the two halves as rigidly connected but independent rigid bodies. Any

Gravitational Field blocks in your model then exert a gravitational force at the center of mass of each half. This strategy enables you to account for gravitational torques acting on a rigid component, such as an asteroid orbiting the Sun.

The figure shows a simple-pendulum model. In this model, a subsystem block neatly encapsulates each rigid body. The model contains two rigid bodies: a pivot mount and a binary link. A joint block separates the mount and the link.



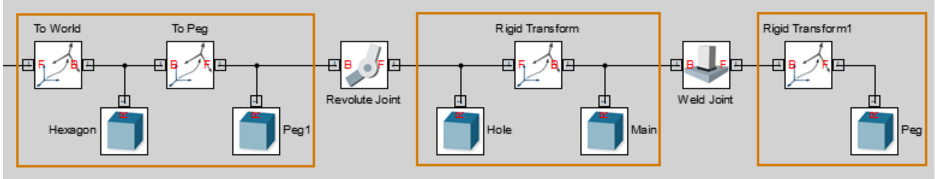
The following figure shows the same model without subsystem blocks. The model treats the blocks on either side of the Revolute Joint block as separate rigid bodies. The blocks to the left of the joint block represent the pivot mount, while the blocks to the right of the joint block represent the binary link.



If you connect a Weld Joint block between the Main and Rigid Transform1 blocks, the SimMechanics model recognizes three rigid bodies:

- Rigid body I to the left of the Revolute Joint block.
- Rigid body II between the Revolute Joint and Weld Joint blocks.
- Rigid body III to the right of the Weld Joint block.

The following figure shows the modified model with the Weld Joint block. The Mechanism Configuration, World Frame, and Solver Configuration blocks are omitted to conserve space.

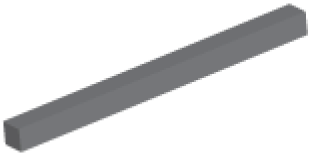


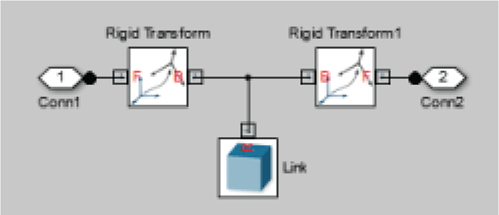
## Simple and Compound Rigid Bodies

Rigid bodies can be simple or compound. The difference between the two rigid body types lies in their complexity. Simple rigid bodies typically have basic shapes, uniform mass distributions, and a single color. Compound rigid bodies have more complex shapes and, occasionally, segmented mass distributions that require multiple Solid blocks to model.

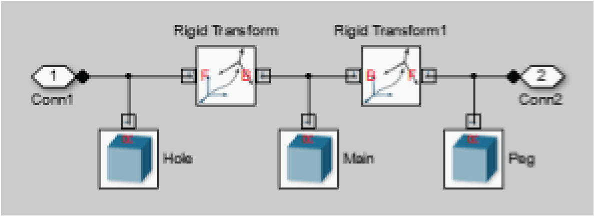
Consider a binary link, the basic component of mechanical linkages such as the double pendulum and the four-bar mechanism. Depending on the level of detail you want to incorporate in your model, you can treat the binary link as a simple rigid body or as a compound rigid body:

- Simple — Approximate the rigid-body geometry using a standard SimMechanics shape. For example, you can model the binary link using a brick shape with a uniform mass distribution and a single color. The tutorial “Model Simple Link” shows this approach.





- Compound — Model the rigid-body geometry accurately using multiple standard shapes. For example, you can model the binary-link main, hole, and peg sections using separate Solid blocks, each with its own shape and possibly its own inertial properties and color. The tutorial “Model Binary Link” on page 2-74 shows this approach.


Rigid Transform
Rigid Transform1
1
Conn1
F
B
B
F
2
Conn2
Hole
Main
Peg

# Solid Geometries

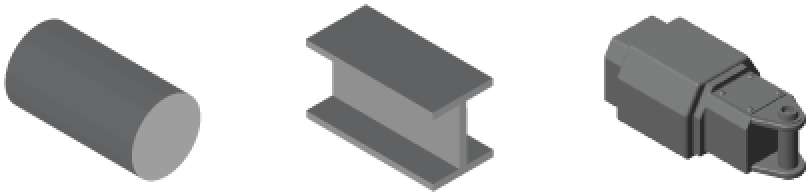
**In this section...**

## Geometry Essentials

Geometry is one of three solid properties that you can specify in a SimMechanics model. This property includes the shape of a rigid body and its size. For example, the geometry of Earth consists of a spherical shape and an approximate radius of 6,370 km. Specifying these parameters enables SimMechanics to perform two tasks:

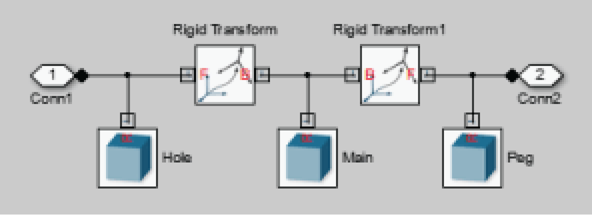
- Set the visual appearance of a rigid body, excluding color and lighting, in Mechanics Explorer.
- Automatically calculate the inertial properties of a rigid body, including the center of mass, the moments of inertia, and the products of inertia.

You specify solid geometry using the Solid block. Solid shapes that you can specify range from basic, such as a cylinder, to more sophisticated, such as a general extrusion. For intricate shapes, you can also load solid geometry from external files. The following figure shows examples of the different types of shapes.



## Compound Shapes

Solid shapes are versatile but, used individually, limited. To make the most of solid shapes, you must combine them using multiple Solid blocks. This approach yields a compound rigid body with an aggregate shape that can be more complex than a single solid shape would allow. The binary link shown in the following figure is one example.



In this example, three Solid blocks provide the geometries of the binary-link sections —the main body, the hole end, and the peg end. Two Rigid Transform blocks provide the spatial relationships between the three binary-link sections, including their relative positions and orientations. For more information, see “Model Binary Link” on page 2-74.

## Basic Shapes

Often, you can model a rigid body using basic shapes. These are common, simple shapes, such as sphere and cylinder, with parameterizations based on dimensions such as radius and length. Such shapes enable you to quickly model a rigid body approximately—e.g., for quick, proof-of-concept models. The table lists the basic shapes that you can model using the Solid block.

| Shape | Example | Parameters |
|---|---|---|
| Cylinder | | • Length<br>• Radius |
| Sphere | | • Radius |
| Brick | | • Length<br>• Width<br>• Thickness |
| Ellipsoid | | • Ellipsoid radii along semi-principal axes |

| Shape | Example | Parameters |
|---|---|---|
| Regular Extrusion | | • Length<br>• Outer radius<br>• Number of sides |

As an example, you can model a binary link using a brick shape. By using this approximation, you can quickly move on to more important aspects of the modeling workflow, such as multibody assembly. Once you have a working model, you can add detail to the rigid bodies, e.g., by treating the binary links as compound rigid bodies with more complex shapes.

For a tutorial showing how to model a binary link using a brick approximation, see "Model Simple Link".

## General Extrusion and Revolution Shapes

To model more complex rigid bodies, the Solid block provides general-extrusion and revolution shapes. These shapes enable you to model rigid bodies with arbitrary cross-sections, such as I-shaped beams and circular domes. The parameterizations for these shapes are more advanced and require detailed knowledge of the cross-section coordinates. The table summarizes the differences between two shapes.

| Shape | Description | Example | Parameters |
|---|---|---|---|
| General extrusion | Shape with a general cross-section that remains constant along an extrusion axis. | | • Cross-section coordinates<br>• Extrusion length |
| Revolution | Shape with a general cross-section that remains constant about a revolution axis. | | • Cross-section coordinates<br>• Revolution angle |

For tutorials showing how to model general-extrusion and revolution shapes, see:

- "Model I-Beam" on page 2-62
- "Model Dome" on page 2-56

## Imported Shapes

Instead of modeling a complex geometry manually, you can import it using one of two file formats: STL and STEP. Files in these formats specify the surface geometries of 3-D solids, albeit using different approaches. The table summarizes the key differences between the two formats.

| Geometry File Format | Description |
|---|---|
| STL (Standard Tesselation Language) | Provides the vertex coordinates and normal-vector components for each triangle segment in a tesselated 3-D surface. |
| STEP (Standard for Exchange of Product Data) | Provides the analytical curves that describe a 3-D surface. STEP files enable the Solid block to automatically compute inertial properties. |

Note that these files do not specify graphic or inertial properties. You must specify those properties separately. To generate the STL or STEP files, you must use an external application, such as a CAD tool.

# Solid and Body Visualization

**In this section...**

## Visualization Essentials

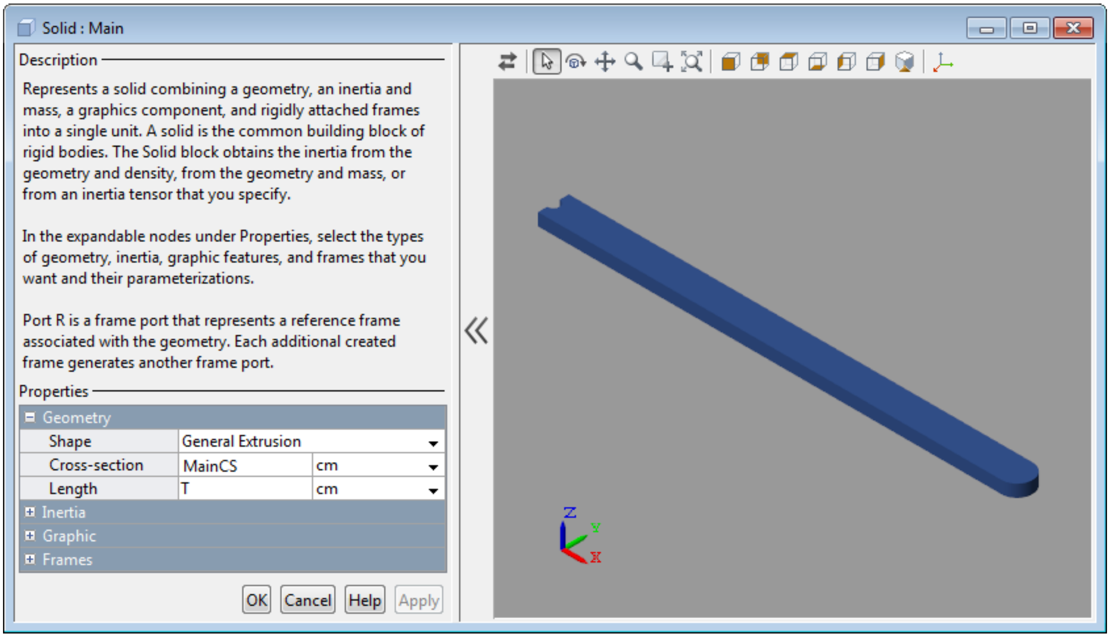
Visualization is an integral part of the SimMechanics modeling workflow. You can render a model graphically and examine its bodies and their interconnections—for example, to ensure that solid geometries, colors, and spatial relationships are what you expect. To visualize bodies and the solids that comprise them, you use one of the following:

- Solid block visualization pane — Visualize individual solids and their reference port frames. Use the Solid block visualization pane to examine solid geometries and colors without having to update or simulate the block diagram, a Mechanics Explorer requirement.
- Mechanics Explorer — Visualize multibody systems, the bodies that comprise the systems, and the solids that comprise the bodies. Use Mechanics Explorer if you must visualize more than one solid at a time, or if the spatial relationships between different frames are important to you.

## Solid Visualization

To visualize individual solids—for example, the binary link main section shown in the "Model Binary Link" on page 2-74 tutorial—you use the embedded visualization pane in the `Solid` block dialog box. This pane enables you to change Solid block parameters and examine their effect on solid geometry and color near instantly without having to open a separate window. Above the pane, the Solid block includes a visualization toolstrip that you can use to:

- Update the visualization pane after a parameter change.
- Rotate, pan, and zoom the solid view.
- Select a standard view, such as Isometric, Front, Top, or Right.
- Toggle visualization on and off for the solid reference port frame. The position of the frame varies for shapes such as `General Extrusion` and `Revolution`, making frame visualization useful when deciding how to connect your solid in a model.



Solid : Main

Description

Represents a solid combining a geometry, an inertia and mass, a graphics component, and rigidly attached frames into a single unit. A solid is the common building block of rigid bodies. The Solid block obtains the inertia from the geometry and density, from the geometry and mass, or from an inertia tensor that you specify.

In the expandable nodes under Properties, select the types of geometry, inertia, graphic features, and frames that you want and their parameterizations.

Port R is a frame port that represents a reference frame associated with the geometry. Each additional created frame generates another frame port.

Properties

⊟ Geometry

| | | |
|---|---|---|
| Shape | General Extrusion | |
| Cross-section | MainCS | cm |
| Length | T | cm |

⊞ Inertia

⊞ Graphic

⊞ Frames

## Solid Block Visualization Pane

You do not need to update or simulate the block diagram—which means that you can visualize a solid before the block diagram is complete. For example, you can visualize a solid before connecting a Simscape `Solver Configuration` block to the SimMechanics portion of the physical network. You can also visualize a solid before connecting the Solid block to any other blocks on the model canvas.

You can visualize a solid even if you parameterize its properties in terms of MATLAB variables—for example, entering a variable `T` in the **Length** parameter of a `General Extrusion` shape. However, you must assign numerical values to your MATLAB variables before the Solid block can render your solid. You can do this in a Simulink Subsystem mask, or in the model workspace through Model Explorer. For examples, see:

- "Model Cone" on page 2-50
- "Model I-Beam" on page 2-62

Each time you change a Solid block parameter, you must select the Update Visualization button ⇄—or press **F5**—in order to refresh the solid view in the visualization pane.

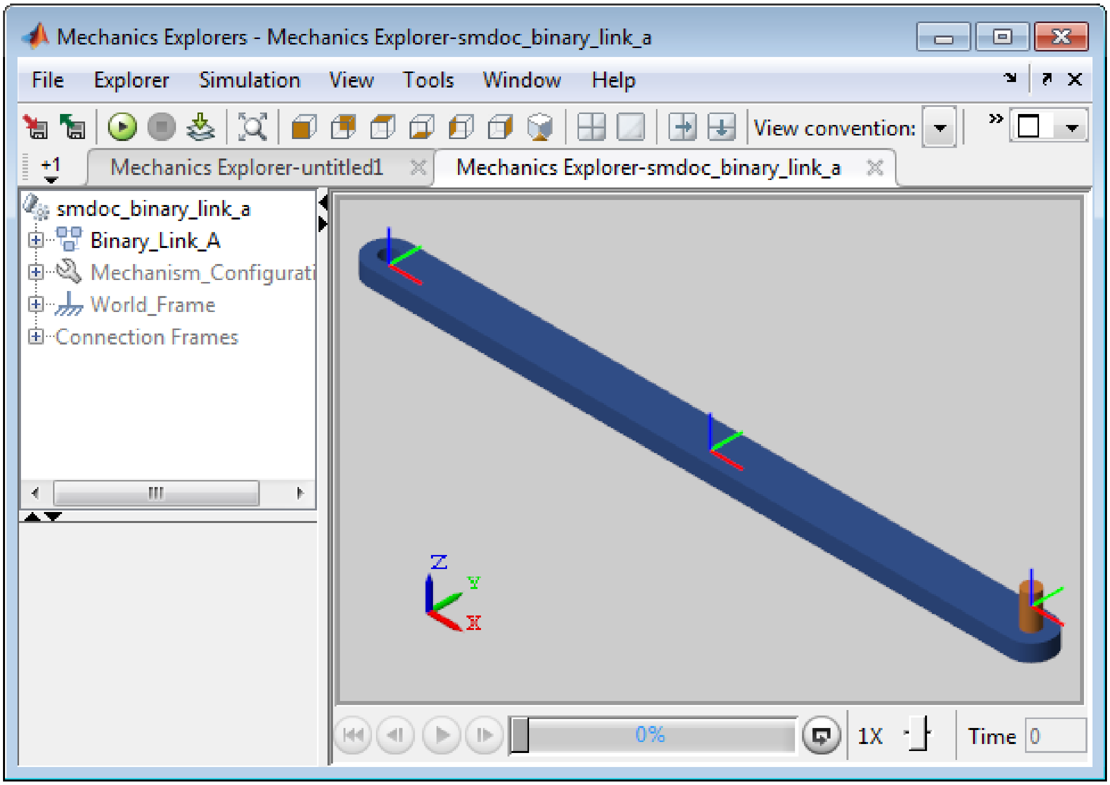
The Update Visualization button does not apply your changes to the model, giving you a chance to verify them before you commit to them. To apply you parameter changes to the model, you must select **Apply**.

## Body Visualization

To visualize a body containing multiple solids—for example, the complete binary link shown in the "Model Binary Link" on page 2-74 tutorial—you use Mechanics Explorer. This visualization utility displays the entire contents of your model, enabling you to examine not only solid geometries and colors, but also how solids connect in compound bodies and how bodies connect in multibody systems.

Mechanics Explorer provides its own visualization toolstrip—similar to, but more expansive than, the Solid block visualization toolstrip. You can use the toolstrip to:

- Start simulation or update the block diagram.
- Rotate, pan, and zoom the model view.
- Select a standard view, such as Isometric, Front, Top, or Right.
- Toggle visualization on and off for the frames and body centers of mass in your model.
- Split the visualization pane into tiles, each with a different model view.
- Change the view convention—for example, so the World frame Z axis points down on your screen.
- Change the background color of the visualization pane.



## Mechanics Explorer Visualization Utility

To visualize a body or multibody system in Mechanics Explorer, you must update the block diagram (**Simulation** > **Update Diagram**) or start simulation (**Simulation** > **Run**). As a result, you must also ensure that your block diagram is valid. In particular, you must connect a Simscape `Solver Configuration` block to the SimMechanics portion of the model.

By default, when you update or simulate a model, Mechanics Explorer opens automatically with a 3-D view of your model. Each time you change the block diagram or a block parameter, you must update or simulate the model for Mechanics Explorer to reflect your changes. For examples, see:

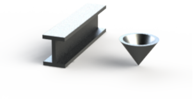
- "Model Binary Link" on page 2-74
- "Model Pivot Mount" on page 2-88

# Revolution and General Extrusion Shapes

**In this section...**

## Shape Cross Sections

Using `General Extrusion` and `Revolution` shapes, you can model solids with arbitrary cross sections that remain constant along or about some axis. I-beam and cone shapes are two examples. These shapes differ from simple shapes such as `Brick` and `Sphere` in their requirement that you specify the cross-section coordinates explicitly. You specify these coordinates in the Solid block dialog box in a matrix format.



**Extrusion and Revolution Shapes**

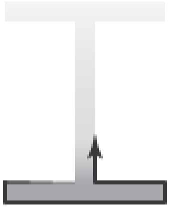
## Coordinate Matrices

Coordinate matrices are M × 2 in size. Each row corresponds to a point on the cross-section outline and each column corresponds to a coordinate on the cross-section plane. If you specify a cross-section shape using ten points, the resulting coordinate matrix is 10 × 2 in size.

The cross-section plane differs between `General Extrusion` and `Revolution` shapes. That plane is the XY plane for general extrusion shapes and the XZ plane for revolution shapes. Cross-section coordinates are therefore [X, Y] pairs for general extrusion shapes and [X, Z] pairs for revolution shapes.

The Solid block generates the cross-section shape from your coordinate matrix by connecting each point to the next with a straight line. If the last point is different from the first point, the block connects the two in order to close the cross-section outline.

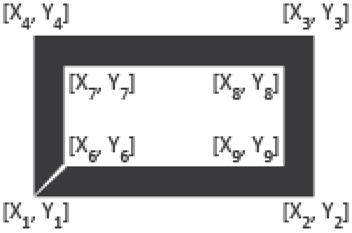
The cross-section outline divides the solid region of your shape from the empty region outside of it. To decide where the solid region is, the Solid block uses a special rule: looking from one point to the next, the solid region must lie to the left of the connecting line. The figure shows the application of this rule to an I-beam cross-section shape.



## Hollow Cross Sections

Bodies are often hollow. A box beam is one example. You can model such bodies using `General Extrusion` and `Revolution` shapes. As before, you must specify the coordinate matrix as a continuous path so that, looking from one coordinate pair to the next, the solid region lies to your left and the empty region lies to your right.

However, the path must now traverse not only the outer outline of the cross-section, but also the outline of its hollow region. To do this, the path must cut across the solid portion of the cross section. The figure shows such a cut.



The coordinate matrix for the outer cross-section outline is

$$outerCS = \left[X_1, Y_1; X_2, Y_2; X_3, Y_3; X_4, Y_4; X_1, Y_1\right].$$

Similarly, the coordinate matrix for the inner cross-section outline is

$$innerCS = \left[X_6, Y_6; X_7 Y_7; X_8, Y_8; X_9, Y_9; X_6, Y_6\right].$$

The complete coordinate matrix is then

$$CS = [outerCS; innerCS].$$

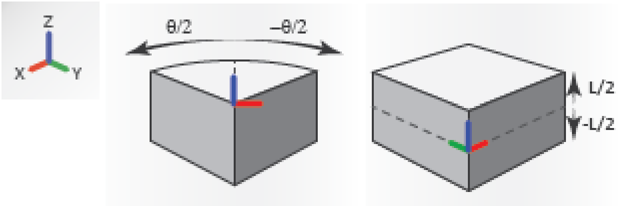
Note that *outerCS* traces the outer profile counterclockwise, while *innerCS* traces the inner profile clockwise. You must reverse the order of the coordinates as you transition between the two outlines in order to keep the solid region to the left of the cross-section line.

Note also that, taken individually, *outerCS* and *innerCS* each trace a closed outline. You must close each outline by ending its coordinate matrix on the first coordinate pair for that outline. The Solid block automatically closes the overall cross-section profile by connecting the last coordinate pair to the first. In doing so, the Solid block traces the first cut at the same location and in reverse, ensuring that the cut thickness is zero.

## 3-D Shape Generation

The Solid block produces a 3-D shape from a cross-section outline by sweeping the outline along or about the reference frame Z axis. The sweep amount is the same in the positive and negative directions of the Z axis. The figure shows the directions and amounts of sweep for a revolution shape and a general extrusion shape:

- For a general extrusion of thickness $L$, the block sweeps the cross-section outline by $L/2$ along the positive and negative directions of the Z axis.
- For a revolution with a sweep angle of $\theta$, the block sweeps the cross-section outline by $\theta/2$ about the positive and negative directions of the Z axis.



The reference port frame of revolution and general extrusion solids has its origin at the [0, 0] coordinate. This coordinate lies in the cross-section plane for a general extrusion solid and in the revolution axis for a revolution solid.

# Solid Inertia

**In this section...**

## Inertial Properties

Inertia is the resistance of matter to acceleration due to applied forces and torques. The inertial properties of a body include its mass and inertia tensor—a symmetric 3×3 matrix that contains the moments and products of inertia. Mass resists translational acceleration while the moments and products of inertia resist rotational acceleration.

Among the solid properties of a model, the inertial properties have the greatest impact on multibody dynamics. Those that you must specify depend on the type of inertia you are modeling—a point mass or a body with a 3-D mass distribution. They include one or more of the following:

- Mass or density
- Center of mass
- Moments of inertia
- Products of inertia

In a SimMechanics model, these properties are time-invariant. Rigid bodies cannot gain or lose mass nor can they deform in response to an applied force or torque. The mass distribution of a body—and therefore its inertia tensor and center of mass—remain constant throughout simulation.

## Blocks with Inertia

You can model inertia using the following blocks:

- `Solid` — Model a complete solid element with geometry, inertia, and color. This block can automatically compute the moments of inertia, products of inertia, and center of mass based on the solid geometry and mass or mass density. During simulation, Mechanics Explorer renders the solid using the geometry and color specified.
- `Inertia` — Model only the inertial properties of a solid element. You must specify the moments of inertia, products of inertia, and center of mass explicitly. During simulation, Mechanics Explorer identifies the center of mass using the inertia icon ⏢.

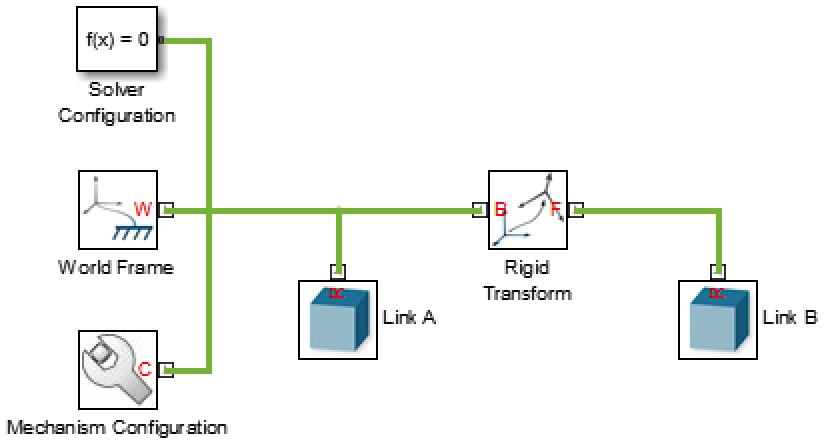
## Inertia in a Model

To add a Solid or Inertia block to your model, connect its frame port to another frame entity in the model. Frame entities include frame lines, nodes, and ports. The frame entity to which you connect the block determines the position and orientation of the inertia within the model. See "Representing Frames" on page 1-6.

The Solid and Inertia blocks each provide a reference frame port. The Solid block enables you to create additional frames, each of which adds a new frame port to the block. You can use any of these frames to connect a Solid block in a model.

The figure shows an example. The model shown contains two Solid blocks, labeled Link A and Link B. The reference frame port of Link A connects directly to the World Frame block. Its reference frame is therefore coincident with the World frame.

The reference frame port of Link B connects to the follower frame port of the Rigid Transform block. This block applies a spatial transform between the World frame the reference frame of the Link B block. The spatial transform translates and/or rotates the two frames relative to each other.



For examples showing how to position solid elements in a model, see:

- "Model Two-Hole Binary Link" on page 2-82
- "Model Pivot Mount" on page 2-88

## Inertia Parameterizations

Once you have connected the Solid or Inertia blocks in a model, you must specify their inertial parameters. These depend on the inertia parameterization that you select. The blocks provide three optional parameterizations:

- `Calculate from Geometry` — Specify mass or density. The Solid block automatically computes the remaining inertial properties based on the solid geometry. Only the Solid block provides this parameterization.
- `Point Mass` — Specify the mass and ignore the remaining inertial parameters. The inertia behaves as point mass with no rotational inertia.
- `Custom` — Manually specify every inertial parameter. You must obtain each parameter through direct calculation or from an external modeling platform.

## Custom Inertia

If you select the `Custom Inertia` parameterization, you must specify the moments of inertia, products of inertia, and center of mass explicitly. These parameters depend closely on the reference frame used in their calculations, so you must ensure that frame matches the one used in SimMechanics:

- Moments and products of inertia — Enter with respect to a frame parallel to the reference port frame but with origin at the center of mass.
- Center of mass — Enter with respect to the reference port frame.

Consider the main section of the binary link in “Model Binary Link” on page 2-74. You model this solid using a single solid block with a `General Extrusion` shape. As described in the Solid block documentation, the reference port frame for a general extrusion has its origin in the XY plane at the [0,0] cross-section coordinate.

The figure shows the solid reference port frame, labeled R. The center-of-mass coordinates must be with respect to this frame. The moments and products of inertia must be with respect to a parallel frame offset so that its origin coincides with the center of mass. This frame is virtual , as it does not correspond to any frame port, line, or node in the model. It is labeled R* in the figure.

## Moments and Products of Inertia

You can extract the moments and products of inertia directly from the inertia tensor. This tensor is symmetric: elements with reciprocal indices have the same magnitude. That is:

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

This symmetry reduces the number of unique tensor elements to six—three moments of inertia and three products of inertia. The complete inertia tensor has the form:

$$\begin{bmatrix} I_{xx} & I_{xy} & I_{zx} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{zx} & I_{yz} & I_{zz} \end{bmatrix}$$

The moments of inertia are the diagonal elements:

$$\begin{bmatrix} I_{xx} & & \\ & I_{yy} & \\ & & I_{zz} \end{bmatrix}$$

SimMechanics defines these elements as follows:

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

The products of inertia are the unique off-diagonal elements, each of which appears in the inertia tensor twice:

$$\begin{bmatrix} & I_{xy} & I_{zx} \\ I_{xy} & & I_{yz} \\ I_{zx} & I_{yz} & \end{bmatrix}$$

SimMechanics defines these elements as follows:

- $$I_{\mathrm{yz}} = -\int_V yz\, dm$$
- $$I_{\mathrm{zx}} = -\int_V zx\, dm$$
- $$I_{\mathrm{xy}} = -\int_V xy\, dm$$

The inertia tensor is simplest when it is diagonal. Such a tensor provides the moments of inertia about the principal axes of the solid or inertia element—known as the principal moments of inertia. The products of inertia reduce to zero:

$$\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

For more information, see the `Solid` and `Inertia` block reference pages.

## Complex Inertias

Bodies often comprise different materials, have complex shapes, or contain material imperfections that alter their centers of mass and principal axes. One example is an imbalanced automobile wheel after driving through a pothole. You can model complex inertias such as these using two approaches:

- Use a divide-and-conquer approach. Break up the complex solid or inertia into simpler chunks and model each using a separate Solid or Inertia block. The resulting set of Solid and Inertia blocks constitute a compound inertia. You use a similar approach to model complex geometries, such as the binary link geometry in "Model Binary Link" on page 2-74.
- Manually specify the complete inertial properties using a single Solid or Inertia block with the inertia parameterization set to `Custom`. You must obtain the moments of inertia, products of inertia, and center of mass through direct calculation, from another modeling platform, or from another external source.

For bodies with complex shapes but uniform mass distributions, you can also import a STEP file containing the solid geometry and set the inertia parameterization to `Calculate from Geometry`.

# Specify Custom Inertia

**In this section...**

## Custom Inertia

You can specify the inertia tensor and center of mass of a solid explicitly. To do this, in the Solid or Inertia block, you set the inertia parameterization to Custom. This option exposes additional fields so that you can enter the moments of inertia, products of inertia, and center of mass.

Before entering the inertia parameters, you must ensure that they are defined with respect to the correct frame. This frame typically coincides with the reference port frame for standard shapes such as Sphere, Brick, and Cylinder, but not for the more advanced Revolution and General Extrusion shapes or imported shapes.

This tutorial shows how you can specify the inertial parameters of a solid explicitly, clarifying along the way the frames that the moments of inertia, products of inertia, and center of mass must refer to.

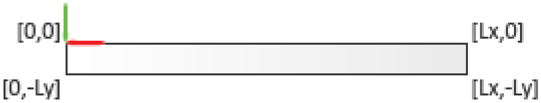
## Model Overview

The model in this tutorial is simple. It contains a single Solid block through which you specify the inertial parameters of your solid. This block connects to the World frame through a Revolute Joint block, providing it one rotational degree of freedom. You examine the effect of the inertial parameters by simulating the angular motion in the model.

The solid that you model has a brick shape. To illustrate the challenges associated with the Custom Inertia parameterization, the tutorial represents the brick shape as a general extrusion. The result is a brick shape with the reference port frame origin located at the [0,0] cross–section coordinates—which in this tutorial do not coincide with the solid center of mass.

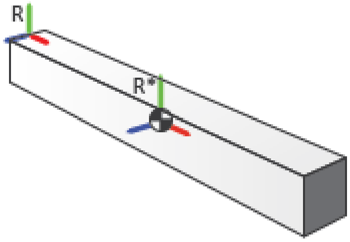
The figure shows the cross section of the general extrusion and its coordinates. The [0,0] coordinates, and therefore the reference port frame origin, coincide with the upper left vertex in the cross section. Variables *Lx*, *Ly*, and *Lz* refer to the length, width, and thickness of the solid, respectively. Their values are:

- $Lx = 20\,\text{cm}$
- $Ly = 1\,\text{cm}$
- $Lz = 1\,\text{cm}$



You specify the center of mass, but not the moments or products of inertia, with respect to the reference port frame. To specify the moments and products of inertia, you must use a different frame—one whose axes are parallel to the reference port frame axes but whose origin is coincident with the solid center of mass.

The figure shows the extruded solid, its reference port frame (R), and the parallel frame with origin at the center of mass (R*). This frame does not correspond to any frame entity in the model. It is said to be virtual.



## Inertia Parameters

To fully specify the inertia of a solid, you must specify four parameters:

- Mass
- Center of mass
- Moments of inertia
- Products of inertia

The mass of the solid is the product of its density and volume. For an aluminum solid, the density is $\rho = 2.7\ {}^{\text{g}}\!/\!_{\text{cm}^3}$. The volume is the product of the side lengths $V = Lx\,Ly\,Lz = 20\,\text{cm}^3$. The solid mass is therefore:

$$M = \rho V = 54\ \text{g}.$$

Assuming the mass distribution is uniform, the center of mass must coincide with the center of geometry, which in frame R has the coordinates:

$$CM = \left[\frac{Lx}{2}, -\frac{Ly}{2}, 0\right] = [10, -0.5, 0]\,\text{cm}.$$

The axes of frame R* align with the principal axes of the solid. Taken with respect to this frame, the moments of inertia are:

- $I_{\text{xx}} = \dfrac{M}{12}\left(Ly^2 + Lz^2\right) = 9\,\text{g}\,\text{cm}^2$
- $I_{\text{yy}} = \dfrac{M}{12}\left(Lz^2 + Lx^2\right) = 1804.5\,\text{g}\,\text{cm}^2$
- $I_{\text{zz}} = \dfrac{M}{12}\left(Lx^2 + Ly^2\right) = 1804.5\,\text{g}\,\text{cm}^2$

The axes of frame R* are the solid principal axes. The products of inertia are therefore zero:

- $I_{yz} = 0$
- $I_{zx} = 0$

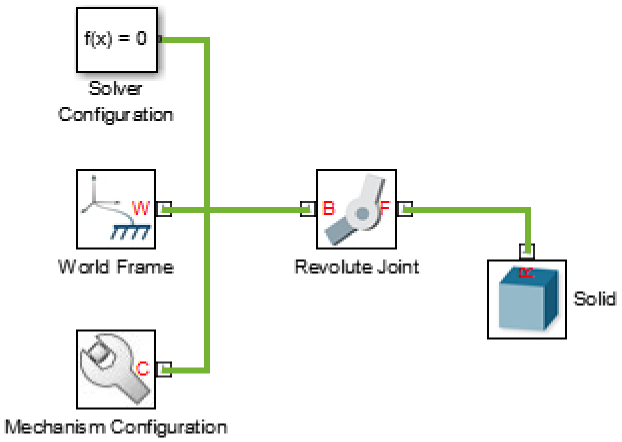
- $I_{xy} = 0$

## Build Model

1 At the MATLAB command prompt, enter `smnew`. SimMechanics opens a new model with some commonly used blocks. The SimMechanics library also opens up.

2 From the Joints library, drag a `Revolute Joint` block.

3 Connect the blocks as shown in the figure. You can remove any unused blocks.



4 In the Mechanism Configuration block, set **Gravity** to `[0 -9.81 0]`. The new gravity vector is perpendicular to the revolute joint axis, ensuring that the solid you modeled oscillates due to gravity when displaced from its equilibrium position.

## Specify Inertia

1 In the Solid block dialog box, under **Geometry**, specify the following parameters. These parameters define the shape and size of the solid.

| Parameter | Value | Units |
|---|---|---|
| **Shape** | `General Extrusion` | |

| Parameter | Value | Units |
|---|---|---|
| **Cross-Section** | `[0 0; 0 -1; 20 -1; 20 0]` | `cm` |
| **Length** | `1` | `cm` |

2 Under **Inertia**, specify the following parameters. These are the inertia parameters that you calculated at the beginning of the tutorial.

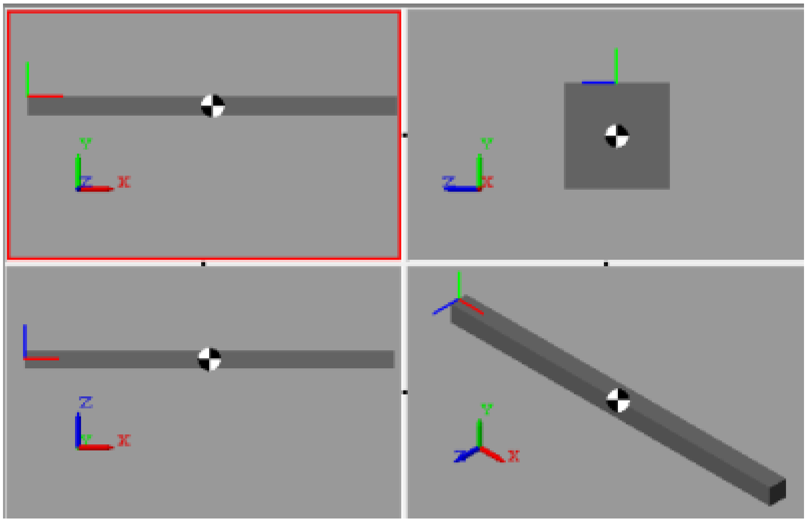
| Parameter | Value | Units |
|---|---|---|
| **Type** | `Custom` | |
| **Mass** | `54` | `g` |
| **Center of mass** | `[10,-0.5,0]` | `cm` |
| **Moments of Inertia** | `[9, 1804.5, 1804.5]` | `g*cm^2` |
| **Products of Inertia** | `[0, 0, 0]` | `g*cm^2` |

3 Update the block diagram—for example, by selecting **Simulation** > **Update Diagram**. Mechanics Explorer opens with a static 3-D view of the model in its initial configuration.

4 In the Mechanics Explorer menu bar, select:

- **View** > **Layout** > **Four Standard Views**. This option splits the visualization window into four panes, each with a different view.
- **View** > **Show Frames**. This option exposes all the frames in the model.
- **View** > **Show COMs**. This option exposes the center of mass for each rigid body in the model.

The figure shows the resulting view in Mechanics Explorer.



5 Examine the solid reference frame and center of mass. Verify that the center of mass appears at the geometric center of the solid and that the solid reference frame origin coincides with the upper left corner of the solid cross section.

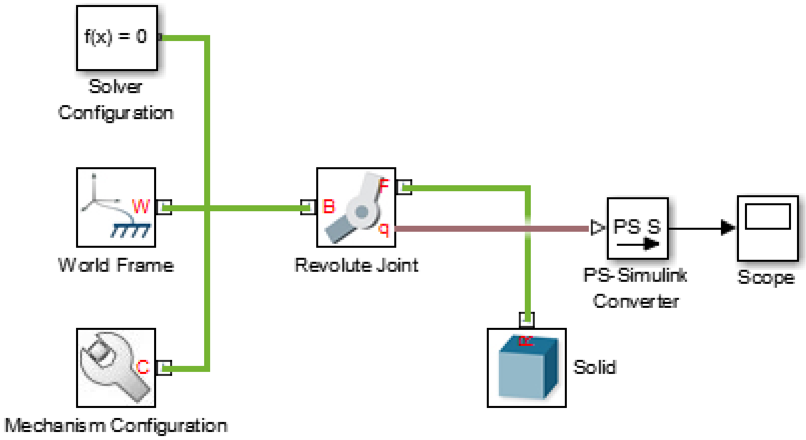
## Add Motion Sensing

1 In the Revolute Joint block dialog box, under **Sensing**, select **Position** and click **OK**. The block exposes the physical signal output port q.

2 From the Simscape Utilities library, drag a `PS-Simulink Converter` block. This block enables you to convert the physical signal output from the joint block into a Simulink signal.

3 From the Simulink Sinks library, drag a `Scope` block. This block enables you to plot the joint position output as a function of time.

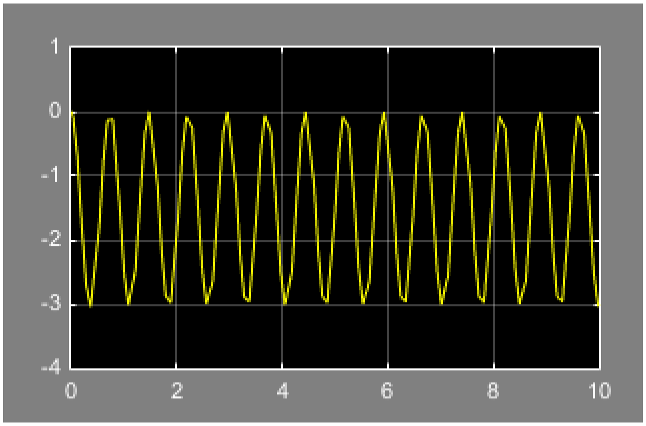
4 Connect the blocks as shown in the figure.



## Run Simulation

Simulate the model—for example, by selecting **Simulation** > **Run**. Mechanics Explorer plays an animation of the model. To ensure that the gravity vector aligns vertically on your screen, in the Mechanics Explorer toolstrip, set **View Convention** to `Y up (XY Front)`.

Double-click the Scope block and examine the oscillation period of the solid. The figure shows the resulting plot.



Try changing the inertia parameterization in the Solid block to `Calculate from Geometry` and simulate the model once again. Compare the plot from the second simulation to the first. The results are identical.

# Interactively Create Solid Frames

**In this section...**

## Solid Frames

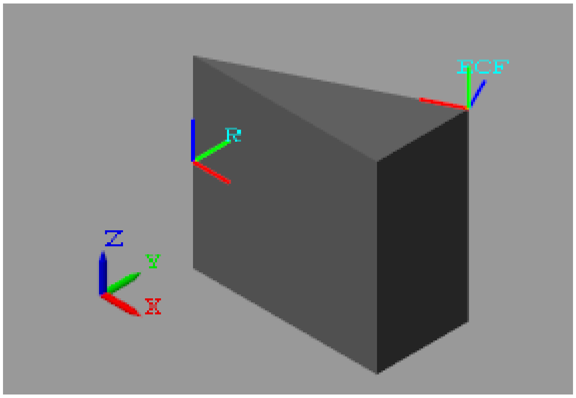
By default, the Solid block provides only a reference frame port, labeled R. In simple shapes, such as bricks, cylinders, and spheres, the reference frame origin coincides with the solid center of mass. In more complex shapes, such as extrusions and revolutions, the reference frame can be anywhere relative to the 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.

In this example, you create a new frame in 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.

The figure shows the solid shape, its default reference frame (R), and the new frame that you create (ECF).



## 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 ⇄.

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.

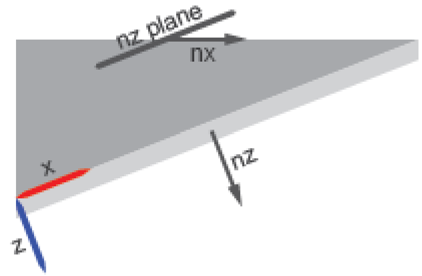
Frames that you define by geometric features are specific to the shapes the features belong to. If you make the frame origin coincident with the vertex of a brick, the new frame is valid only for that particular brick shape. If you change shapes, you must edit or delete the new frame, as the geometric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

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

- Frame origin
- Primary axis
- Secondary axis

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.



## Model Solid Shape

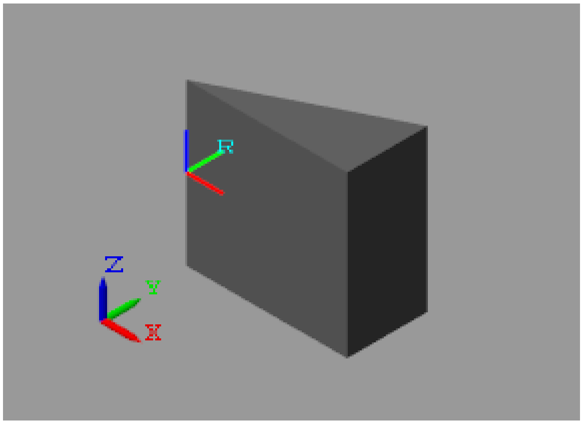
1 From the Body Elements library, add one Solid block to a new model. The Solid block provides its own visualization utility. You do not need to update the block diagram to visualize the solid shape or its frames.

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]` |

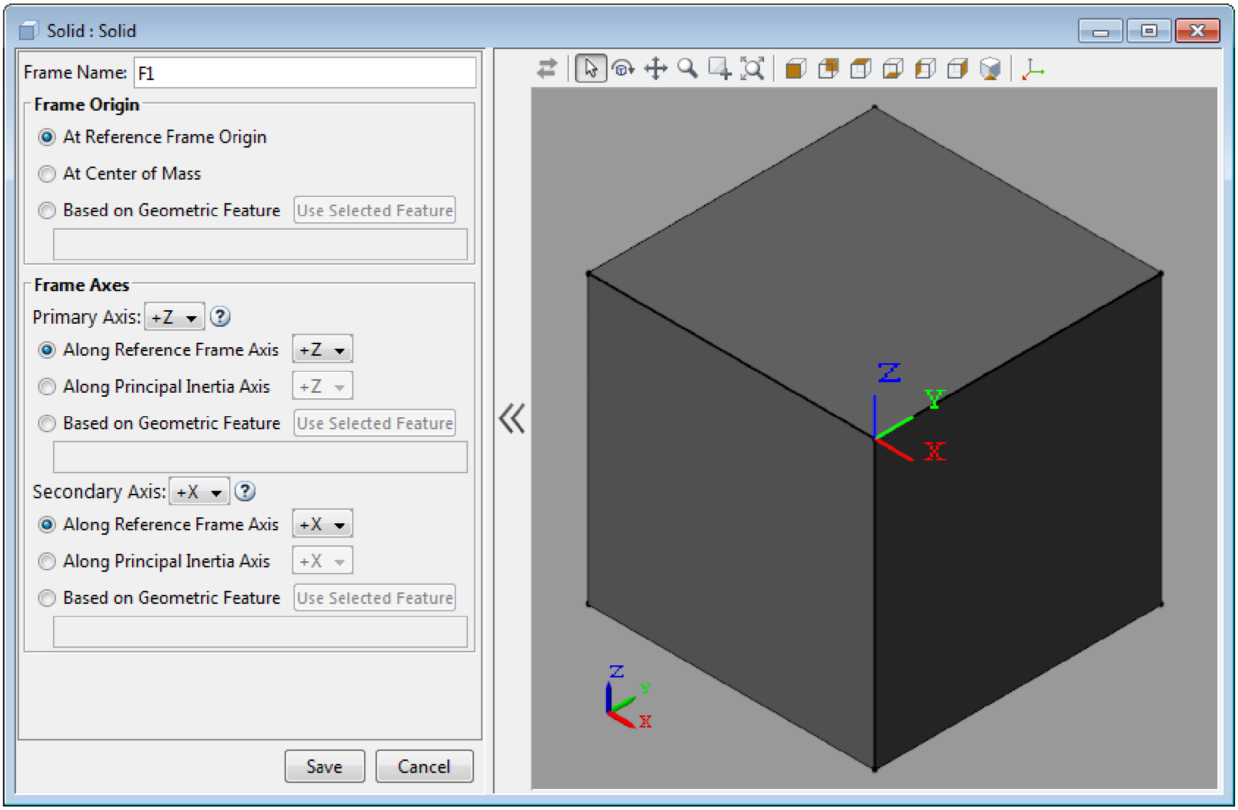
3 In the visualization toolstrip, select the **Update Visualization** button ⇄. The visualization pane updates with the three-sided extrusion that you specified.

4 Select the **Toggle 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 New Frame

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



Solid : Solid

Frame Name: F1

Frame Origin

● At Reference Frame Origin

○ At Center of Mass

○ Based on Geometric Feature [Use Selected Feature]

Frame Axes

Primary Axis: +Z

● Along Reference Frame Axis +Z

○ Along Principal Inertia Axis +Z

○ Based on Geometric Feature [Use Selected Feature]

Secondary Axis: +X

● Along Reference Frame Axis +X

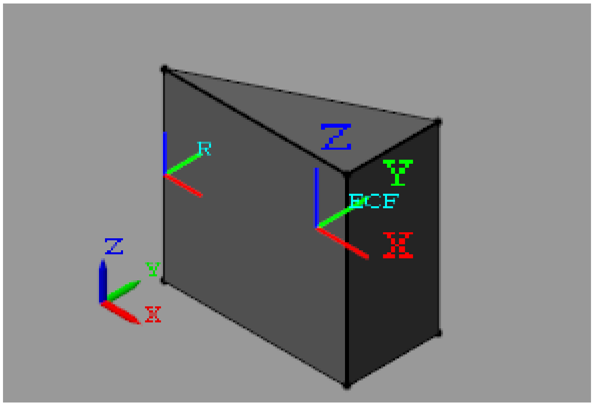
○ Along Principal Inertia Axis +X

○ Based on Geometric Feature [Use Selected Feature]

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 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.

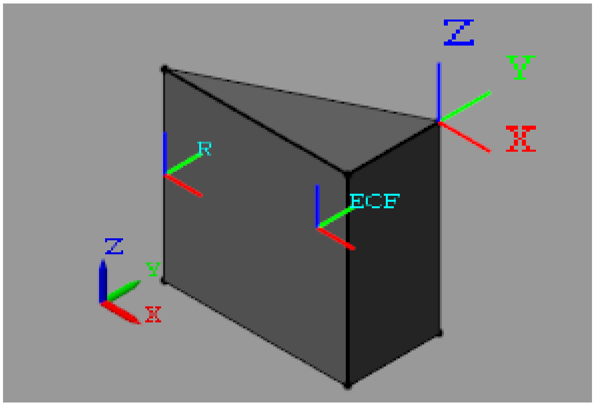


Experiment with other frame origin locations. Define the origin location using one of the extrusion vertices.

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.

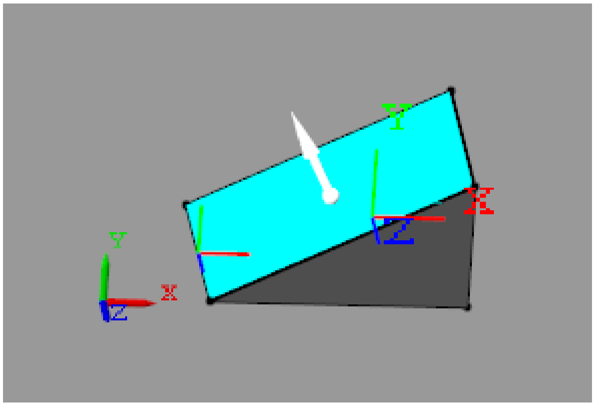


## Specify Primary Axis

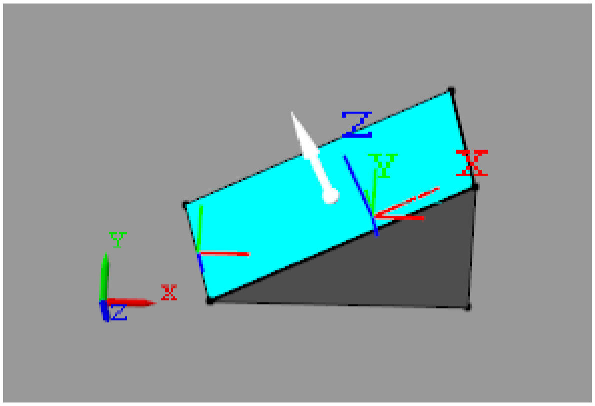
The primary axis constrains the remaining two axes to lie on its normal plane. In this sense, the primary axis plays the dominant role in setting the orientation of the frame. Make the primary axis normal to the surface that contains the cross-section hypotenuse:

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.



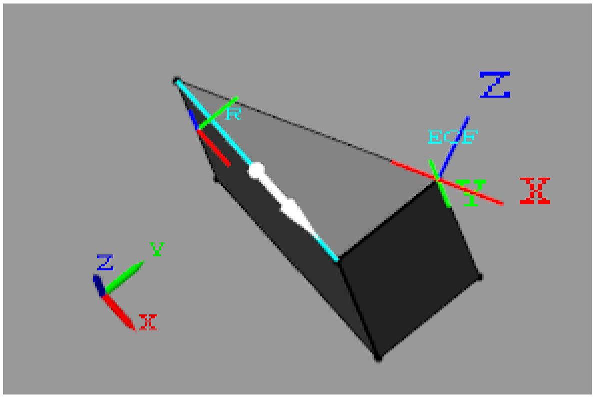
## Specify Secondary Axis

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.

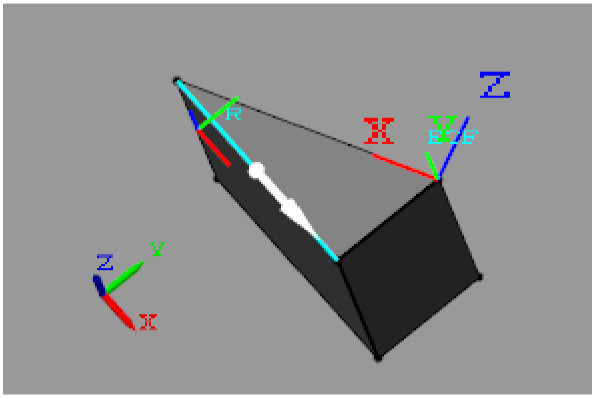
2 In the **Frame Axes** area, under **Secondary Axis**, select **Based on Geometric Feature**.

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.

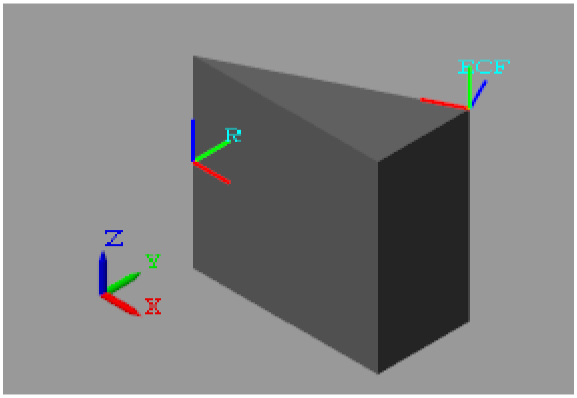


## Save New Frame

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.



# Solid Color

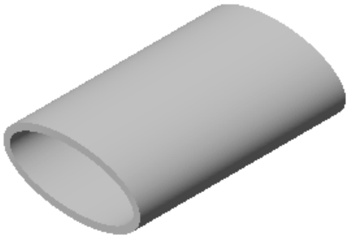
| In this section... |
|---|
| "Basic Graphic Parameters" on page 2-46 |
| "Advanced Graphic Parameters" on page 2-47 |

To make the most of the visualization capability of Mechanics Explorer, the `Solid` block provides two parameterizations that you can use to specify the graphic appearance of a solid: `Simple` and `Advanced`. The two parameterizations accept material color and opacity parameters as input. Light source parameters are fixed for all models. The table provides a comparison of the input parameters present in each graphic parameterization.

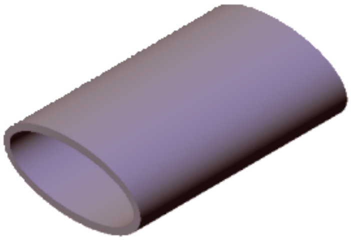
| Graphic Parameter | `Simple` | `Advanced` |
|---|---|---|
| **Diffuse Color** | ✓ | ✓ |
| **Ambient Color** | ✗ | ✓ |
| **Specular Color** | ✗ | ✓ |
| **Emissive Color** | ✗ | ✓ |
| **Opacity** | ✓ | ✓ |
| **Shininess** | ✗ | ✓ |

As an example, the figure shows two identical elliptical extrusions, one based on `Simple` and the other on `Advanced` graphic parameterizations. In both cases, the extrusion is completely opaque with a gray diffuse color. The advanced version adds to the solid a set of blue highlights, through the use of specular color, and a red ambient hue, through the use of ambient color.

| Color Parameter | `Simple` | `Advanced` |
|---|---|---|
| **Diffuse Color** | `[0.8 0.8 0.8]` | `[0.8 0.8 0.8 1.0]` |
| **Ambient Color** | — | `[0.1 0.05 0.05 1.0]` |
| **Specular Color** | — | `[0 0 1.0 1.0]` |



Solid with *Simple* color parameterization.



Solid with *Advanced* color parameterization.

The material colors — diffuse, ambient, specular, and emissive — form the core of the graphical representation of a solid in SimMechanics. You can specify the material colors in terms of RGB or RGBA color vectors.

## Basic Graphic Parameters

Both `Simple` and `Advanced` graphic parameterizations require you to specify the diffuse color and opacity of the solid. Together, these two parameters represent the graphical core of a SimMechanics solid. The way in which you specify the parameters differs slightly between the two parameterizations, but the meaning of each parameter remains the same.

### Diffuse Color

Apparent color of a rough solid surface exposed to direct white light. Diffuse light scatters equally in all directions according to Lambert's law, causing the intensity and color of the scattered light to appear the same from all angles. The diffuse color normally provides the dominant contribution to the color of a solid surface. In most cases, you can think of the diffuse color as the "true color" of a solid surface.

| Parameterization | Parameter Name Used | Specification |
|---|---|---|
| `Simple` | **Color** | [R G B] vector |
| `Advanced` | **Diffuse Color** | [R G B A] vector |

The figure shows the effect of varying the diffuse color of a solid. The array of spheres have identical graphical properties, with the exception of **Diffuse Color**. The RGBA color vector of the diffuse color progresses from [1 1 1], at the left corner, to [0.85 0.45 0], at the right corner. A gray ambient color gives the solid a darker appearance.

### Opacity

The opacity is the degree to which a solid blocks light from passing through. A completely opaque solid blocks all light penetration through the solid. The opposite of a completely opaque solid is a transparent solid, which allows all light to pass through. You can reduce the opacity of a solid in order to improve the visibility of other solids otherwise blocked from view.

| Parameterization | Parameter Name Used | Specification |
|---|---|---|
| `Simple` | **Opacity** | Scalar number (0–1) |
| `Advanced` | *A* element of **Diffuse Color** [R G B A] vector | Scalar number (0–1) |

The figure shows the effect of varying the opacity of a solid. The array of spheres have identical graphical properties, with the exception of **Opacity**. The opacity value progresses from 0.1, at the left corner, to 1, at the right corner. An opacity value of 0 represents a completely transparent, or invisible, solid. An opacity value of 1 represents a completely opaque solid.

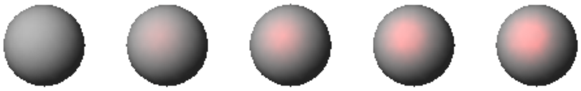
## Advanced Graphic Parameters

In addition to the diffuse color and opacity, the `Advanced` parameterization provides a set of colors that enhance the 3–D graphical appearance of the solid. The additional colors include specular, ambient, and emissive colors, each of which includes an opacity (*A*) element in the [R G B A] color vector. You can omit the fourth element in the RGBA vector, in which case the color uses a maximum opacity value of `1`.

### Specular Color

The specular color is the apparent color of the glossy highlights arising from a solid surface exposed to direct light. The size of the specular highlights depends on the value of the **Shininess** parameter. The intensity of the specular color is not uniform in space, and has a strong dependence on the viewing angle. Changing the specular color changes only the color of the specular highlights. For most applications, the [R G B A] vector [0.5 0.5 0.5 1] works well.

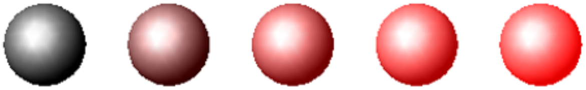
The figure shows the effect of varying the specular color of a solid. The array of spheres have identical graphical properties, with the exception of **Specular Color**. The RGBA color vector of the specular color progresses from [1 1 1 1], at the left corner, to [1 0 0 1], at the right corner. A gray ambient color gives the solid a darker appearance.



### Ambient Color

The ambient color is the apparent color of a solid surface exposed only to indirect light. Changing the ambient color changes the overall color of the entire solid surface. For most applications, the RGBA vector [0.15 0.15 0.15 1] works well.

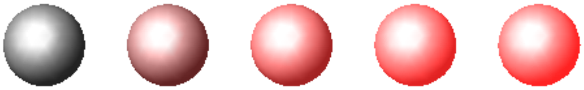
The figure shows the effect of varying the ambient color of a solid. The array of spheres have identical graphical properties, with the exception of the **Ambient Color**. The RGBA color vector of the ambient color progresses from [1 1 1 1], at the left corner, to [1 0 0 1], at the right corner. A gray ambient color gives the solid in the left corner a darker appearance.



### Emissive Color

The emissive color is the apparent color of light emitted directly by the solid surface. Examples of solids with a nonzero emissive color include glowing hot metal, light displays, and the Sun. For most applications, the RGBA vector [0 0 0 1] works well.

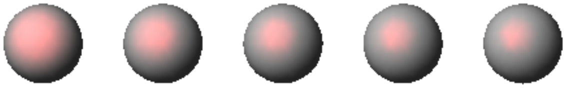
The figure shows the effect of varying the emissive color of a solid. The array of spheres have identical graphical properties, with the exception of the **Emissive Color**. The RGBA color vector of the emissive color progresses from [1 1 1 1], at the left corner, to [1 0 0 1], at the right corner. A gray ambient color gives the solid in the left corner a darker appearance. The glowing appearance of the emissive color differentiates the emissive color from ambient and diffuse colors.



## Shininess

The shininess is a parameter that encodes the size and rate of decay of specular highlights on a solid surface. A small shininess value corresponds to a large specular highlight with gradual falloff in highlight intensity. On the other hand, a large shininess value corresponds to a small specular highlight with sharp falloff in highlight intensity.

The figure shows the effect of varying the shininess of a solid. The array of spheres have identical graphical properties, with the exception of **Shininess**. The shininess value progresses from 5, at the left corner, to 25, at the right corner. As the shininess value increases, the area of the specular highlight decreases, while the falloff rate in highlight intensity increases.

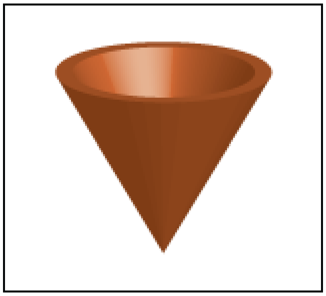


# Model Cone

**In this section...**

## Model Overview

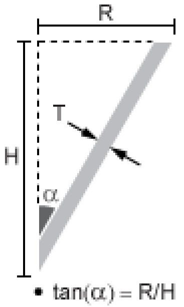
You can model solids of revolution using the SimMechanics `Revolution` shape. Examples of solids of revolution include cone and circular dome shapes. In this example, you model a simple solid with cone shape using the `Revolution` shape. For an example that shows you how to model a circular dome solid, see "Model Dome" on page 2-56.



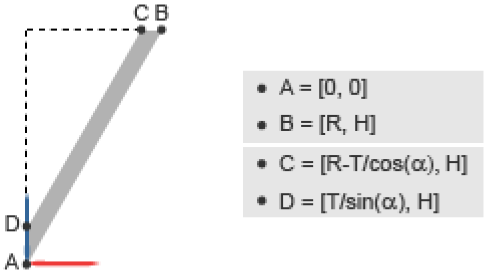
## Modeling Approach

To represent the cone geometry, first identify its cross-section shape. This is the 2-D area that SimMechanics revolves to obtain the 3-D cone. Then, specify the cross-section coordinates in the `Solid` block dialog box. These coordinates must satisfy certain restrictions. See "Revolution and General Extrusion Shapes" on page 2-17.

The cone in this example has a trapezoidal cross-section. The figure shows this cross-section.



The [0 0] cross-section coordinate identifies the reference frame origin for this solid. To place the solid reference frame at the cone tip, you by specify the coordinates so that the [0 0] coordinate coincides with the tip. By parameterizing the cross-section coordinates in terms of the relevant cone dimensions, you can quickly change the cone dimensions without having to reenter the cross-section coordinates. The figure shows the parameterized cross-section coordinates points.



- A = [0, 0]
- B = [R, H]
- C = [R-T/cos(α), H]
- D = [T/sin(α), H]

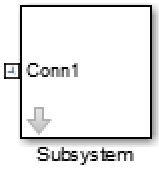
## Build Solid Model

1. At the MATLAB command prompt, enter `smnew`. A new SimMechanics model opens with some commonly used blocks. Delete all but the `Solid` block.
2. In the Solid block dialog box, specify the following parameters. You later initialize the different MATLAB variables in a subsystem mask.

| Parameter | Select or Enter |
|---|---|
| **Geometry > Shape** | `Revolution`. |
| **Geometry > Cross-Section** | `CS`, units of `cm` |

| Parameter | Select or Enter |
|---|---|
| **Inertia** > **Density** | `Rho` |
| **Graphic** > **Visual Properties** > **Color** | `RGB` |

3 Select the Solid block and generate a new subsystem, e.g., by pressing **Ctrl+G**.



## Define Solid Properties

1 Select the Subsystem block and create a subsystem mask, e.g., by pressing **Ctrl+M**.

2 In the **Parameters & Dialog** tab of the Mask Editor, drag five Edit boxes into the **Parameters** group and specify these parameters.

| Prompt | Name |
|---|---|
| `Base Radius` | `R` |
| `Cone Height` | `H` |
| `Wall Thickness` | `T` |
| `Density` | `Rho` |
| `Color` | `RGB` |

3 In the **Initialization** tab of the Mask Editor, define the cross-section coordinates and assign them to the MATLAB variable `CS`:

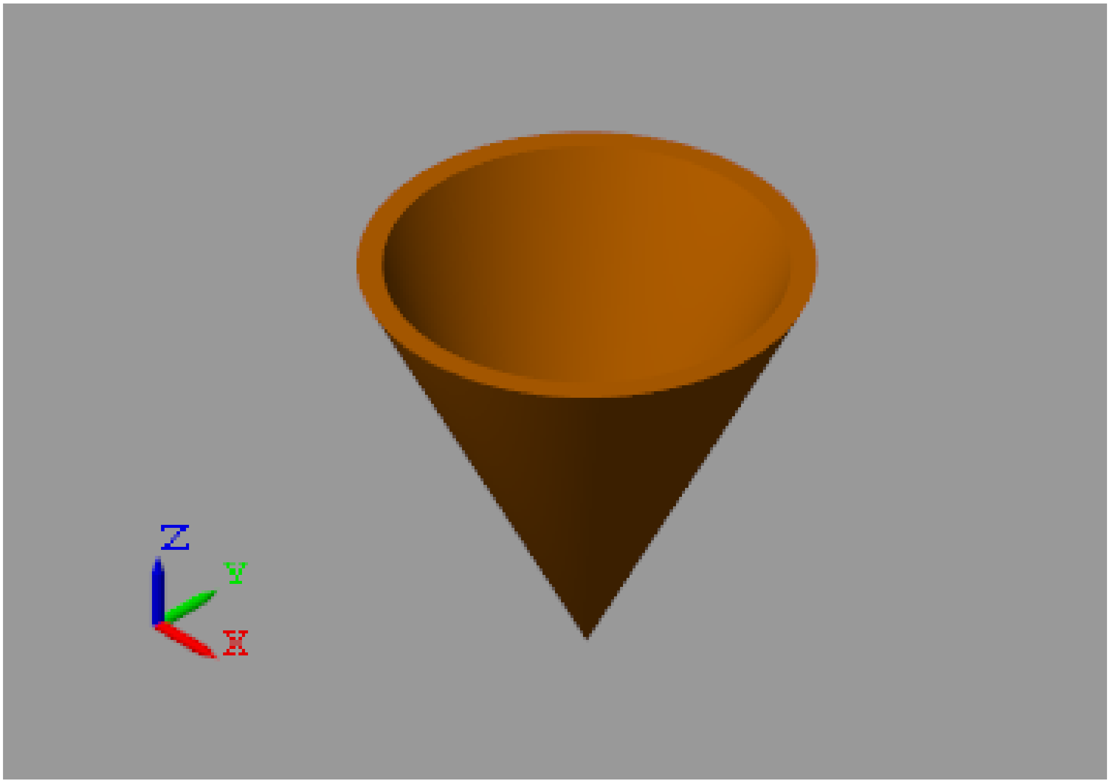
```
Alpha = atan(R/H);
CS = [0 0; R H; R-T/cos(Alpha) H; 0 T/sin(Alpha)];
```

4. In the Subsystem block dialog box, specify the numerical values of the solid properties. The table shows some values that you can enter.

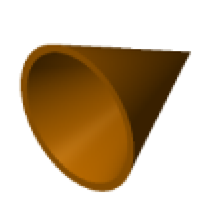
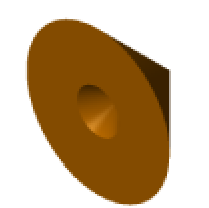
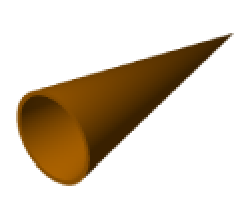
| Parameter | Enter |
|---|---|
| **Base Radius** | `1` |
| **Cone Height** | `2` |
| **Wall Thickness** | `0.1` |
| **Density** | `2700` |
| **Color** | `[0.85 0.45 0]` |

## Visualize Solid Model

You can now visualize the cone solid. To do this, look under the Subsystem mask—e.g., by selecting the Subsystem block and pressing **Ctrl+U**—and open the Solid block dialog box. The solid visualization pane shows the solid that you modeled.



Parameterizing the solid dimensions in terms of MATLAB variables enables you to modify the solid shape without having to redefine its cross-section coordinates. You can change the solid size and proportions simply by changing their values in the Subsystem block dialog box. The figure shows some examples.

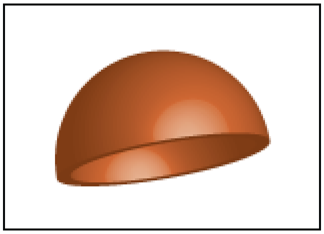
|  |  |  |
|---|---|---|
|  |  |  |
| R = 1;<br>H = 2;<br>T = 0.1; | R= 1;<br>H = 1;<br>T = 0.5; | R = 1;<br>H = 5;<br>T = 0.1; |

# Model Dome

| In this section... |
|---|
| "Model Overview" on page 2-56 |
| "Modeling Approach" on page 2-56 |
| "Build Solid Model" on page 2-57 |
| "Define Solid Properties" on page 2-58 |
| "Visualize Solid Model" on page 2-59 |

## Model Overview

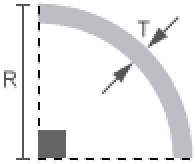
You can model a solid of revolution with a round cross-section. One example is the circular dome. In this example, you specify the cross-section coordinates of a circular dome using the MATLAB `cos` and `sin` functions. For an example that shows you how to model a cone-shaped solid, see "Model Cone" on page 2-50.



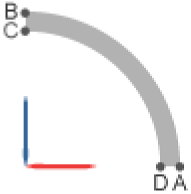
## Modeling Approach

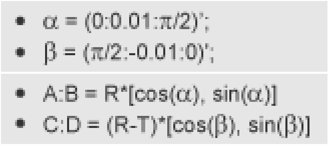
To represent the dome geometry, first identify its cross-section shape. This is the 2-D shape that SimMechanics revolves to obtain the 3-D dome. You can then specify the cross-section coordinates in the `Solid` block dialog box. These coordinates must satisfy certain restrictions. See "Revolution and General Extrusion Shapes" on page 2-17.

The dome has a quarter-circle cross-sectional shape. The figure shows this shape.



The [0 0] cross-section coordinate identifies the reference frame origin for this solid. To place the solid reference frame at the dome base center, you specify the coordinates so that the [0 0] coordinate coincides with the base center. By parameterizing the cross-section coordinates in terms of the relevant dome dimensions, you can quickly change the dome dimensions without having to reenter the cross-section coordinates. The figure shows the parameterized cross-section coordinate points.





- $\alpha$ = (0:0.01:$\pi$/2)';
- $\beta$ = ($\pi$/2:-0.01:0)';

- A:B = R*[cos($\alpha$), sin($\alpha$)]
- C:D = (R-T)*[cos($\beta$), sin($\beta$)]

To define the dome cross-section, first define two angle arrays—one in counterclockwise order, running from 0–90°; the other in a clockwise order running from 90–0°. You can then use the first array to define the outer cross-section coordinates, and the second array to define the inner cross-section coordinates. You do that using the MATLAB `cos` and `sin` functions.

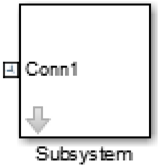
## Build Solid Model

1. At the MATLAB command prompt, enter `smnew`. A new SimMechanics model opens with some commonly used blocks. Delete all but the `Solid` block.
2. In the Solid block dialog box, specify the following parameters. You later initialize the different MATLAB variables in a subsystem mask.

| Parameter | Select or Enter |
|---|---|
| **Geometry > Shape** | `Revolution` |
| **Geometry > Cross-Section** | `CS`, units of `cm` |
| **Inertia > Density** | `Rho` |

| Parameter | Select or Enter |
|---|---|
| **Graphic > Visual Properties > Color** | RGB |

3 Select the Solid block and generate a subsystem, e.g., by pressing **Ctrl+G**.



## Define Solid Properties

1 Select the Subsystem block and create a subsystem mask, e.g., by pressing **Ctrl+M**.

2 In the **Parameters & Dialog** tab of the Mask Editor, drag four Edit boxes into the **Parameters** group and specify these parameters.

| Prompt | Name |
|---|---|
| `Base Radius` | R |
| `Wall Thickness` | T |
| `Density` | Rho |
| `Color` | RGB |

3 In the **Initialization** tab of the Mask Editor, define the cross-section coordinates and assign them to the MATLAB variable `CS`:

```
% Circular dome outer coordinates:
Alpha = (0:0.01:pi/2)';
OuterCS = R*[cos(Alpha), sin(Alpha)];

% Circular dome inner coordinates:
Beta = (pi/2:-0.01:0)';
```

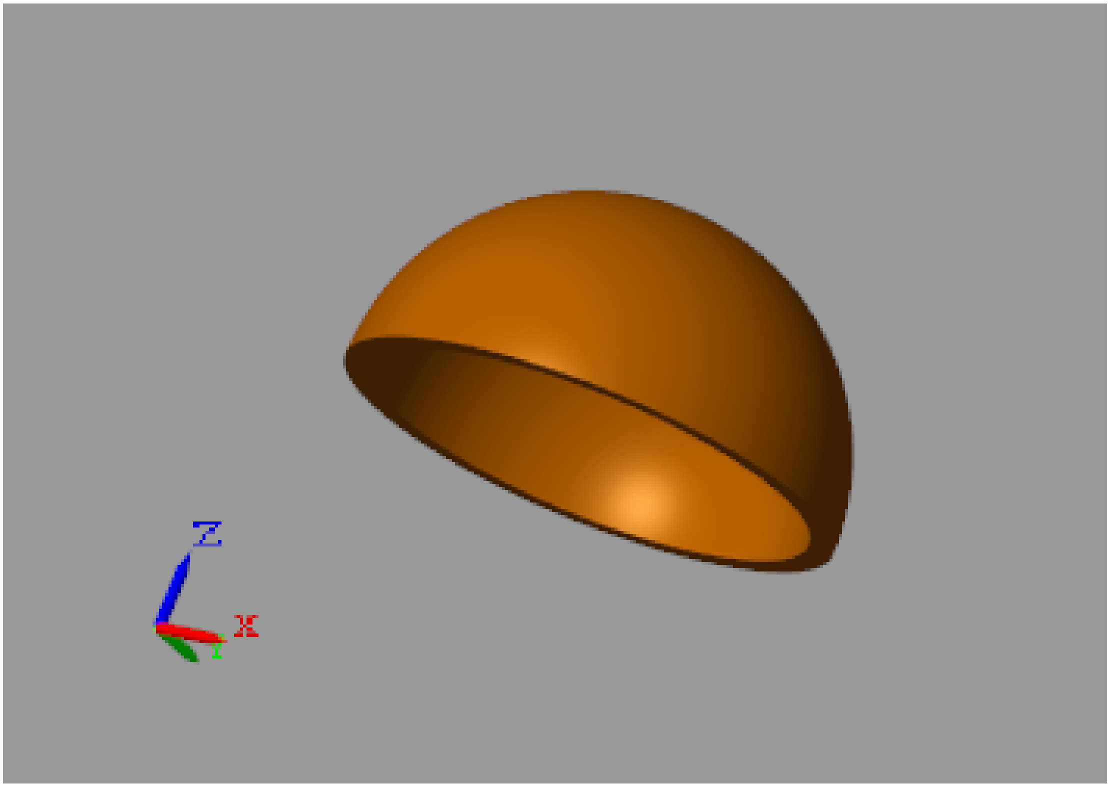
```
InnerCS = (R-T)*[cos(Beta), sin(Beta)];

CS = [OuterCS; InnerCS];
```

4 In the Subsystem block dialog box, specify the numerical values of the solid properties. The table shows some values that you can enter.

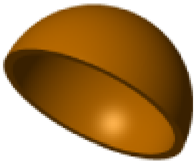
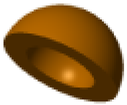
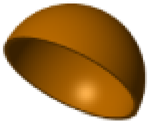
| Parameter | Enter |
|---|---|
| **Base Radius** | `1` |
| **Wall Thickness** | `0.1` |
| **Density** | `2700` |
| **Color** | `[0.85 0.45 0]` |

## Visualize Solid Model

You can now visualize the dome solid. To do this, look under the Subsystem mask—e.g., by selecting the Subsystem block and pressing **Ctrl+U**—and open the Solid block dialog box. The solid visualization pane shows the solid that you modeled.



Parameterizing the solid dimensions in terms of MATLAB variables enables you to modify the solid shape without having to redefine its cross-section coordinates. You can change the solid size and proportions simply by changing their values in the Subsystem block dialog box. The figure shows some examples.

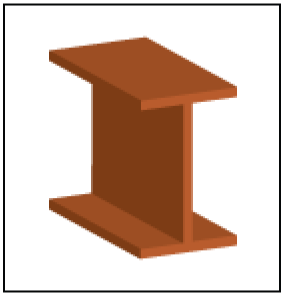
| | | |
|---|---|---|
|  |  |  |
| R = 1;<br>T = 0.1; | R = 0.5;<br>T = 0.2; | R = 0.75;<br>T = 0.05; |

# Model I-Beam

In this section...

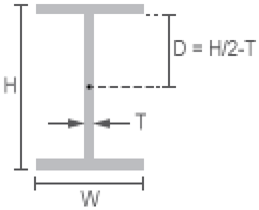
## Model Overview

You can model an extrusion using the SimMechanics shape `General Extrusion`. Examples of extrusions include the I-beam and box-beam shapes. In this example, you model a simple solid with I-beam shape using the `General Extrusion` shape. For an example that shows you how to model a box beam, see "Model Box Beam" on page 2-68.



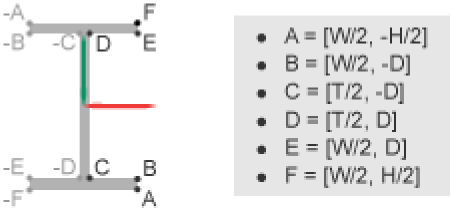
## Modeling Approach

To represent the I-beam geometry, first identify its cross-section. This is the 2-D area that SimMechanics extrudes to obtain the 3-D I-beam. You can then specify the cross-section coordinates in the `Solid` block dialog box. The figure shows the I-beam cross-section that you specify in this example.



The [0 0] coordinate identifies the solid reference frame origin. To place the reference frame at the center of the I-beam, specify the coordinates so that the [0 0] coordinate is at the cross-section center. Because the I-beam cross-section is symmetric about the horizontal and vertical axes, you need only define the coordinates for one cross-section half—e.g, the right half. You can then define the left half coordinates in terms of the right half coordinates.

By parameterizing the cross-section coordinates in terms of relevant I-beam dimensions, you can quickly change the I-beam dimensions without having to reenter the cross-section coordinates. The figure shows the cross-section dimensions and coordinates that you must specify to represent the I-beam.



- A = [W/2, -H/2]
- B = [W/2, -D]
- C = [T/2, -D]
- D = [T/2, D]
- E = [W/2, D]
- F = [W/2, H/2]

Using the cross-section points that the figure shows, you define the coordinate matrix as:

```
HalfCS = [A; B; C; D; E; F];
CS = [HalfCS; -HalfCS];
```

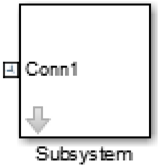
## Build Solid Model

1 At the MATLAB command prompt, enter `smnew`. A new SimMechanics model opens with some commonly used blocks. Delete all but the `Solid` block.

2 In the Solid block dialog box, specify the following parameters. You later initialize the different MATLAB variables in a subsystem mask.

| Parameter | Select or Enter |
|---|---|
| **Geometry > Shape** | `General Extrusion` |
| **Geometry > Cross-Section** | `CS`, units of `cm` |
| **Geometry > Length** | `L`, units of `cm` |
| **Inertia > Density** | `Rho` |
| **Graphic > Visual Properties > Color** | `RGB` |

3 Select the Solid block and generate a new subsystem, e.g., by pressing **Ctrl+G**.



## Define Solid Properties

1 Select the Subsystem block and create a subsystem mask, e.g., by pressing **Ctrl+M**.

2 In the **Parameters & Dialog** tab of the Mask Editor, drag six Edit boxes into the **Parameters** group and specify these parameters.

| Prompt | Name |
|---|---|
| `Length` | `L` |
| `Height` | `H` |
| `Width` | `W` |
| `Thickness` | `T` |

| Prompt | Name |
|---|---|
| `Density` | `Rho` |
| `Color` | `RGB` |

3 In the **Initialization** tab of the Mask Editor, define the cross-section coordinates and assign them to MATLAB variable `CS`:

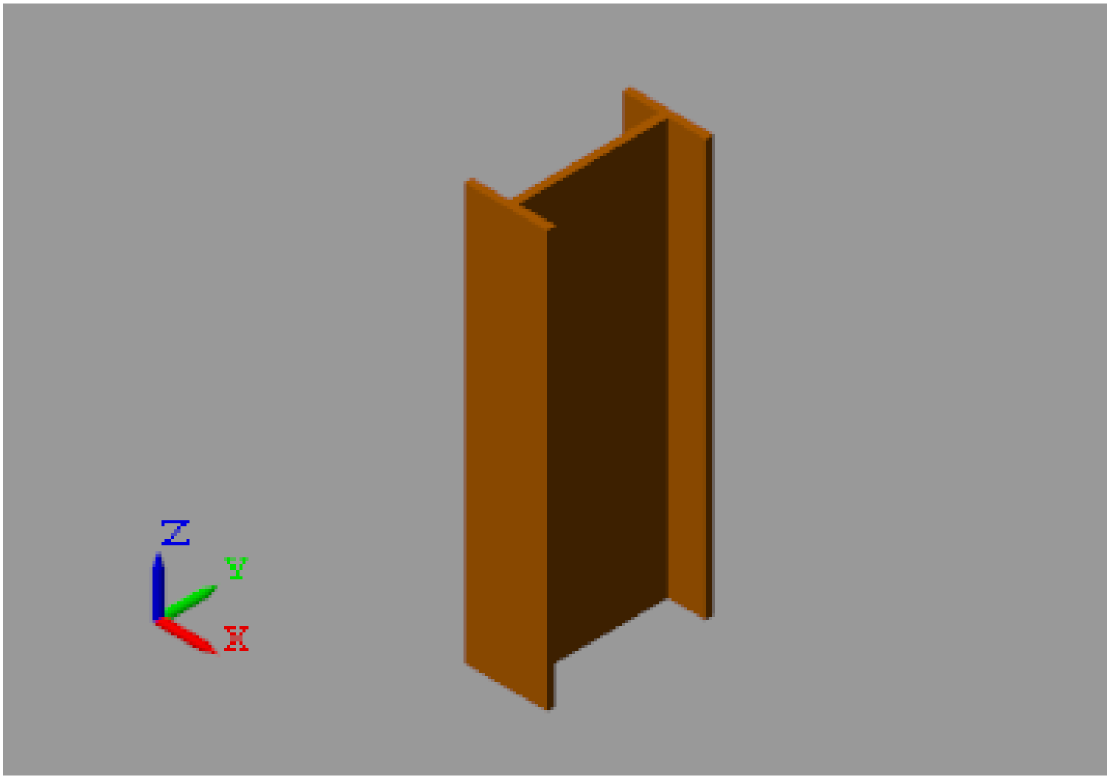
```
D = H/2-T;
HalfCS = [W/2, -H/2; W/2, -D; T/2, -D;...
T/2, D; W/2, D; W/2, H/2];
CS = [HalfCS; -HalfCS];
```

4 In the Subsystem block dialog box, specify the numerical values of the solid properties. The table shows some values that you can enter.

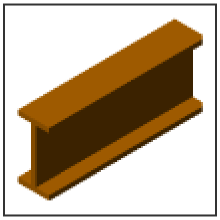
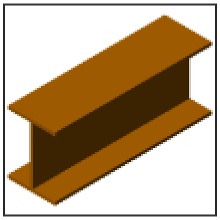
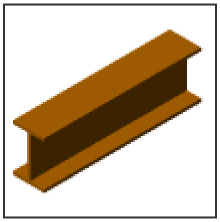
| Parameter | Enter |
|---|---|
| **Length** | `10` |
| **Height** | `4` |
| **Width** | `2` |
| **Thickness** | `0.2` |
| **Density** | `2700` |
| **Color** | `[0.85 0.45 0]` |

## Visualize Solid Model

You can now visualize the I-beam solid. To do this, look under the Subsystem mask—e.g., by selecting the Subsystem block and pressing **Ctrl+U**—and open the Solid block dialog box. The solid visualization pane shows the solid that you modeled.



Parameterizing the solid dimensions in terms of MATLAB variables enables you to modify the solid shape without having to redefine its cross-section coordinates. You can change the solid size and proportions simply by changing their values in the Subsystem block dialog box. The figure shows some examples.

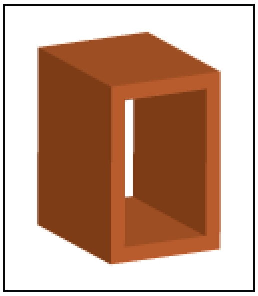
| | | |
|---|---|---|
|  |  |  |
| L = 10;<br>H = 4;<br>W =  2;<br>T = 0.3; | L = 10;<br>H = 4<br>W =  3;<br>T = 0.2; | L = 10;<br>H = 3;<br>W =  2;<br>T = 0.2; |

# Model Box Beam

**In this section...**

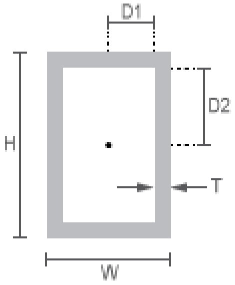
## Model Overview

You can model an extrusion with a hole. One example is the box beam. Specifying hollow cross-sections must satisfy the cross-section guidelines. See "Revolution and General Extrusion Shapes" on page 2-17. In this example, you specify the cross-section coordinates of a box beam. For an example that shows you how to model an I-beam extrusion, see "Model I-Beam" on page 2-62.

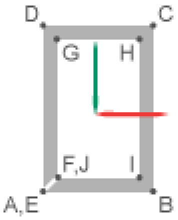


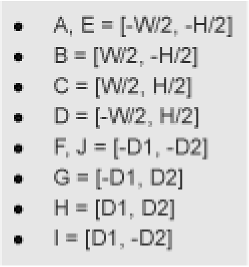
## Modeling Approach

To represent the box beam geometry, first identify its cross-section. This is the 2-D area that you sweep along an axis to obtain the 3-D box beam. You can then specify the cross-section coordinates using the `Solid` block. The figure shows the box beam cross-section that you specify in this example.



The [0 0] coordinate identifies the solid reference frame origin. To place the reference frame at the center of the box beam, specify the coordinates so that the [0 0] coordinate is at the cross-section center. By parameterizing the cross-section coordinates in terms of relevant box beam dimensions, you can later change the box beam dimensions without having to reenter the cross-section coordinates. The figure shows the cross-section dimensions and coordinates that you must specify to represent the box beam.





- A, E = [-W/2, -H/2]
- B = [W/2, -H/2]
- C = [W/2, H/2]
- D = [-W/2, H/2]
- F, J = [-D1, -D2]
- G = [-D1, D2]
- H = [D1, D2]
- I = [D1, -D2]

Using the cross-section points that the figure shows, you define the coordinate matrix as:

```
OuterCS = [A, B, C, D, E];
InnerCS = [F, G, H, I, J];
CS = [OuterCS; InnerCS];
```

For more information about specifying the hollow cross-section coordinates, see “Hollow Cross Sections” on page 2-18.

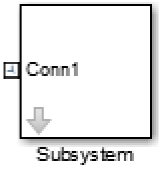
## Build Solid Model

1 At the MATLAB command prompt, enter `smnew`. A new SimMechanics model opens with some commonly used blocks. Delete all but the `Solid` block.

2 In the Solid block dialog box, specify the following parameters. You later initialize the different MATLAB variables in a subsystem mask.

| Parameter | Select or Enter |
|---|---|
| **Geometry** > **Shape** | `General Extrusion` |
| **Geometry** > **Cross-Section** | `CS`, units of `cm` |
| **Geometry** > **Length** | `L`, units of `cm` |
| **Inertia** > **Density** | `Rho` |
| **Graphic** > **Visual Properties** > **Color** | `RGB` |

3 Select the Solid block and generate a new subsystem, e.g., by pressing **Ctrl+G**.



## Define Solid Properties

In the subsystem mask, initialize the solid parameters. Then, in the subsystem dialog box, specify their values.

1 Select the Subsystem block and create a subsystem mask, e.g., by pressing **Ctrl+M**.

2 In the **Parameters & Dialog** tab of the Mask Editor, drag six Edit boxes into the **Parameters** group and specify these parameters.

| Prompt | Name |
|---|---|
| `Length` | `L` |
| `Height` | `H` |

| Prompt | Name |
|---|---|
| `Width` | `W` |
| `Thickness` | `T` |
| `Density` | `Rho` |
| `Color` | `RGB` |

3 In the **Initialization** tab of the Mask Editor, define the cross-section coordinates and assign them to MATLAB variable `CS`:

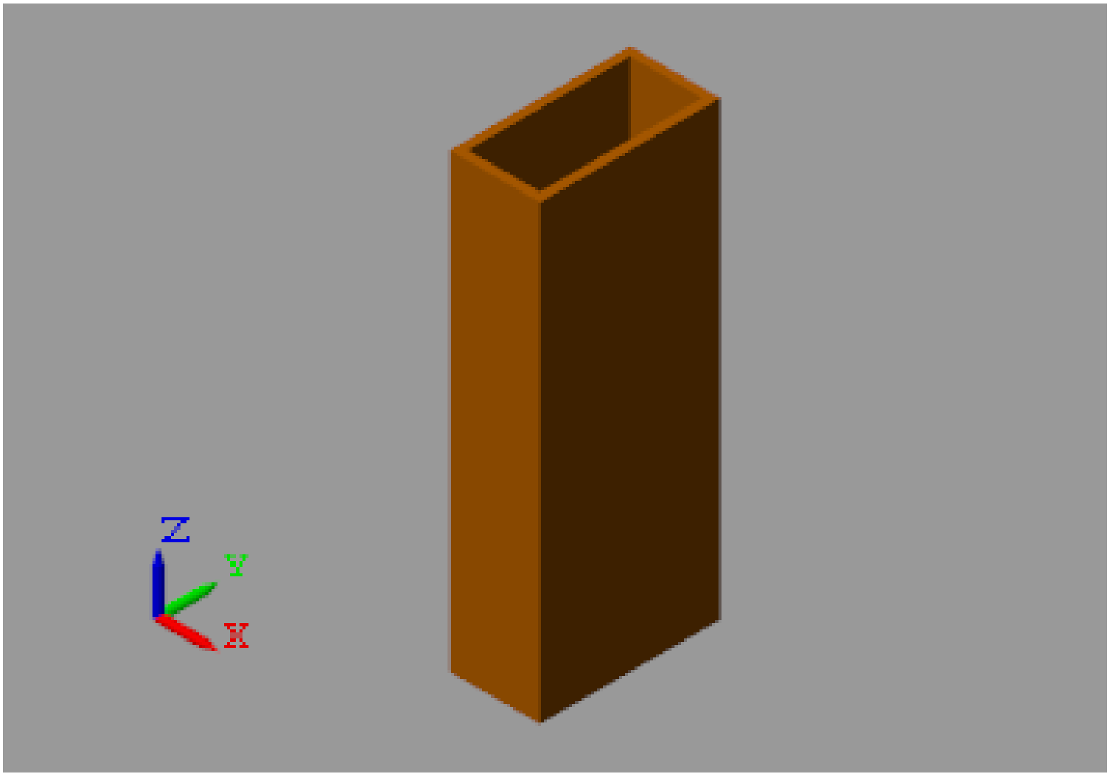
```
D1 = W/2-T;
D2 = H/2-T;
OuterCS = [-W/2,-H/2; W/2,-H/2; W/2,H/2; ...
-W/2,H/2; -W/2,-H/2];
InnerCS = [-D1,-D2; -D1,D2; D1,D2; D1 -D2; -D1,-D2];
CS = [OuterCS; InnerCS];
```

4 In the Subsystem block dialog box, specify the numerical values of the solid properties. The table shows some values that you can enter.

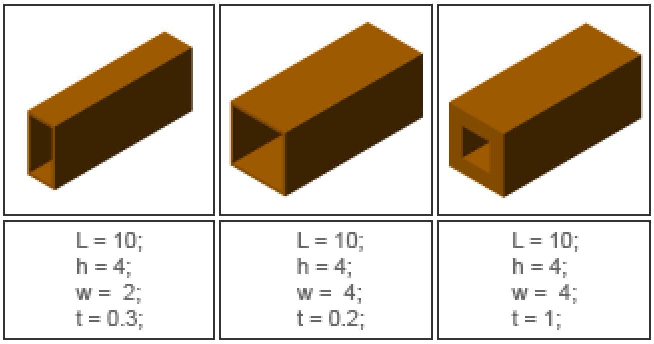
| Parameter | Enter |
|---|---|
| **Length** | `10` |
| **Height** | `4` |
| **Width** | `2` |
| **Thickness** | `0.2` |
| **Density** | `2700` |
| **Color** | `[0.85 0.45 0]` |

## Visualize Solid Model

You can now visualize the box-beam solid. To do this, look under the Subsystem mask —e.g., by selecting the Subsystem block and pressing **Ctrl+U**—and open the Solid block dialog box. The solid visualization pane shows the solid that you modeled.



Parameterizing the solid dimensions in terms of MATLAB variables enables you to modify the solid shape without having to redefine its cross-section coordinates. You can change the solid size and proportions simply by changing their values in the Subsystem block dialog box. The figure shows some examples.



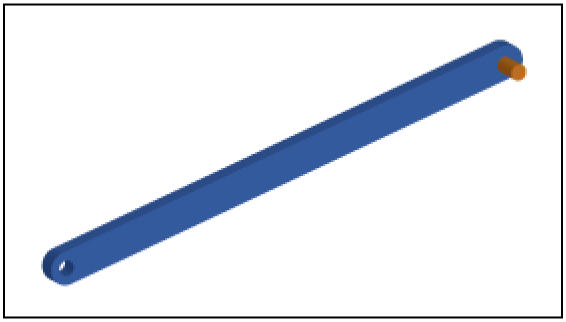
| | | |
|---|---|---|
| L = 10;<br>h = 4;<br>w =  2;<br>t = 0.3; | L = 10;<br>h = 4;<br>w =  4;<br>t = 0.2; | L = 10;<br>h = 4;<br>w =  4;<br>t = 1; |

# Model Binary Link

**In this section...**

## Model Overview

In example "Represent Binary Link Frame Tree" on page 1-33, you modeled the frame tree of a binary link rigid body. In this example, you add to that frame tree the solid properties of the binary link: geometry, inertia, and color.



## Modeling Approach

To model a binary link, you must use multiple Solid blocks. Each Solid block represents an elementary portion of the binary link. Rigid bodies that you model using multiple

Solid blocks are called *compound rigid bodies*. The compound rigid body technique reduces a single complex task (modeling the entire binary link shape) into several simple tasks (modeling the Main, Hole, and Peg sections of the binary link).

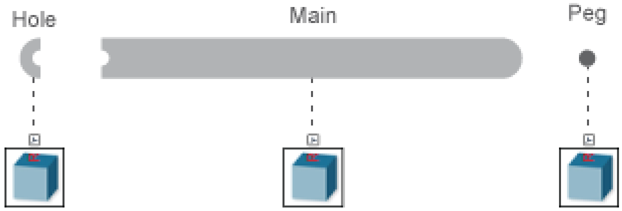
To use the compound rigid body technique:

1 Divide shape into simple sections.

Dividing the shape simplifies the modeling task in more complex cases. You can divide the binary link into three simple sections: Main, Peg, and Hole, shown in the figure.

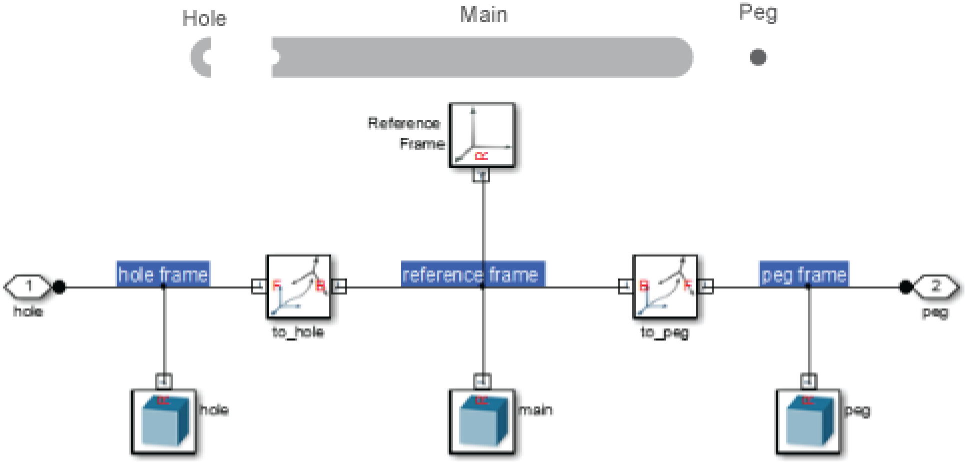
2 Represent each section using a Solid block.

Each section should be simple enough to model using a single Solid block. In the binary link example, you can represent sections Main and Hole using SimMechanics shape `General Extrusion`, and section peg with SimMechanics shape `Cylinder`.



3 Rigidly connect Solid blocks to rigid body frame tree.

Rigid connections ensure the different solid sections move as a single rigid body. Connect the Solid blocks to the binary link frame tree to apply the correct spatial relationships between the solid sections.



## Solid Properties

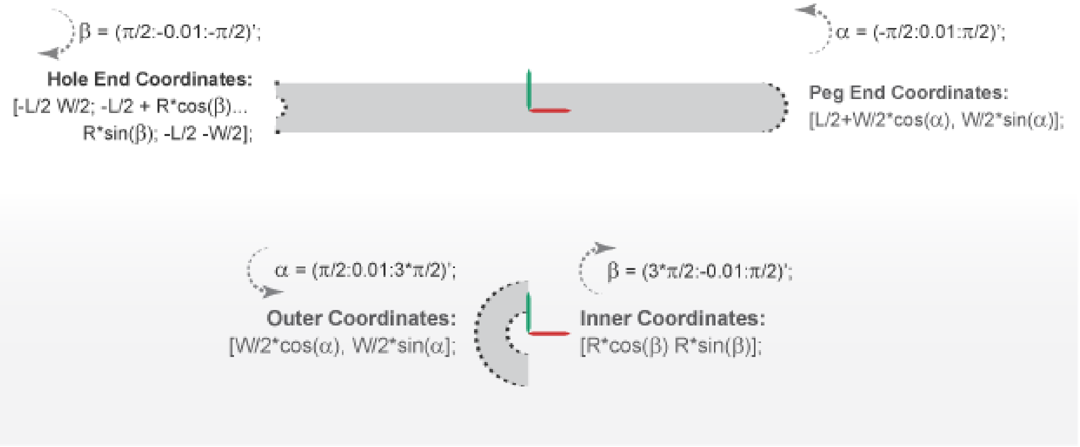
You model the binary link as a compound rigid body subsystem. In this subsystem, three Solid blocks represent the basic solid sections of the binary link. Each solid section has a shape and a local reference frame that you connect to the binary link frame tree. Two SimMechanics shapes are used: `General Extrusion` and `Cylinder`.

You can promote subsystem reusability by parameterizing solid properties in terms of MATLAB variables. In this example, you initialize the variables in a subsystem mask. You can then specify their numerical values in the subsystem dialog box. The table provides the dimensions needed to model the binary link solid sections. In the previous example, “Represent Binary Link Frame Tree” on page 1-33, you used the first three dimensions to specify the spatial relationships between the different binary link frames.

| Dimension | MATLAB Variable |
|---|---|
| Length | L |
| Width | W |
| Thickness | T |
| Peg Radius | R |

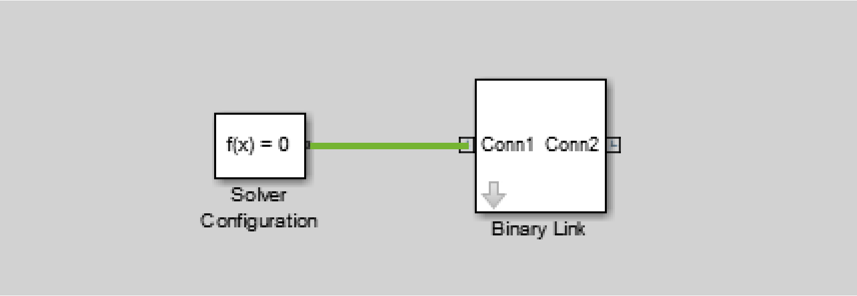
SimMechanics shape `General Extrusion` requires you to specify a set of cross-section coordinates. This is a MATLAB matrix with all the [X Y] coordinate pairs needed to draw the cross-section. Straight line segments connect adjacent coordinate pairs.

Coordinate matrices must obey a set of rules. The most important rule is that the solid region must lie to the left of the line segment connecting adjacent coordinate pairs. For more information, see “Revolution and General Extrusion Shapes” on page 2-17. The figure shows the coordinates required to specify the cross-section shapes of solid sections Main and Hole.

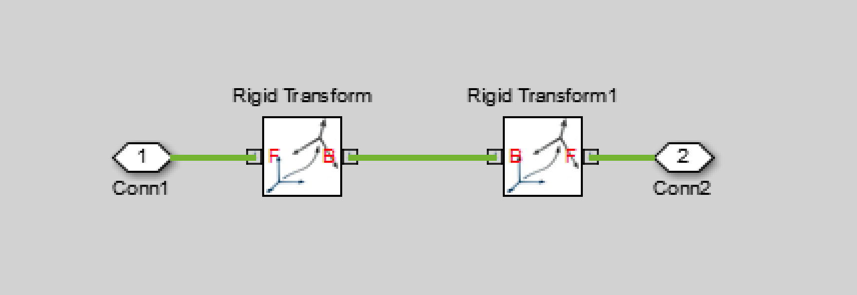


## Build Model

1 At the MATLAB command prompt, enter `smdoc_binary_link_frames`. A SimMechanics model opens with the frame tree you modeled in the “Represent Binary Link Frame Tree” on page 1-33 tutorial.

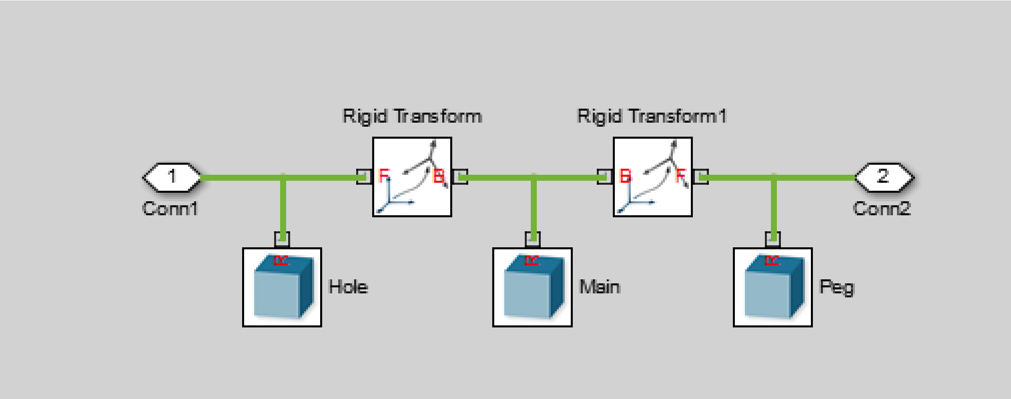


2 Right click Binary Link and select **Mask** > **Look Under Mask**.



From the SimMechanics Body Elements library, drag three `Solid` blocks into the model.

3 Connect and name the blocks as shown in the figure.



4 In the Solid block dialog boxes, specify these parameters.

| Parameter | Hole | Main | Peg |
|---|---|---|---|
| **Geometry > Shape** | Select `General Extrusion`. | Select `General Extrusion`. | Select `Cylinder`. |
| **Geometry > Cross-section** | Enter `HoleCS`. Select units of `cm`. | Enter `MainCS`. Select units of `cm`. | — |
| **Geometry > Radius** | — | — | Enter `R`. Select units of `cm`. |
| **Geometry > Length** | Enter `T`. Select units of `cm`. | Enter `T`. Select units of `cm`. | Enter `2*T`. Select units of `cm`. |
| **Geometry > Density** | Enter `Rho`. | Enter `Rho`. | Enter `Rho`. |
| **Graphic > Color** | Enter `LinkRGB`. | Enter `LinkRGB`. | Enter `PegRGB`. |

## Update Subsystem

In the subsystem mask, initialize the MATLAB variables you entered for the block parameters.

1 Select the subsystem block and press **Ctrl+M** to create a subsystem mask.

2 In the **Parameters & Dialog** tab of the Mask Editor, drag four edit boxes into the **Parameters** group and specify these parameters. Then, click **OK**.

| Prompt | Name |
|---|---|
| `Peg Radius` | `R` |
| `Mass Density` | `Rho` |
| `Link Color [R G B]` | `LinkRGB` |
| `Peg Color [R G B]` | `PegRGB` |

**Note:** The subsystem mask should contain three other parameters: L, W, and T. You specify those parameters in "Represent Binary Link Frame Tree" on page 1-33.

3 In the **Initialization** tab of the Mask Editor, define the extrusion cross-sections and press **OK**:

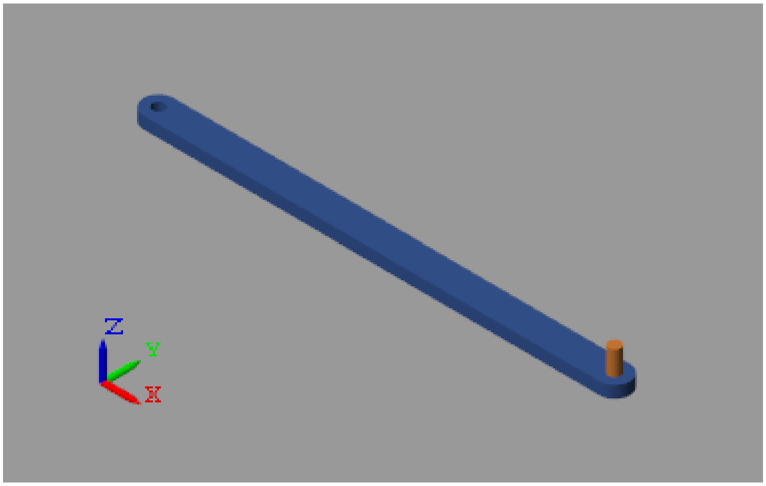
```
% Cross-section of Main:
Alpha = (-pi/2:0.01:pi/2)';
Beta = (pi/2:-0.01:-pi/2)';
PegCS = [L/2+W/2*cos(Alpha)...
W/2*sin(Alpha)];
HoleCS = [-L/2 W/2; -L/2 + R*cos(Beta)...
R*sin(Beta); -L/2 -W/2];
MainCS = [PegCS; HoleCS];

% Cross-section of Hole:
Alpha = (pi/2:0.01:3*pi/2)';
Beta = (3*pi/2:-0.01:pi/2)';
HoleCS = [W/2*cos(Alpha) W/2*sin(Alpha);
R*cos(Beta) R*sin(Beta)];
```

4 In the binary_link subsystem block dialog box, specify these parameters.

| Parameter | Value |
|---|---|
| **Length** | 30 |
| **Width** | 2 |
| **Thickness** | 0.8 |
| **Peg Radius** | 0.4 |
| **Mass Density** | 2700 |
| **Link Color [R G B]** | [0.25 0.4 0.7] |
| **Peg Color [R G B]** | [1 0.6 0.25] |

## Visualize Model

Update the block diagram. You can do this by pressing **Ctrl+D**. Mechanics Explorer opens with a static view of the binary link. To obtain the view shown in the figure, in the Mechanics Explorer toolstrip select the isometric view button .



## Open Reference Model

To view a completed version of the binary link model, at the MATLAB command prompt enter `smdoc_binary_link_a`.

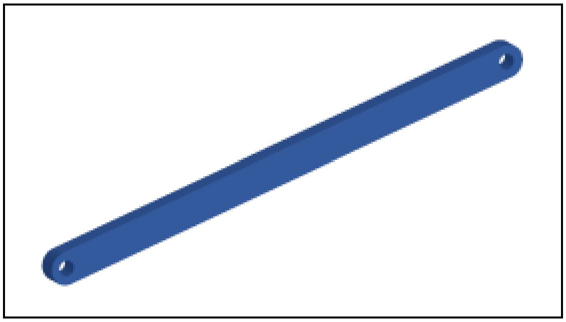
## Related Examples

- "Model Two-Hole Binary Link" on page 2-82
- "Model Pivot Mount" on page 2-88

# Model Two-Hole Binary Link

## Model Overview

In this example, you model a two-hole binary link as a rigid body. Three Solid blocks represent the main body and hole sections of the link. Two Rigid Transform blocks define the spatial relationships between the three solids. This example is a variation of "Model Binary Link" on page 2-74.



## Build Model

1 Start a new model.

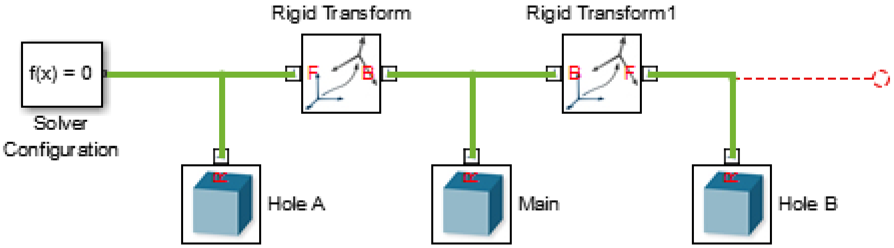
2 Drag the following blocks to the model.

| Library | Block | Quantity |
|---|---|---|
| **Simscape > Utilities** | `Solver Configuration` | 1 |

| Library | Block | Quantity |
|---|---|---|
| **SimMechanics > Second Generation > Frames and Transforms** | `Rigid Transform` | 2 |
| **SimMechanics > Second Generation > Body Elements** | `Solid` | 3 |

**3** Connect and name the blocks as shown in the figure.

Be sure to flip the Rigid Transform block. Its B frame port must face the Main Solid block. Also include the broken line extending from the Hole B block (right click the existing connection line and drag).



**4** In the solid block dialog boxes, specify these parameters.

| Parameter | Hole A | Main | Hole B |
|---|---|---|---|
| **Geometry > Shape** | Select `General Extrusion`. | Select `General Extrusion`. | Select `General Extrusion`. |
| **Geometry > Cross-section** | Enter `HoleACS`. Select units of `cm`. | Enter `MainCS`. Select units of `cm`. | Enter `HoleBCS`. Select units of `cm`. |
| **Geometry > Length** | Enter `T`. Select units of `cm`. | Enter `T`. Select units of `cm`. | Enter `T`. Select units of `cm`. |
| **Inertia > Density** | Enter `Rho`. | Enter `Rho`. | Enter `Rho`. |

| Parameter | Hole A | Main | Hole B |
|---|---|---|---|
| **Graphic > Visual Properties > Color** | Enter `LinkRGB`. | Enter `LinkRGB`. | Enter `LinkRGB`. |

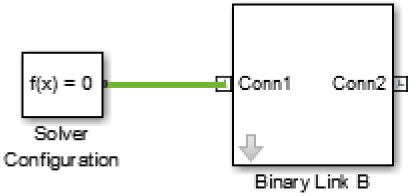
5 In the rigid transform block dialog boxes, specify these parameters.

| Parameter | Rigid Transform | Rigid Transform1 |
|---|---|---|
| **Translation > Method** | Select `Standard Axis`. | Select `Standard Axis`. |
| **Translation > Axis** | Select `+X`. | Select `+X`. |
| **Translation > Offset** | Enter `-L/2`. Select units of `cm`. | Enter `+L/2`. Select units of `cm`. |

## Generate Subsystem

Enclose the binary link blocks in a Subsystem block, define the general extrusion coordinates, and specify the relevant parameter values:

1 Select all blocks excluding Solver Configuration and press **Ctrl+G.**. Simulink encloses the selected blocks in a new subsystem block. Rename the subsystem block as shown in the figure.



2 Select the subsystem block and press **Ctrl+M**. Simulink adds a parameter mask to the subsystem block.

3 In the **Parameters & Dialog** tab of the Mask Editor, drag six edit boxes into the **Parameters** group and specify the following parameters.

| Prompt | Name |
|---|---|
| `Length` | `L` |
| `Width` | `W` |
| `Thickness` | `T` |
| `Peg Hole Radius` | `R` |
| `Mass Density` | `Rho` |
| `Link Color` | `LinkRGB` |

4 In the **Initialization** tab of the Mask Editor, define the extrusion cross sections and click **OK**:

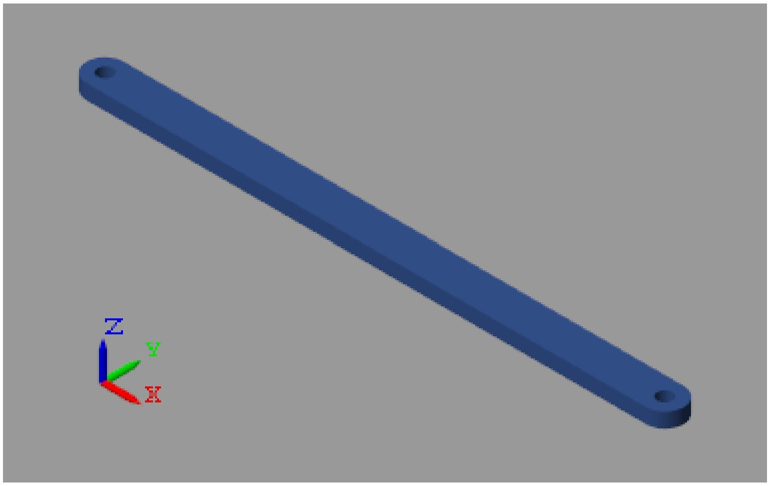
```
% Cross-section of Main:
Alpha = (pi/2:-0.01:-pi/2)';
Beta = (3*pi/2:-0.01:pi/2)';

EndACS = [-L/2 W/2; -L/2+R*cos(Alpha)...
R*sin(Alpha); -L/2 -W/2];

EndBCS = [L/2 -W/2; L/2+R*cos(Beta)...
R*sin(Beta); L/2 W/2];

MainCS = [EndACS; EndBCS];

% Cross-section of HoleA:
Alpha = (pi/2:0.01:3*pi/2)';
Beta = (3*pi/2:-0.01:pi/2)';
HoleACS = [W/2*cos(Alpha) W/2*sin(Alpha);...
R*cos(Beta) R*sin(Beta)];

% Cross-section of HoleB:
Alpha = (-pi/2:0.01:pi/2)';
Beta = (pi/2:-0.01:-pi/2)';
HoleBCS = [W/2*cos(Alpha) W/2*sin(Alpha);...
R*cos(Beta) R*sin(Beta)];
```

5 In the dialog box of the Binary Link B subsystem block, specify these parameters.

| Parameter | Value |
|---|---|
| **Length** | `30` |
| **Width** | `2` |
| **Thickness** | `0.8` |
| **Peg Hole Radius** | `0.4` |
| **Mass Density** | `2700` |
| **Link Color [R G B]** | `[0.25 0.4 0.7]` |

## Visualize Model

Update the block diagram. You can do this by pressing **Ctrl+D**. Mechanics Explorer opens with a static display of the binary link rigid body. To obtain the view shown in the figure, in the Mechanics Explorer toolstrip select the isometric view button.



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

## Open Reference Model

To open a completed version of the two-hole binary link model, at the MATLAB command prompt, enter `smdoc_binary_link_b`.

## Related Examples

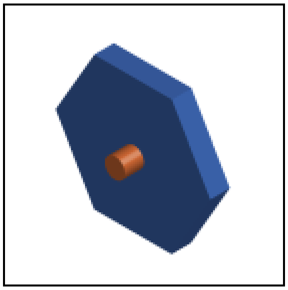
- "Model Binary Link" on page 2-74
- "Model Pivot Mount" on page 2-88
- "Model Four Bar" on page 3-19

# Model Pivot Mount

**In this section...**

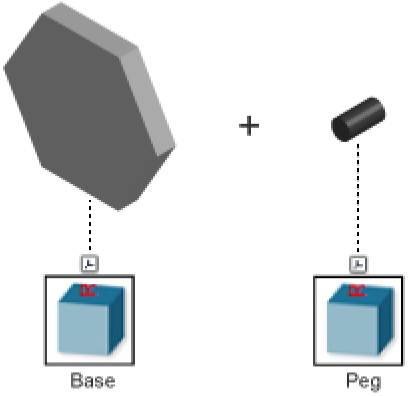
## Model Overview

In this example, you model a simple pivot mount. This mount is a compound rigid body with a hexagonal shape and a protruding cylindrical peg. You represent the hexagonal shape using solid shape `Regular Extrusion`. You then offset the protruding peg from the hexagonal shape using a Rigid Transform block. In later examples, you use this mount to support mechanical linkages like the double pendulum and the four bar system.



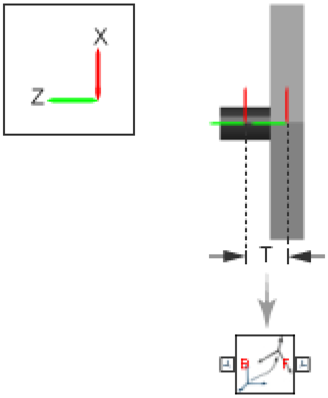
## Modeling Approach

To model the pivot mount, you use two `Solid` blocks. Because the pivot mount has a hexagonal shape, you can model it using the `Regular Extrusion` shape. To represent the cylindrical peg, you use the `Cylinder` shape.



Each shape has a reference frame with origin at the geometry center. To offset the cylindrical peg with respect to the hexagonal mount, you apply a rigid transform between the two reference frames. You do this using the Rigid Transform block.

The Z axes of the two reference frames align with the cylindrical and extrusion axes of the peg and mount, respectively. Assuming the two solids both have thickness *T*, the rigid transform between the two reference frames is a translation *T* along the common Z axis.



In later examples, you connect the pivot mount to a binary link using a revolute joint. One example is a double pendulum that moves due to gravity. In this example, it helps to rotate the Z axis of the mount so that it is orthogonal to the world frame Z axis. This task, which involves a Rigid Transform block, makes the pivot rotation axis orthogonal to the gravity vector, `[0 0 -9.81] m/s^2`.

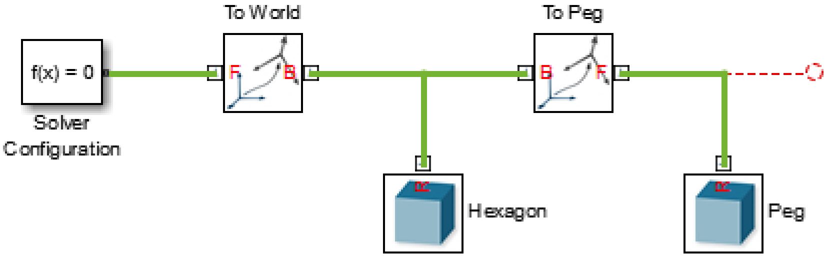
## Build Model

1 Drag these blocks into a new model.

| Block | Library | Quantity |
|---|---|---|
| `Solid` | **SimMechanics** > **Second Generation** > **Body Elements** | 2 |
| `Rigid Transform` | **SimMechanics** > **Second Generation** > **Frames and Transforms** | 2 |
| `Solver Configuration` | **Simscape** > **Utilities** | 1 |

2 Connect and name the blocks as shown in the figure.

**Note:** Include the disconnected frame line. This line becomes important when you generate a subsystem for the pivot mount. To add this line, right-click on the solid frame line and drag to the right.



3 In the Hexagon block dialog box, specify these parameters.

| Parameter | Value |
|---|---|
| **Geometry > Shape** | Select `Regular Extrusion`. |
| **Geometry > Number of Sides** | Enter `6`. |
| **Geometry > Outer Radius** | Enter `HexagonR`. Select units of `cm`. |
| **Geometry > Length** | Enter `T`. Select units of `cm`. |
| **Inertia > Density** | Enter `Rho`. |
| **Graphic > Color** | Enter `HexagonRGB`. |

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

| Parameter | Value |
|---|---|
| **Geometry > Shape** | Select `Cylinder` |
| **Geometry > Radius** | Enter `PegR`. Select units of `cm`. |
| **Geometry > Length** | Enter `2*T`. |
| **Inertia > Density** | Enter `Rho`. |
| **Graphic > Color** | Enter `PegRGB`. |

5 In the To Peg block dialog box, specify these parameters.

| Parameter | Value |
|---|---|
| **Translation > Method** | Select `Standard Axis`. |
| **Translation > Axis** | Select +Z. |
| **Translation > Offset** | Enter `3/2*T`. Select units of `cm`. |

6 In the To World block dialog box, specify these parameters.

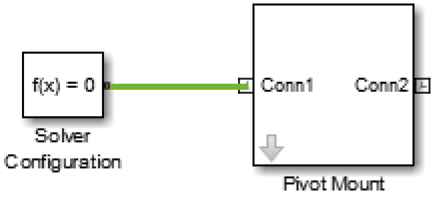
| Parameter | Value |
|---|---|
| **Rotation > Method** | Select `Standard Axis`. |
| **Rotation > Axis** | Select –Y. |
| **Rotation > Angle** | Enter `90`. |

## Generate Subsystem

You can now generate a subsystem to encapsulate the pivot mount block diagram. The subsystem mask provides a convenient place to initialize the MATLAB variables that you defined the block parameters with. To generate the subsystem:

1 Select all the blocks excluding Solver Configuration.

2 Press **Ctrl+G** to enclose the blocks in a subsystem. Name the subsystem block Pivot Mount.



3 Select the Pivot Mount block and create a subsystem mask, e.g., by pressing **Ctrl+M**.

4 In the **Parameters & Dialog** tab of the Mask Editor, drag six edit boxes into the **Parameters** group and specify their properties. Click **OK**.

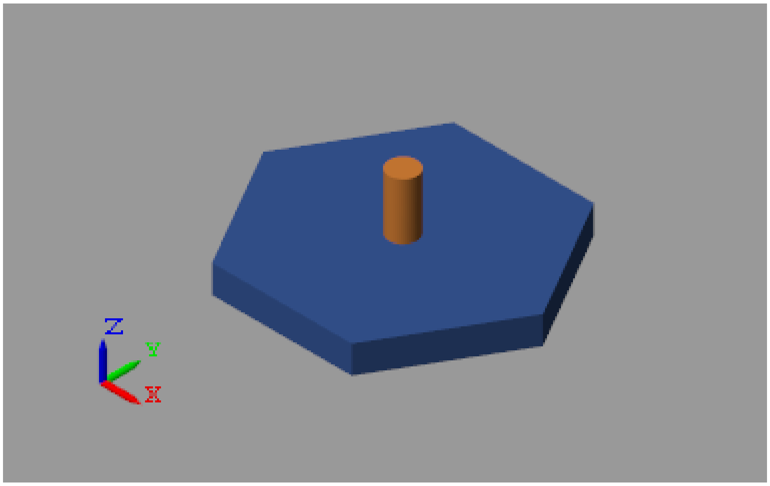
| Prompt | Name |
|---|---|
| `Hexagon Outer Radius` | `HexagonR` |
| `Hexagon Thickness` | `T` |
| `Mass Density` | `Rho` |
| `Hexagon Color` | `HexagonRGB` |
| `Peg Radius` | `PegR` |
| `Peg Color` | `PegRGB` |

5 In the Pivot Mount block dialog box, specify these parameters.

| Parameter | Value |
|---|---|
| **Hexagon Outer Radius (m):** | `4` |
| **Hexagon Thickness (m):** | `0.8` |
| **Mass Density (kg/m^3):** | `2700` |
| **Hexagon Color [R G B]:** | `[0.25 0.4 0.7]` |
| **Peg Radius (m):** | `0.4` |
| **Peg Color [R G B]:** | `[1 0.6 0.25]` |

## Visualize Model

Update the block diagram. You can do this by pressing **Ctrl+D**. Mechanics Explorer opens with a static display of the pivot mount rigid body. To obtain the view shown in the figure, in the Mechanics Explorer toolstrip select the isometric view button .



## Open Reference Model

To view a completed version of the pivot mount model, at the MATLAB command prompt enter `smdoc_pivot_mount`.

## Related Examples

- "Represent Binary Link Frame Tree" on page 1-33
- "Model Binary Link" on page 2-74

## More About

- "Representing Frames" on page 1-6
- "Revolution and General Extrusion Shapes" on page 2-17
- "Solid Inertia" on page 2-21
- "Solid Color" on page 2-45

# Multibody Systems

# Modeling Joint Connections

**In this section...**

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 SimMechanics, you model both connection types using Joint blocks.

**Examples of physical and virtual connections between bodies**

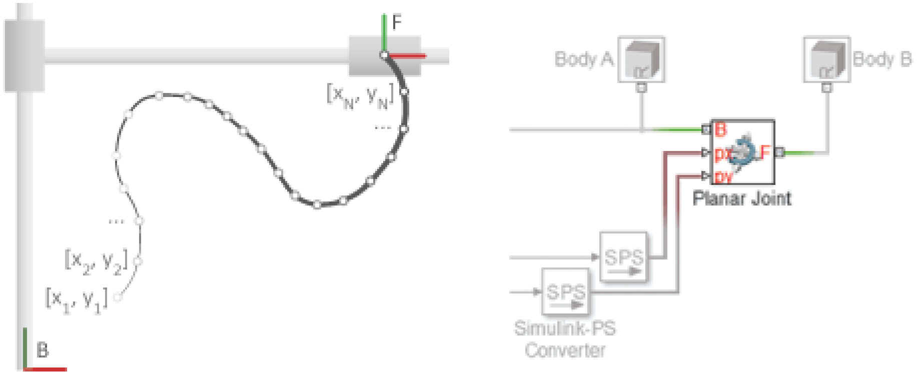
Gear and Constraint blocks too impose kinematic constraints between bodies. How are joint blocks different? While Gear and Constraint blocks are parameterized in terms of the DoFs they remove between bodies, Joint blocks are parameterized in terms of the DoFs they provide, through modules called joint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

## Joint Degrees of Freedom

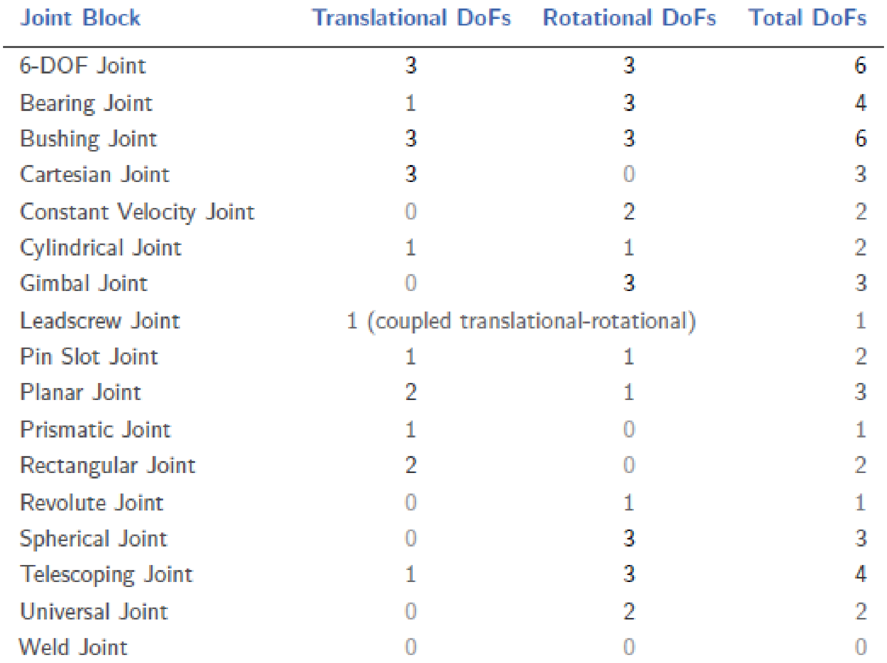
Each Joint block connects exactly two bodies. Such a connection determines the maximum degrees of freedom, or DoFs, that the adjoining bodies can share. These DoFs range from zero in the Weld Joint block to six—three translational and three rotational—in 6-DOF Joint and Bushing Joint blocks. Translation 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 mode 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.

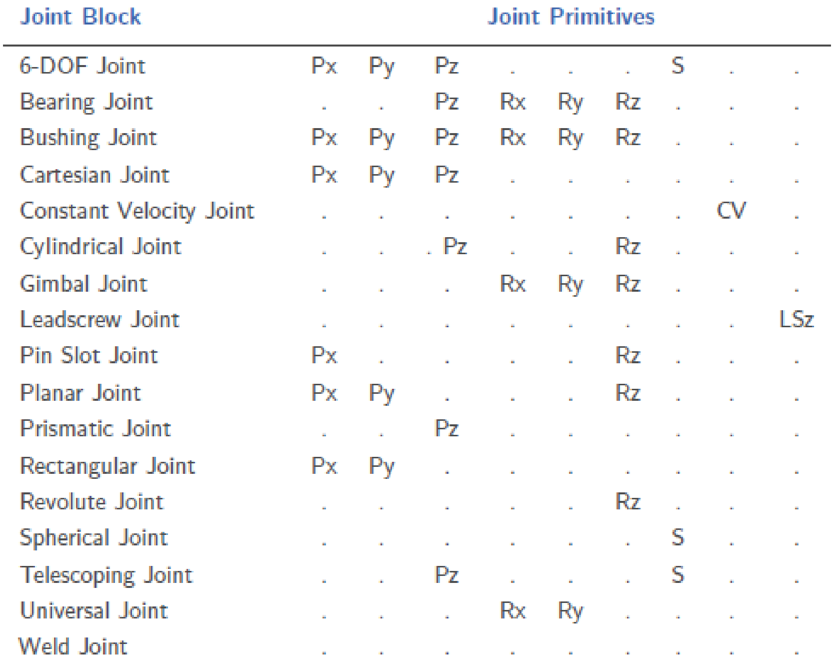
## Joint Primitives

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

SimMechanics joints are idealizations. 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 "Solid Inertia" on page 2-21.

# Assembling Multibody Models

**In this section...**

## Model Assembly

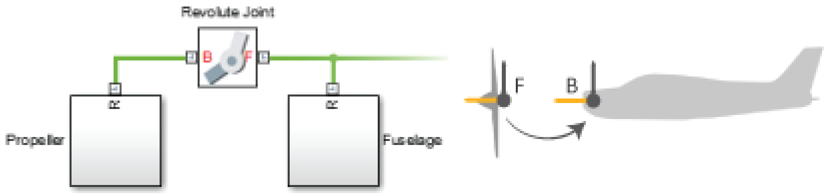
You model an articulated system by interconnecting bodies through joints and occasionally gears and other constraints. Bodies contribute their inertias to the model, while joints, gears, and constraints determine the relative degrees of freedom that exist between the bodies. You interconnect the two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

SimMechanics 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, SimMechanics 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 a tutorial on how to transform frames using the Rigid Transform block, see “Represent Binary Link Frame Tree” on page 1-33. For more information on how SimMechanics software interprets frame ports, nodes, and lines, see “Representing Frames” on page 1-6.

## 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.

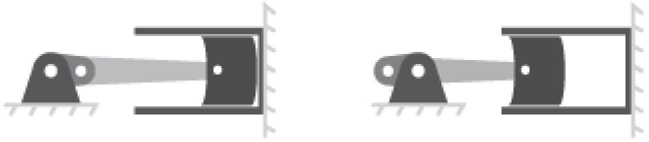
You can specify and change joint alignment by rotating the connection frames local to the adjoining body subsystems. For this purpose, you specify rotation transforms using Rigid Transform blocks, either by adding new blocks to the body subsystems or, if appropriate, by changing the rotation transforms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

For an example of how to rotate joint connection frames, see “Model Pivot Mount” on page 2-88.

## 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, SimMechanics 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 SimMechanics 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°, SimMechanics attempts to assemble the joint at 45°.

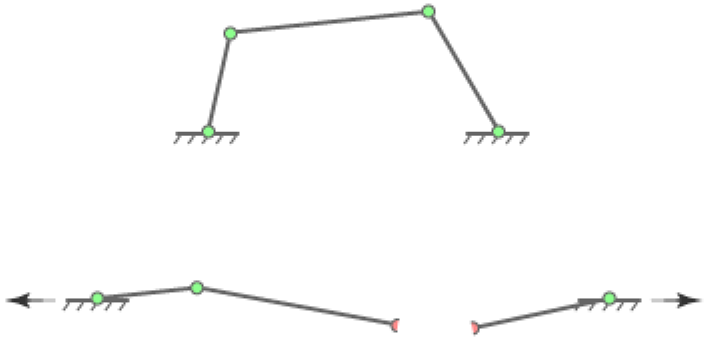
How close the actual joint state is to the state target depends on the kinematic constraints in your model, any conflicts with other state targets, and the state target priority level—a ranking that determines which of two state targets to satisfy if they prove to be mutually incompatible. You can set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

SimMechanics first attempts to satisfy all state targets exactly. If a state target conflict arises, SimMechanics ignores the low-priority state targets and attempts to satisfy only the high-priority state targets. If a state target conflict still exists, SimMechanics 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

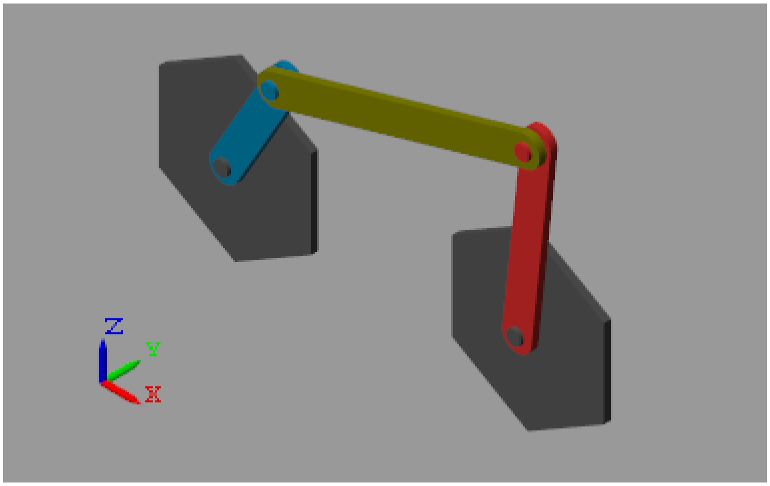
A model assembles successfully only if the connections between its bodies are congruous with each other. If in satisfying one kinematic constraint, SimMechanics must violate another kinematic constraint, the model is kinematically invalid and assembly fails. This happens, for example, when the ground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



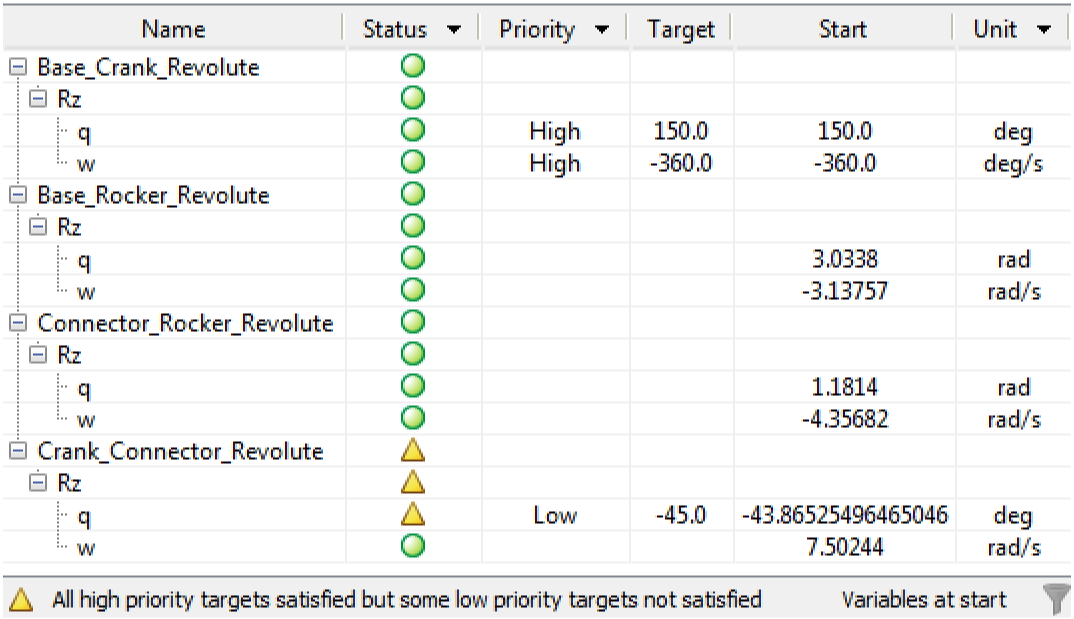
#### Joint Assembly Failure in Four-Bar Linkage with Exceedingly Long Ground Link

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

- Mechanics Explorer — SimMechanics 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.

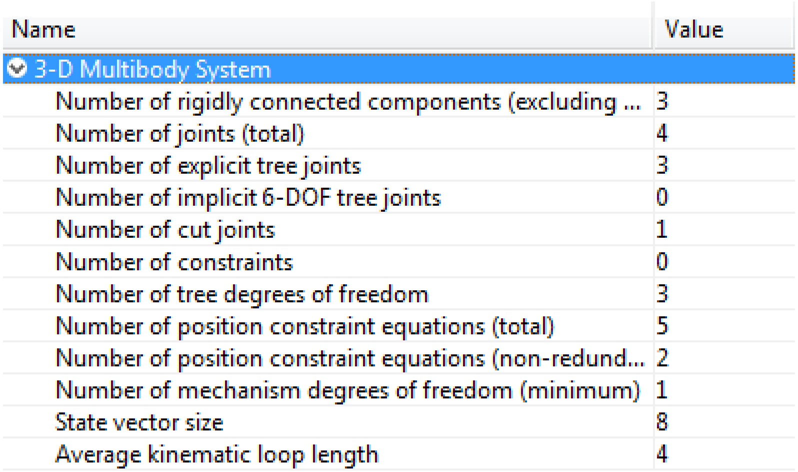


| Name | Status | Priority | Target | Start | Unit |
|---|---|---|---|---|---|
| ⊟ Base_Crank_Revolute | ○ | | | | |
| ⊟ Rz | ○ | | | | |
| q | ○ | High | 150.0 | 150.0 | deg |
| w | ○ | High | -360.0 | -360.0 | deg/s |
| ⊟ Base_Rocker_Revolute | ○ | | | | |
| ⊟ Rz | ○ | | | | |
| q | ○ | | | 3.0338 | 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 |

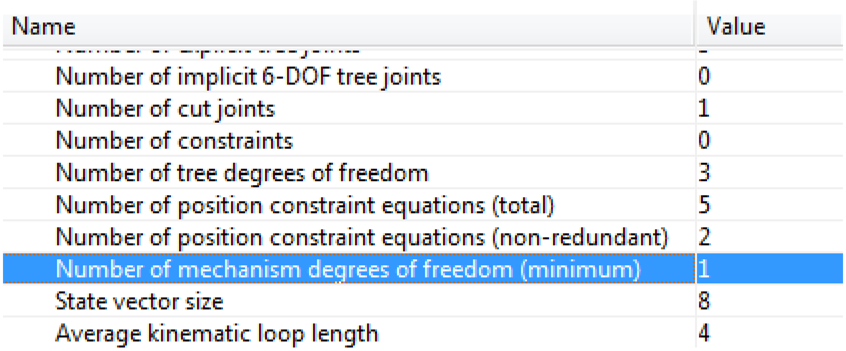
# Mechanism Degrees of Freedom

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 ($f_\mathrm{i} = 1$), 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 SimMechanics 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 **Tools** > **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.

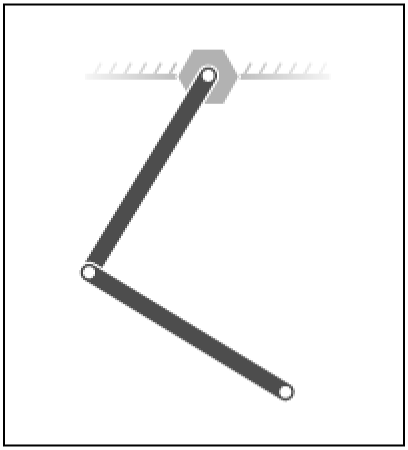


| Name | Value |
|---|---|
| Number of implicit 6-DOF tree joints | 0 |
| Number of cut joints | 1 |
| Number of constraints | 0 |
| Number of tree degrees of freedom | 3 |
| Number of position constraint equations (total) | 5 |
| Number of position constraint equations (non-redundant) | 2 |
| Number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

# Model Double Pendulum

## Model Overview

The double pendulum is a simple multibody system. It contains two links and a pivot mount that connect with joints. This system is nonlinear and does under certain conditions exhibit chaos. In this example, you assemble a double pendulum using custom blocks for the links and the pivot mount. You can later use this model to study the chaotic motion of a double pendulum.



To model the double pendulum, you represent each physical component and constraint using a SimMechanics block. The double pendulum system contains three rigid bodies—one pivot mount and two binary links— that connect in series through a pair of revolute joints. You represent the pivot mount and the binary links using the custom library blocks that you created in previous examples. You represent the two joints using two Revolute Joint blocks from the Joints library.

You can guide model assembly. By specifying joint state targets, you can instruct SimMechanics to assemble a joint in the configuration you want. State targets that you can specify include position and velocity. At times, a state target may conflict with other state targets, or even with other kinematic constraints in the model. In these cases, you can prioritize the most important state targets by assigning them a high priority level. During assembly, if two targets conflict with each other, SimMechanics assembles the high priority target first. To specify both state target values and priority levels, you use the **State Targets** menu of the joint block dialog boxes.

## 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.

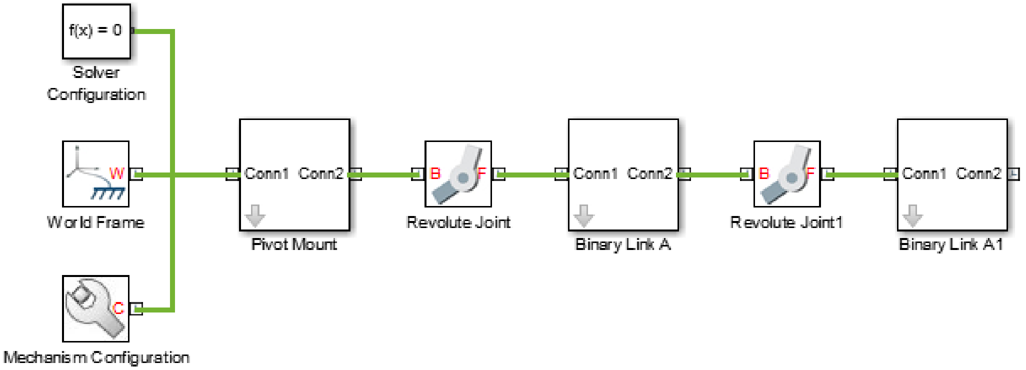
| Library | Block | Quantity |
|---|---|---|
| **Simscape** > **Utilities** | `Solver Configuration` | 1 |
| **SimMechanics** > **Second Generation** > **Utilities** | `Mechanism Configuration` | 1 |
| **SimMechanics** > **Second Generation** > **Frames and Transforms** | `World Frame` | 1 |
| **SimMechanics** > **Second Generation** > **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 rigid body in the double pendulum. See the tutorials in the table for detailed instructions on how to create the blocks.

| Block | Quantity | Modeling Tutorial |
|---|---|---|
| Pivot Mount | 1 | "Model Pivot Mount" on page 2-88 |

| Block | Quantity | Modeling Tutorial |
|---|---|---|
| Binary Link A | 2 | "Model Binary Link" on page 2-74 |

5 Connect the blocks as shown in the figure.



## Guide Model Assembly

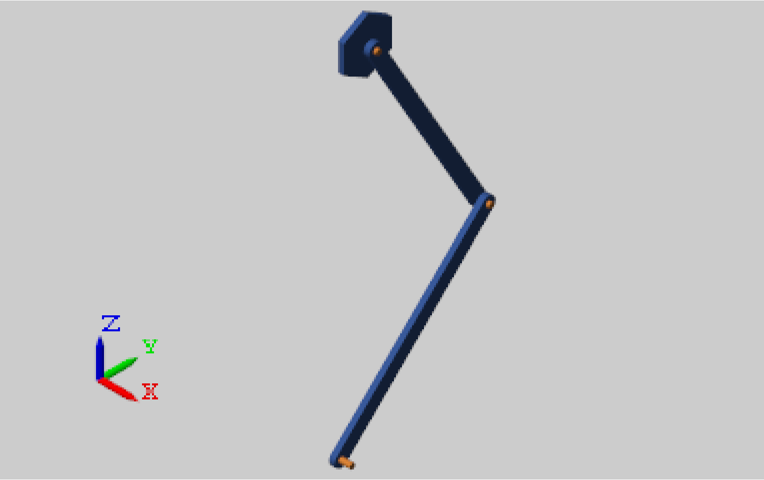
1 In the Revolute Joint block dialog boxes, select **State Targets** > **Specify Position Target**. You can now specify the desired starting positions of the two joints.

2 In **Value**, enter these joint angles.

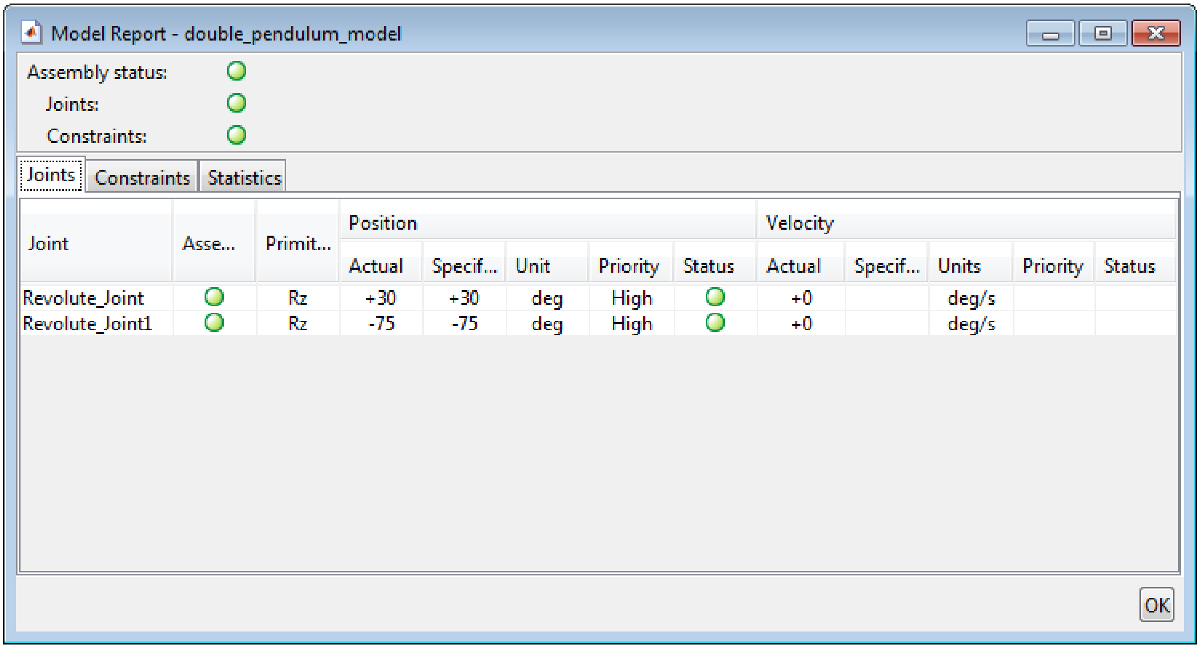
| Block Name | Value (degrees) |
|---|---|
| Revolute Joint | 30 |
| Revolute Joint1 | -75 |

## 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.



Assembly status: ●

Joints: ●

Constraints: ●

| 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 | | |

## 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:

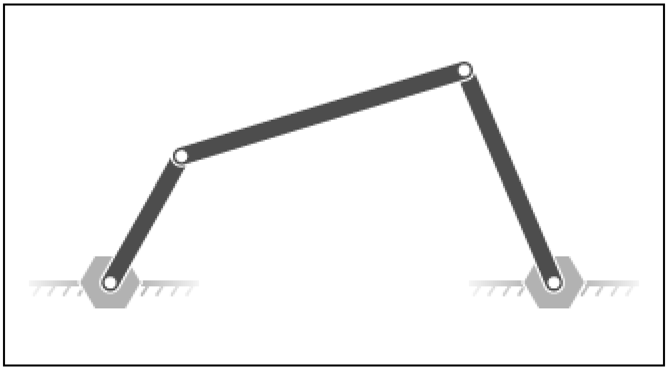
- `smdoc_double_pendulum`

# Model Four Bar

In this section...

## Model Overview

The four-bar linkage is a planar closed-loop linkage used extensively in mechanical machinery. This linkage has four coplanar bars that connect end-to-end with four revolute joints. In this example, you model a four-bar linkage using the Binary Link and Pivot Mount custom blocks that you created in previous examples. For an advanced application of the four-bar linkage, see the bucket actuating mechanism of the Backhoe featured example.

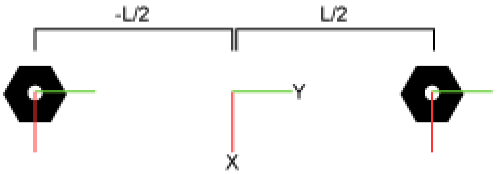


## Modeling Approach

To model the four-bar linkage, you represent each physical component with a SimMechanics block. The linkage in this example has five rigid bodies—three binary links and two pivot mounts—that connect in a closed loop through four revolute joints. Two of the binary links have one peg and one hole. The third binary link has two holes. The fourth link is implicit: the fixed distance between the two coplanar pivot mounts represents this link.

You represent the binary links and pivot mounts using the custom library blocks that you created in previous examples. You represent the four revolute joints using four `Revolute Joint` blocks from the SimMechanics Joints library.

The two pivot mounts connect rigidly to the world frame. For this reason, the implicit link acts as the ground link. Two Rigid Transform blocks provide the rigid connection between the two pivot mounts and the World frame. A translation offset in each Rigid Transform block displaces the two pivot mounts symmetrically along the world frame Y axis.



To guide model assembly, you can specify the desired initial state for one or more joints in the model. To do this, you use the **State Targets** menu of the joint blocks. The state targets that you can specify are the joint position and velocity. These are angular quantities in revolute joints. You can specify state targets for all but one of the joints in a closed loop.

## Build Model

To model the four-bar linkage:

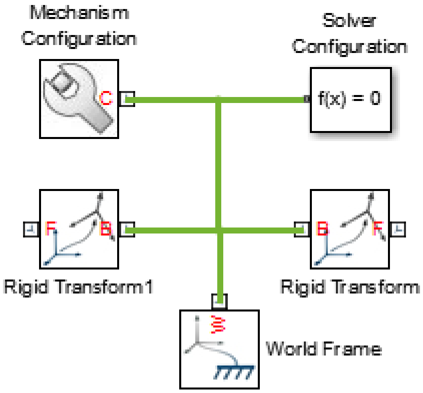
1 Start a new model.

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 |

| Library | Block | Quantity |
|---|---|---|
| **SimMechanics > Second Generation > Utilities** | `Mechanism Configuration` | 1 |
| **SimMechanics > Second Generation > Frames and Transforms** | `World Frame` | 1 |
| **SimMechanics > Second Generation > 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 World Frame block.



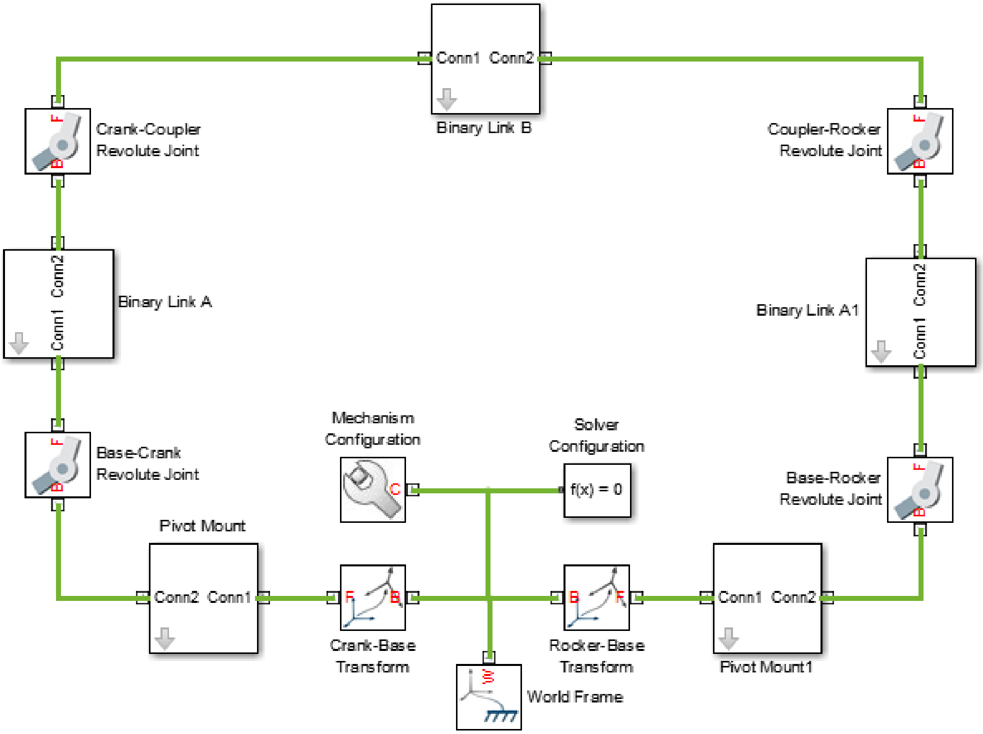
4 From the **SimMechanics** > **Second Generation** > **Joints** library, drag four Revolute Joint blocks into the model.

5 At the MATLAB command prompt, enter `smdoc_compound_rigid_bodies`. A custom library with compound rigid body blocks opens up.

6 From the smdoc_compound_rigid_bodies library, drag these blocks. Each block represents a rigid body present in the four bar assembly. See the tutorials in the table for instructions on how to create the blocks.

| Block | Quantity | Modeling Tutorial |
|---|---|---|
| Pivot Mount | 2 | "Model Pivot Mount" on page 2-88 |
| Binary Link A | 2 | "Model Binary Link" on page 2-74 |
| Binary Link B | 1 | "Model Two-Hole Binary Link" on page 2-82 |

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

| 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

The model is now complete. You can now specify the desired initial state for one or more joints in the model. In this example, you specify an initial angle of 30° for the Base-Crank joint. To do this:

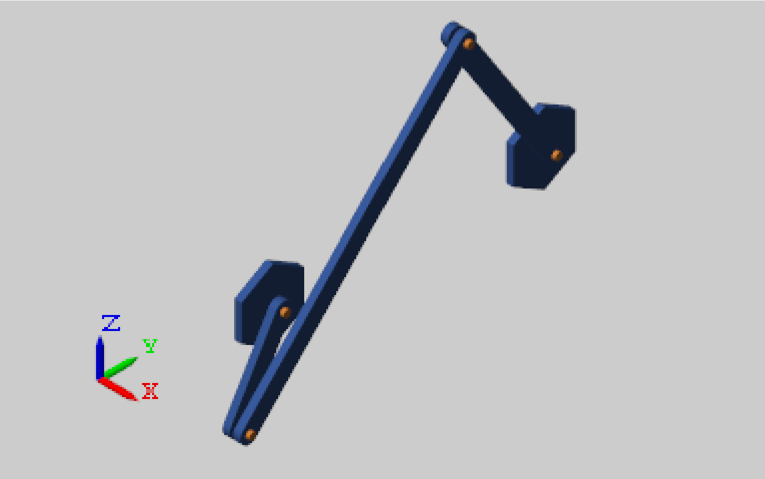
1 Double-click the Base-Crank Revolute Joint block.

2 In the block dialog box, expand **State Targets** and select **Position**.

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

4 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. The figure shows the static display that you see in Mechanics Explorer after updating the model. To obtain the view shown in the figure, in the Mechanics Explorer toolstrip select the isometric view button .

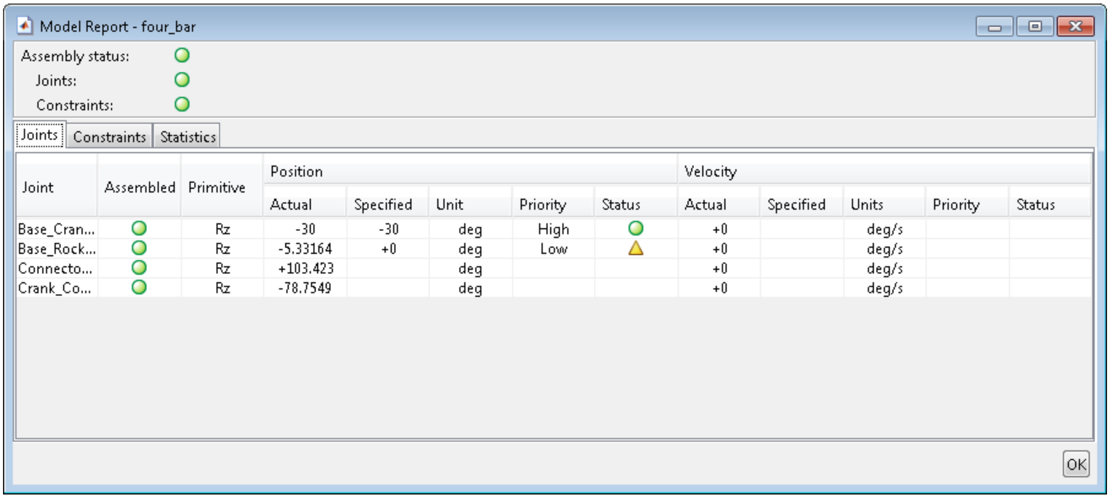


You can guide assembly so that the four-bar linkage assembles in an open configuration instead. To do this, you must specify a position state target for at least one more joint. You do not have to specify this target precisely. If you have a general idea of what the target should be, you can enter an approximate value and select a low priority level for that target.

Closed-loop kinematic chains like the four-bar linkage are especially vulnerable to assembly issues. Even when the model assembles, SimMechanics may fail to meet one or more state targets. You can check the assembly status of the model and of the joints using the Model Report utility:

1. In the Mechanics Explorer menu bar, select **Tools** > **Model Report**.
2. Examine the model report for red squares or yellow triangles. These shapes identify issues in the assembly or in the joint state targets.

The figure shows the model report for the four bar linkage in the open configuration. A green circle indicates that SimMechanics satisfied the Base-Crank Revolute Joint state target precisely. A yellow circle indicates that SimMechanics satisfied the Base-Rocker Revolute Joint state target approximately.



Model Report - four_bar

Assembly status: 🟢
Joints: 🟢
Constraints: 🟢

| Joint | Assembled | Primitive | Position | | | | | Velocity | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Actual | Specified | Unit | Priority | Status | Actual | Specified | Units | Priority | Status |
| Base_Cran... | 🟢 | Rz | -30 | -30 | deg | High | 🟢 | +0 | | deg/s | | |
| Base_Rock... | 🟢 | Rz | -5.33164 | +0 | deg | Low | ⚠ | +0 | | deg/s | | |
| Connecto... | 🟢 | Rz | +103.423 | | deg | | | +0 | | deg/s | | |
| Crank_Co... | 🟢 | Rz | -78.7549 | | deg | | | +0 | | deg/s | | |

## 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`

# Find and Fix Aiming-Mechanism Assembly Errors

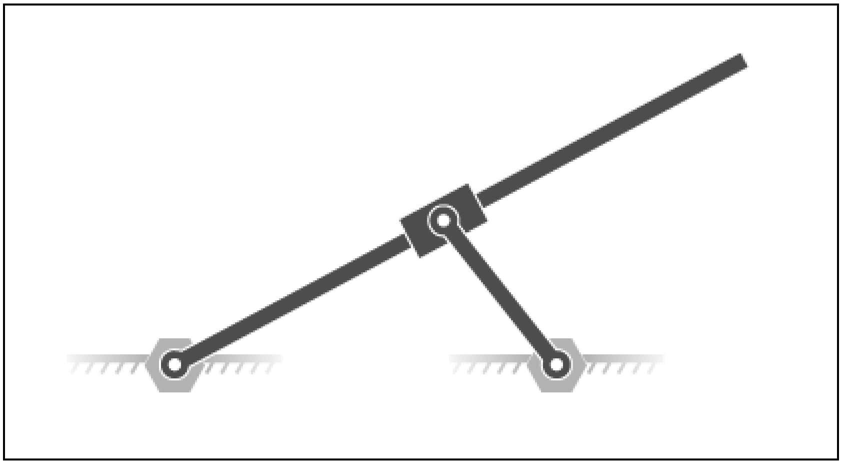
**In this section...**

## 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, SimMechanics 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.

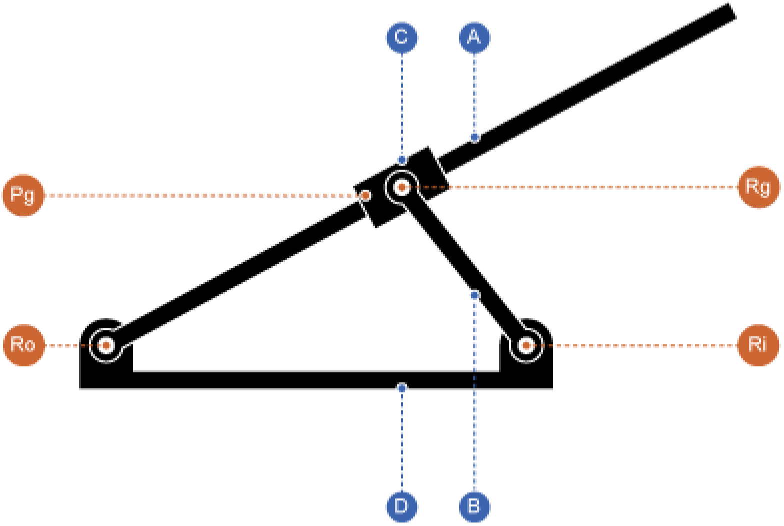
In this example, you identify the assembly error source in an aiming mechanism model using Model Report. Then, using Mechanics Explorer, you determine how to correct that error source. The `sm_dcrankaim_assembly_with_error` featured example provides the basis for this example.



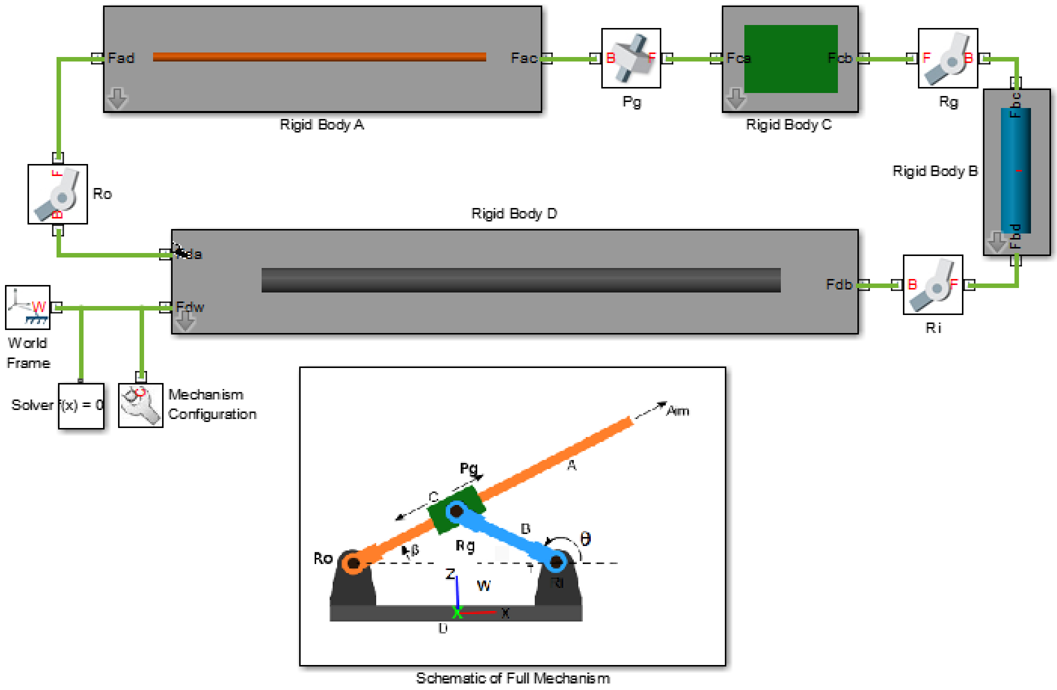
## Explore Model

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 rigid bodies, labeled A-D. These rigid 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.

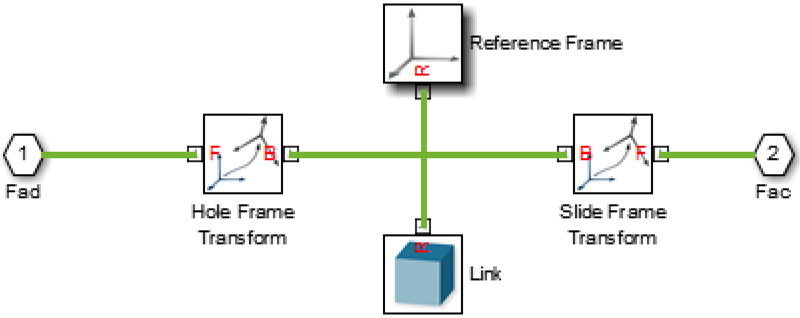


The model represents the components of this system using blocks. Each block represents a physical component. A `World Frame` block provides the ultimate reference frame in the model. The figure shows the block diagram that the model uses to represent the double-crank aiming mechanism.



To represent the rigid 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 rigid 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 rigid body subsystem blocks Rigid Body A-D. To examine the block diagram for a rigid 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 rigid 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 rigid 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:

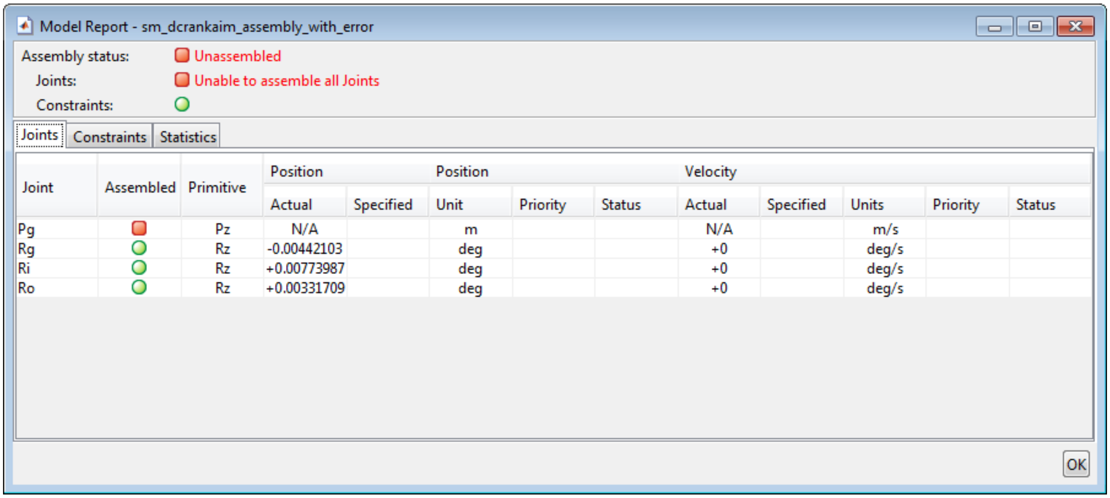
- On the Simulink menu bar, select **Simulation** > **Update Diagram**.

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

## Troubleshoot Assembly Error

Mechanics Explorer provides access to Model Report, a SimMechanics 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.



Model Report - sm_dcrankaim_assembly_with_error

Assembly status: 🟥 Unassembled

Joints: 🟥 Unable to assemble all Joints

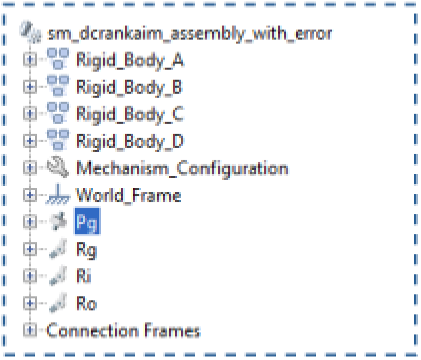
Constraints: 🟢

| | | | Position | | Position | | | Velocity | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Joint | Assembled | Primitive | Actual | Specified | Unit | Priority | Status | Actual | Specified | Units | Priority | Status |
| Pg | 🟥 | Pz | N/A | | m | | | N/A | | m/s | | |
| Rg | 🟢 | Rz | -0.00442103 | | deg | | | +0 | | deg/s | | |
| Ri | 🟢 | Rz | +0.00773987 | | deg | | | +0 | | deg/s | | |
| Ro | 🟢 | Rz | +0.00331709 | | deg | | | +0 | | deg/s | | |

### Identifying Error Root Cause

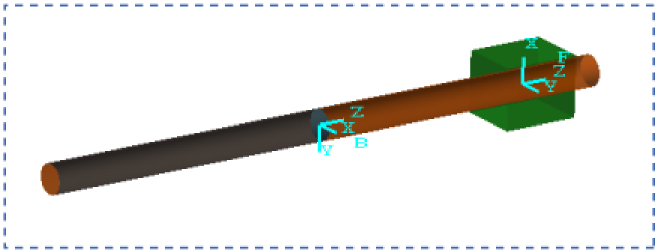
The error message that SimMechanics issued during model update identifies position violation as the root cause of assembly failure. This suggests that the frames connected by joint Pg are improperly aligned. To confirm this hypothesis, check the orientation of these frames in Mechanics Explorer.

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



sm_dcrankaim_assembly_with_error
- Rigid_Body_A
- Rigid_Body_B
- Rigid_Body_C
- Rigid_Body_D
- Mechanism_Configuration
- World_Frame
- Pg
- Rg
- Ri
- Ro
- Connection Frames

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

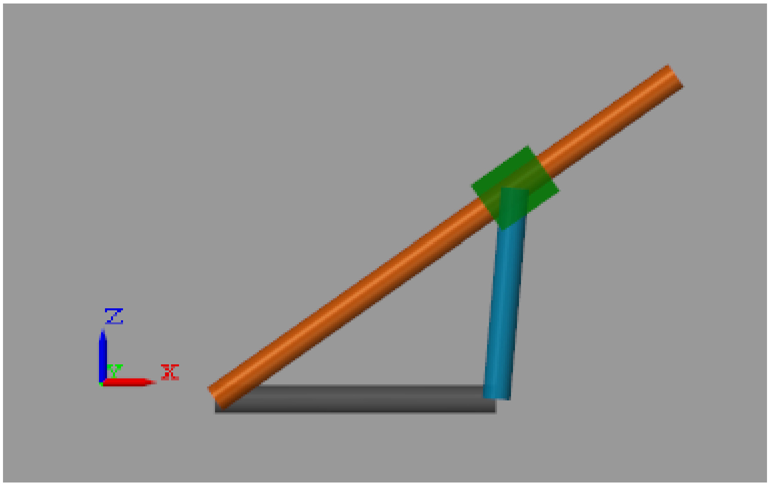
In this example, you apply a rotation transform to the follower frame so that its axes lie parallel to the base frame axes. Alternatively, you could apply an equivalent rotation transform to the base frame. This step enables joint Pg, and hence the model itself, to assemble successfully.

1 In the tree pane of Mechanics Explorer, right-click the Pg node and select `Go To Block`. SimMechanics 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**.

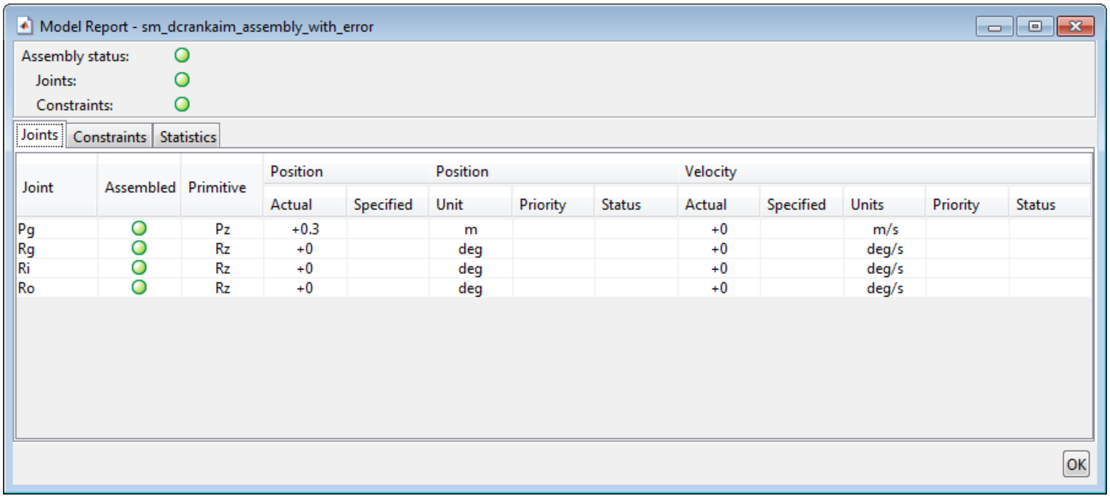
| Parameter | New Value |
|---|---|
| **Rotation** > **Pair 2** > **Follower** | +X |
| **Rotation** > **Pair 2** > **Base** | +Y |

## 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, 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.



Assembly status: 🟢
Joints: 🟢
Constraints: 🟢

| Joint | Assembled | Primitive | Position | | Position | | | Velocity | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Actual | Specified | Unit | Priority | Status | Actual | Specified | Units | Priority | Status |
| Pg | 🟢 | Pz | +0.3 | | m | | | +0 | | m/s | | |
| Rg | 🟢 | Rz | +0 | | deg | | | +0 | | deg/s | | |
| Ri | 🟢 | Rz | +0 | | deg | | | +0 | | deg/s | | |
| Ro | 🟢 | Rz | +0 | | deg | | | +0 | | deg/s | | |

## Related Examples

- “Model Double Pendulum” on page 3-14
- “Model Four Bar” on page 3-19

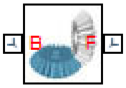
## More About

- “Modeling Joint Connections” on page 3-2

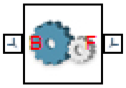
# Gear Constraints

**In this section...**

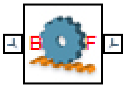
You can represent gear constraints in a multibody model. To do this, SimMechanics provides a Gears and Couplings library. This library contains gear blocks that you can use to constrain the motion of two rigid body frames. The figure shows the gear blocks that the library provides.



Bevel Gear
Constraint



Common Gear
Constraint



Rack and Pinion
Constraint

## Gear Types

The **Gears and Couplings** > **Gears** library provides blocks for modeling gears. The table summarizes the gears you can model with these blocks.

| Block | Description |
|---|---|
| `Common Gear Constraint` | Transfer rotational motion between two frames spinning about parallel axes |

| Block | Description |
|---|---|
| `Rack and Pinion Constraint` | Transfer rotational motion at a pinion into translational motion at a rack and vice-versa. |
| `Bevel Gear Constraint` | Transfer rotational motion between two frames spinning about arbitrarily aligned axes. |

## Featured Examples

SimMechanics provides two featured examples that highlight the use of gear blocks. The table lists these examples. To open an example model, at the MATLAB command line, enter the model name, e.g., `sm_cardan_gear`.

| Featured Example | Model Name | Gear Blocks Used |
|---|---|---|
| Cardan gear | `sm_cardan_gear` | Common Gear Constraint |
| Windshield wiper | `sm_windshield_wiper` | Rack and Pinion Constraint |
| Robotic wrist | `sm_robotic_wrist` | Bevel Gear Constraint |

Open the models and examine the blocks for examples of how to connect the gear blocks and specify their parameters.

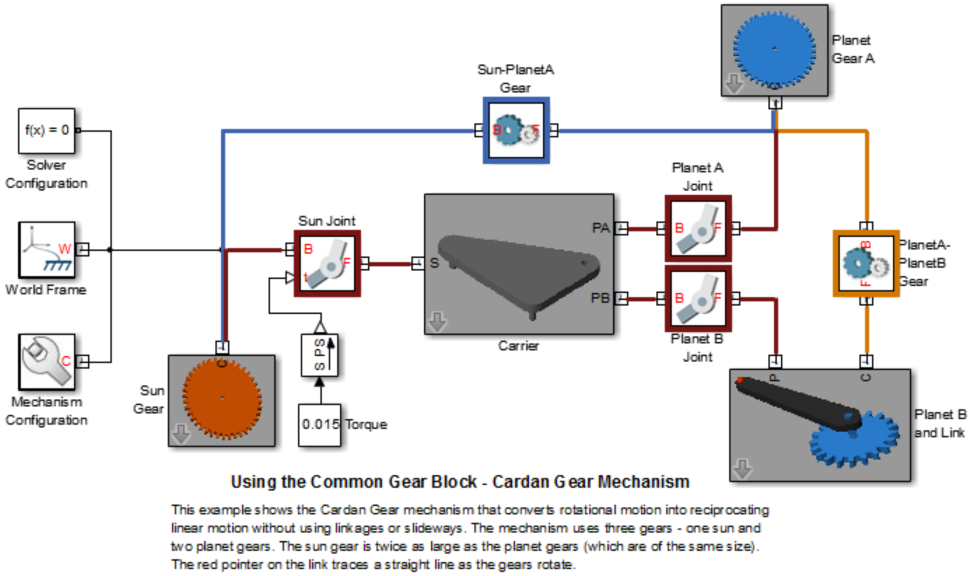
## Inertia, Geometry, and Efficiency

Each gear block represents a kinematic constraint between two rigid body frames. This constraint does not account for the effects of inertia or power transmission losses. It also does not provide gear visualization. If necessary, consider modeling these effects using other SimMechanics and Simscape blocks. To represent gear inertia and geometry, use the `Solid` block.

## Using Gear Blocks

To apply a gear constraint between two rigid bodies, connect the base and follower frames of the gear block to the rigid body frames that you want to constrain. Then, open the gear block dialog box and specify the gear parameters. Parameters can include gear dimensions and ratio.

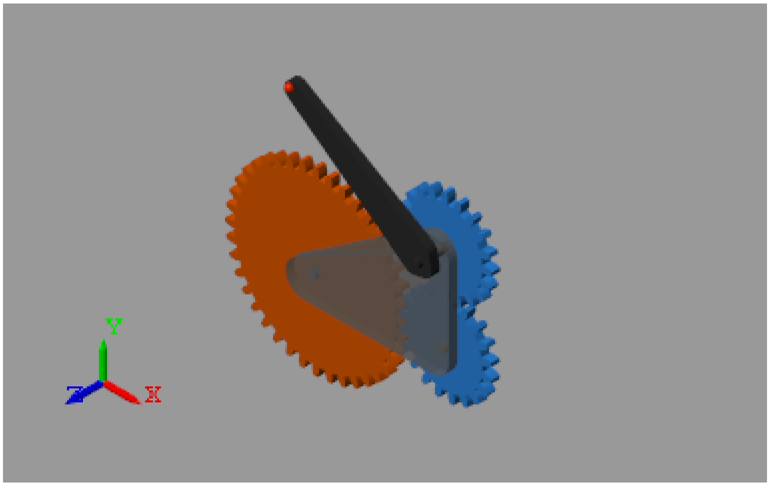
Featured example `sm_cardan_gear` illustrates an application of the Common Gear block. In this model, two Common Gear blocks connect three gear rigid bodies. Subsystems Planet Gear A, Planet B and Link, and Sun Gear represent these rigid bodies. One Common Gear block constrains the motion of subsystem Planet Gear A with respect to subsystem Sun Gear. The other Common Gear block constrains the motion of subsystem Planet B and Link with respect to subsystem Planet Gear A. The figure shows the block diagram of this model.



**Using the Common Gear Block - Cardan Gear Mechanism**

This example shows the Cardan Gear mechanism that converts rotational motion into reciprocating linear motion without using linkages or slideways. The mechanism uses three gears - one sun and two planet gears. The sun gear is twice as large as the planet gears (which are of the same size). The red pointer on the link traces a straight line as the gears rotate.

So that the three gear subsystems can rotate with respect to each other, the model includes three Revolute Joint blocks. Each Revolute Joint block provides one rotational degree of freedom between one gear subsystem and the gear carrier—a rigid body that holds the three rotating gears. The figure shows the Mechanics Explorer display of this model.



## Assembling Rigid Bodies with Gear Constraints

To assemble successfully, a model must satisfy the constraints that a gear block imposes. These include distance and orientation constraints that are specific to each block. The table summarizes these constraints.

| Gear Constraint | Description |
|---|---|
| Frame Distance | The model must maintain a fixed distance between the base and follower gear frames. The value of this distance depends on the gear block that you use. |
| Frame Orientation | The model must orient the base and follower gear frames according to rules that are specific to each block. |

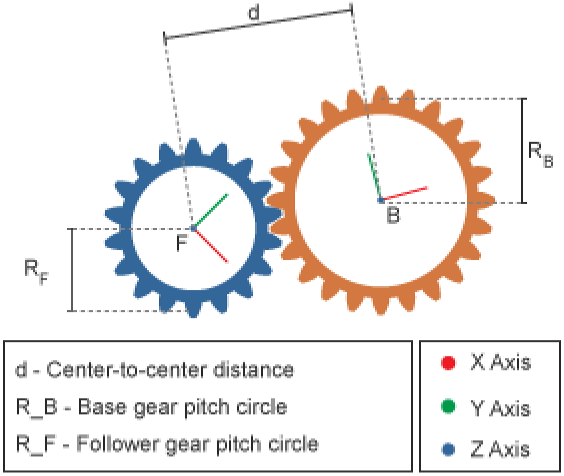
The rigid body frames that the gear block connects must have the proper number and type of degrees of freedom. For a Common Gear block, the frames must have two rotational degrees of freedom with respect to each other. For a Rack and Pinion block, the frames must have one translational and one rotational degree of freedom with respect to each other. You provide these degrees of freedom using joint blocks.

- Use joint blocks with revolute primitives to provide the rotational degrees of freedom.

- Use joint blocks with prismatic primitives to provide the translational degrees of freedom.

## Common Gear Assembly and Simulation

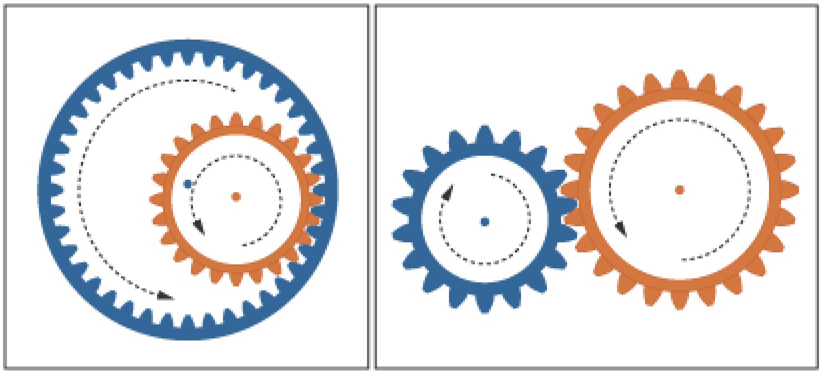
During assembly, the Common Gear block requires that the base and follower frame Z axes align. These are the rotation axes of the two gear frames. Failure to align the Z axes of the two gear frames results in assembly failure during model update. The figure illustrates the common gear rigid bodies, frames, and distance constraints.


d
R_B
B
F
R_F
d - Center-to-center distance
R_B - Base gear pitch circle
R_F - Follower gear pitch circle
● X Axis
● Y Axis
● Z Axis

Connect the gear rigid bodies to joints possessing one (or more) revolute joint primitives. The rotational axis of the revolute primitive must align with the Z axis of the gear frame that it connects to. This ensures that the gear frames possess a rotational degree of freedom about the correct axis (Z).

### Common Gear Types

With the Common Gear block, you can represent internal and external gear constraints. If the gear constraint is internal, the gear frames rotate in the same direction. If it is external, the gear frames rotate in opposite directions. The figure illustrates the two common gear types that you can represent and their relative rotation senses.



### Gear Dimensions

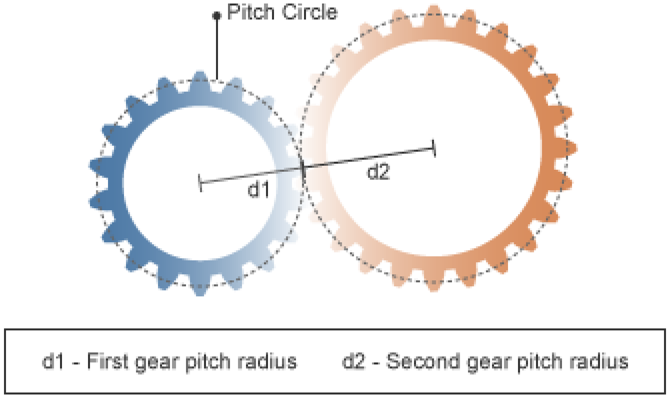
In the block dialog box, you specify the gear dimensions. Depending on the specification method that you choose, you can specify the center-to-center distance between gears or the pitch circle radii. During model assembly, the Common Gear block imposes this distance constraint between the two gear frames. This ensures that the gear assembles properly or, if issues arise, that you can correct any assembly issues early on.

You specify the gear relative sizes in the block dialog box. If you select the `Center Distance and Ratio` specification method, the gear ratio specifies which of the two gears is the larger one. If the gear ratio is greater than one, the follower gear is the larger gear. If the gear ratio is smaller than one, the base gear is the larger gear.

If you specify an internal gear type, the larger gear is the ring gear. A gear ratio greater than unity makes the follower gear the ring gear. A gear ratio smaller than unity makes the base gear the ring gear.

### Gear Pitch Circles

The pitch circle of a gear is an imaginary circle that passes through the contact point between gears. The pitch radius of a gear is the radius of this imaginary circle. The figure illustrates the pitch circles of two meshing gears and their pitch radii. These are the gear radii that you enter in the block dialog box when you select the `Pitch Circle Radii` specification method.



### Simulation

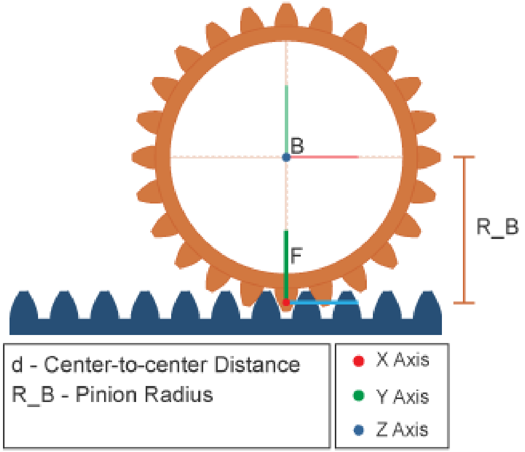
During simulation, the Common Gear block requires that the model maintain the proper distance between gear frames. This distance must equal either the center-to-center distance or the sum of base and follower gear pitch radii that you specify in the block dialog box. The structure of the model must be such that the gears maintain this distance between them. Failure to maintain this distance results in an error during simulation.

In the Cardan Gear example, the Carrier rigid body fixes the distances between the three gears. As long as these distances match the gear dimensions that you specify in the block dialog box, the model should simulate without an issue.

## Rack and Pinion Assembly and Simulation

The base frame of the Rack and Pinion block represents the pinion. It can rotate about its Z axis. The follower frame of the same block represents the rack. It can translate along its Z axis. During assembly, the Rack and Pinion block requires that the base and follower frame Z axes be mutually orthogonal.

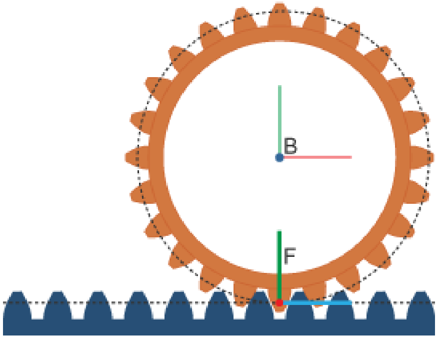
When the gear is in its zero configuration—a configuration in which the angle and displacement between base and follower frames are taken as zero—the follower frame Z axis is also parallel to the base frame X axis, and base and follower frame Y axes are parallel to each other. The follower frame origin lies along the base frame -Y axis, at a distance equal to the base gear pitch radius. The figure illustrates these constraints.



To ensure the rack and pinion can move with respect to each other, you must connect the rack and pinion rigid bodies to joints blocks. The joint block on the rack side must have one (or more) prismatic primitives. At least one primitive axis must align with the Z axis of the follower gear frame. The joint block on the pinion side must have one (or more) revolute primitives. At least one revolute axis must align with the Z axis of the base gear frame.

### Gear Pitch Circles

The pitch circle of a rack and pinion gear is the imaginary circle that passes through the contact point between the pinion and the rack. The pitch radius is the radius of this imaginary circle. The figure illustrates the pitch circle for a rack and pinion. This is the circle whose radius you enter in the block dialog box.



### Simulation

During simulation, the Rack and Pinion block requires that the model maintain the proper distance between gear frames. The distance between the base frame origin (pinion) and the follower frame Z axis must equal the pinion radius. Failure to maintain this distance between gear frames results in a simulation error.

# Model Rack and Pinion

**In this section...**

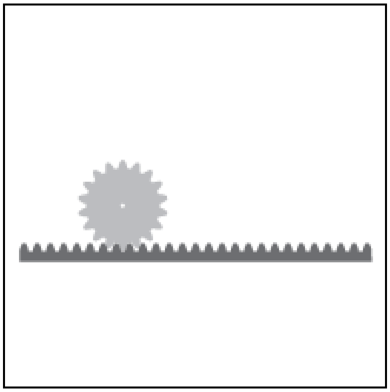
## Model Overview

In this tutorial, you model a kinematic constraint between rack and pinion components. The constraint causes the two components to move in sync such that a pinion rotation corresponds to a rack translation:

$$V_F = \omega_B \cdot R_B,$$

where:

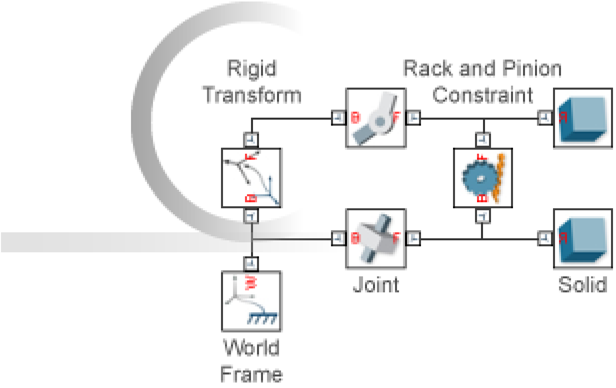
- $V_F$ is the rack translational velocity.
- $\omega_B$ is the pinion rotational velocity.
- $R_B$ is the radius of the pinion pitch circle, an imaginary circle intersecting the contact point between rack and pinion teeth.



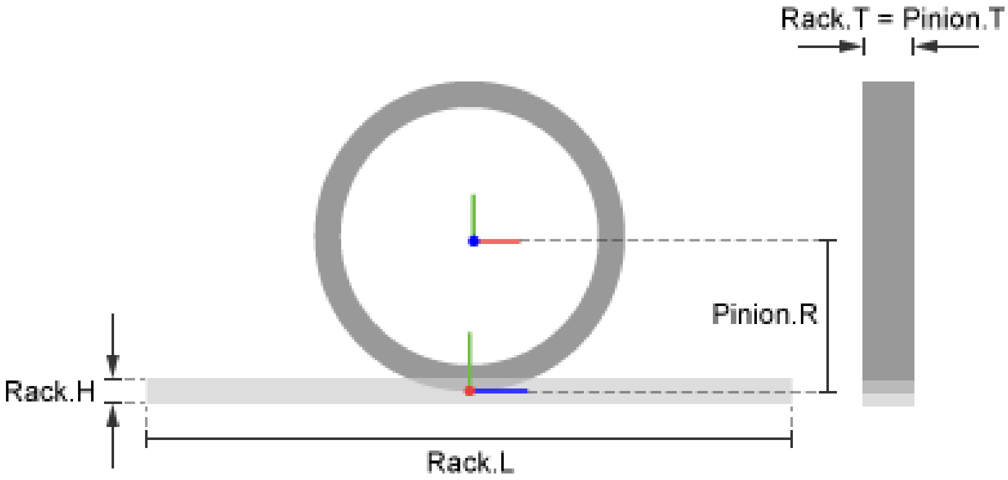
The model uses three key blocks:

- Solid — Specify rack and pinion geometry, inertia, and color
- Joint — Provide motion degrees of freedom to the rack and pinion components. These degrees of freedom enable the rack to translate and the pinion to rotate with respect to the world frame.
- Rack and Pinion Constraint — Constrain the motion of the rack and pinion components so that they move in a meshed configuration.

The figure shows how these blocks connect in the model.



For simplicity, the rack has a brick shape and the pinion has a cylinder shape. These shapes depend on several dimensions, shown in the figure. You specify each dimension using a MATLAB variable. After model assembly, you can add detail to the component shapes. For example, you can specify an involute tooth profile for the rack and pinion.



**Rack and Pinion Dimensions**

## Model Pinion

1 Start a new model.

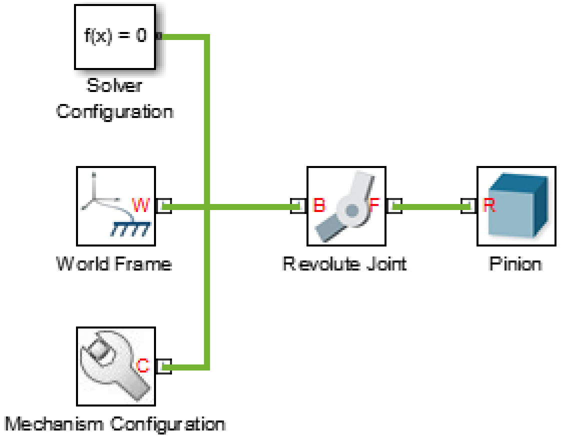
2 Add these blocks to the model.

| Library | Block |
|---|---|
| **Simscape > Utilities** | `Solver Configuration` |
| **SimMechanics > Second Generation > Frames and Transforms** | `World Frame` |
| **SimMechanics > Second Generation > Utilities** | `Mechanism Configuration` |

| Library | Block |
|---|---|
| **SimMechanics > Second Generation > Joints** | `Revolute Joint` |
| **SimMechanics > Second Generation > Body Elements** | `Solid` |

The Solid block specifies the component geometry, inertia, and color. The joint block provides the component its motion degrees of freedom—in this case, one rotational degree of freedom with respect to the world frame.

**3** Connect and name the blocks as shown in the figure. Port frames joined by a connection line are coincident in space.



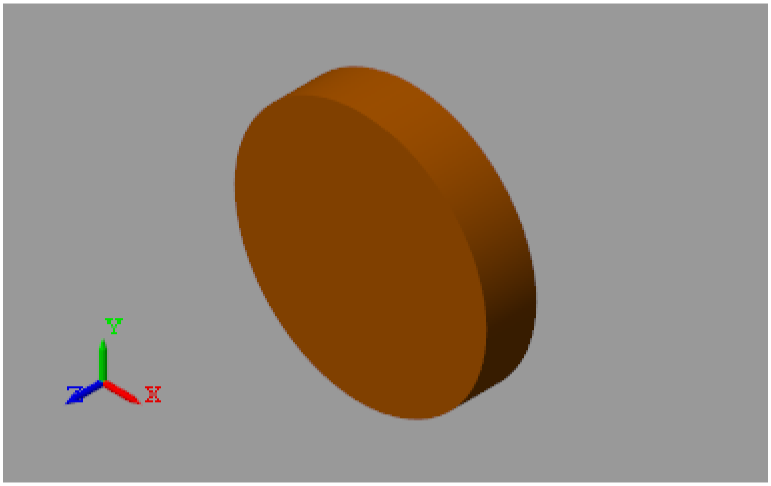
**4** In the Pinion block dialog box, specify geometry, inertia, and color.

| Parameter | Enter or Select |
|---|---|
| **Geometry > Shape** | `Cylinder` |
| **Geometry > Radius** | `Pinion.R`, units of `cm` |
| **Geometry > Length** | `Pinion.T`, units of `cm` |
| **Inertia > Density** | `Rho` |
| **Graphic > Visual Properties > Color** | `Pinion.RGB` |

5 In the model workspace, initialize the MATLAB variables you entered in the block dialog boxes:

```
% Common Parameters
Rho = 2700; % Mass density of both rack and pinion components

% Pinion Parameters
Pinion.R = 10;
Pinion.T = 4;
Pinion.RGB = [0.8, 0.4, 0];
```

6 Update the block diagram. You can do this by selecting **Simulation > Update Diagram**. Mechanics Explorer opens with a 3-D view of the pinion gear. To obtain the view shown in the figure, in the Mechanics Explorer toolstrip set the **View convention** parameter to `Y up (XY Front)`. Then, select the isometric view button .



## Model Rack

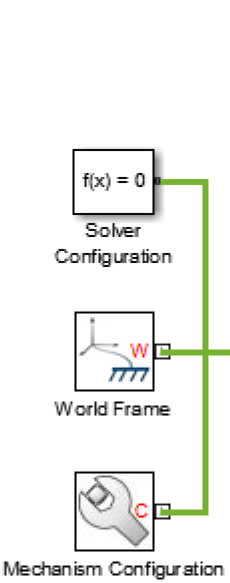
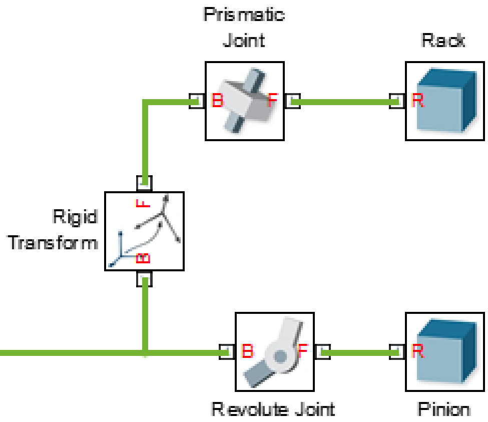
1 Add these blocks to the model.

| Library | Block |
|---|---|
| **SimMechanics > Second Generation > Frames and Transforms** | `Rigid Transform` |

| Library | Block |
|---|---|
| **SimMechanics > Second Generation > Joints** | `Prismatic Joint` |
| **SimMechanics > Second Generation > Body Elements** | `Solid` |

The Rigid Transform block sets the rack position and pose with respect to the pinion. These quantities must satisfy the assembly conditions later imposed by the Rack and Pinion Constraint block.

**2** Connect and name the blocks as shown in the figure.

**3** In the Rack block dialog box, specify geometry, inertia, and color.

| Parameter | Select or Enter |
|---|---|
| **Geometry > Shape** | `Brick` |
| **Geometry > Dimensions** | `[Rack.T,Rack.H,Rack.L]`, units of `cm` |

| Parameter | Select or Enter |
|---|---|
| **Inertia > Density** | `Rho` |
| **Graphic > Visual Properties > Color** | `Rack.RGB` |

4 In the Rigid Transform block dialog box, specify the rack position and pose with respect to the pinion.

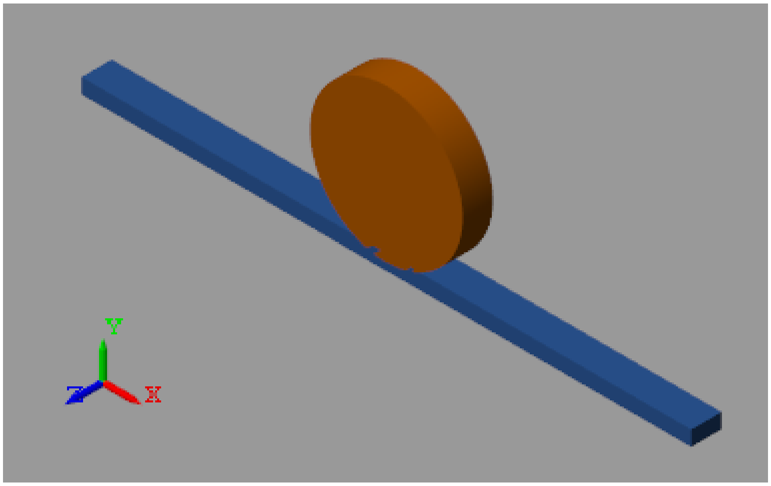
| Parameter | Select or Enter |
|---|---|
| **Rotation > Method** | `Standard Axis` |
| **Rotation > Axis** | `+Y` |
| **Rotation > Angle** | `90` |
| **Translation > Method** | `Standard Axis` |
| **Translation > Axis** | `-Y` |
| **Translation > Offset** | `Pinion.R` in units of `cm` |

The rotation transform makes the rack and pinion Z axes mutually orthogonal while keeping the Y axes parallel. The translation transform separates the rack and pinion frame origins by a distance equal to the pinion pitch radius. These transforms satisfy the assembly conditions imposed by the Rack and Pinion Constraint block.

5 In the model workspace, initialize the new MATLAB variables entered in the block dialog boxes:

```
% Rack Parameters
Rack.L = 80;
Rack.H = 2;
Rack.T = Pinion.T;
Rack.RGB = [0.2, 0.4, 0.7];
```

6 Update the block diagram. Mechanics Explorer displays a 3-D view of the rack and pinion assembly. Examine the assembly from different viewpoints and verify it is accurate. You can view the rack and pinion frames by clicking the frame button in the Mechanics Explorer tool bar.

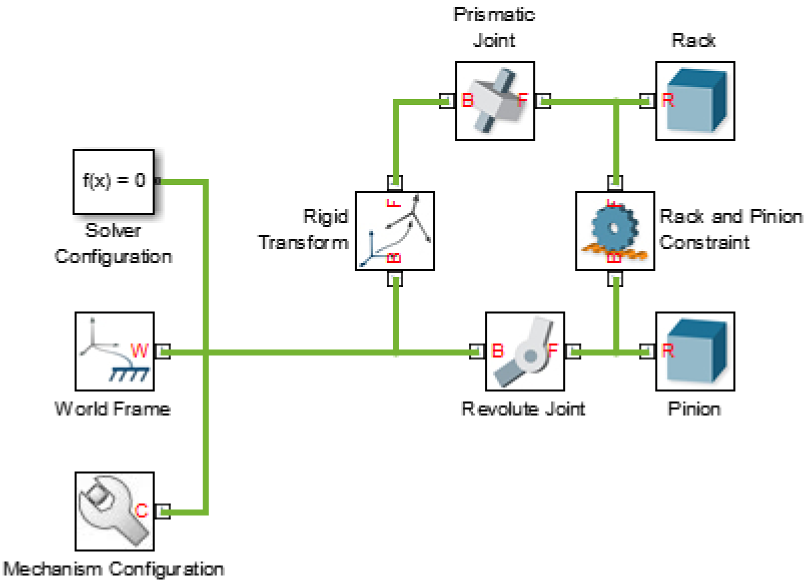


## Add Rack and Pinion Constraint

The model is nearly complete. It remains to constrain the motion of the rack and pinion components. You add this kinematic constraint using the Rack and Pinion Constraint block.

1 From the **Gears and Couplings** > **Gears** library, drag a `Rack and Pinion Constraint` block to the model.

2 Connect the block as shown in the figure. The follower frame port connects to the Rack block, while the base frame port connects to the Pinion block.



3 In the dialog box of the Rack and Pinion Constraint block, enter `Pinion.R` in **Pinion Radius**.

4 Update the block diagram. Mechanics Explorer shows a 3-D display of the updated rack and pinion assembly. Assembly errors due to gear constraints become evident at this stage. If SimMechanics issues an error message, correct the model before attempting to run the simulation.

## Actuate Model

1 In the Revolute Joint block dialog box, for **Z Revolute Primitive (Rz)** > **Actuation** > **Torque**, select `Provided by Input`.

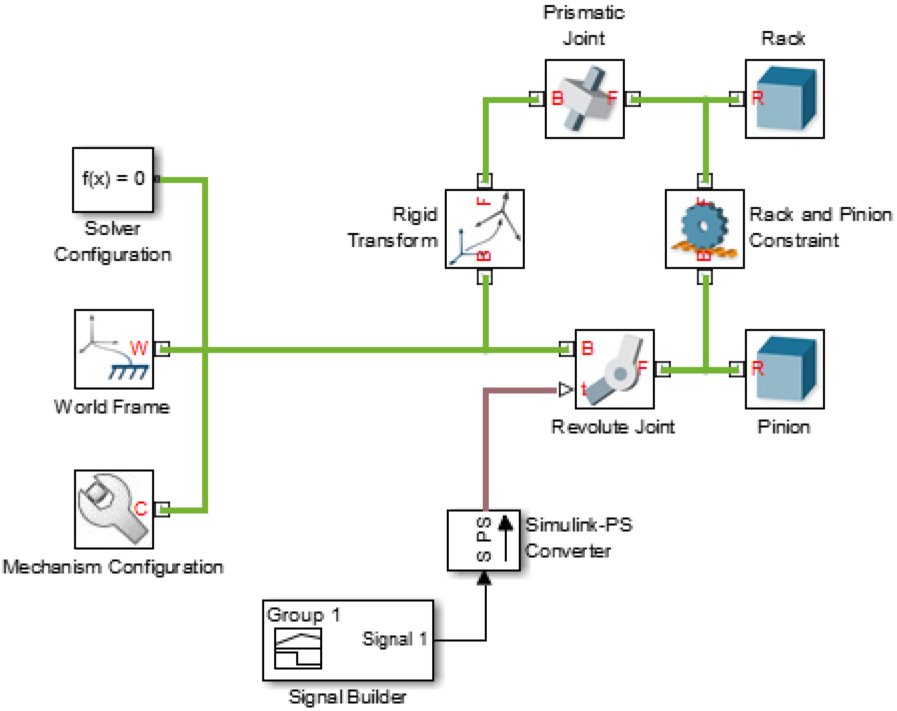
The block exposes a physical signal input port. You use this port to specify a driving torque acting on the pinion. During simulation, this torque will be the source of motion in the model.

2 Drag these blocks to specify and process the input torque signal.

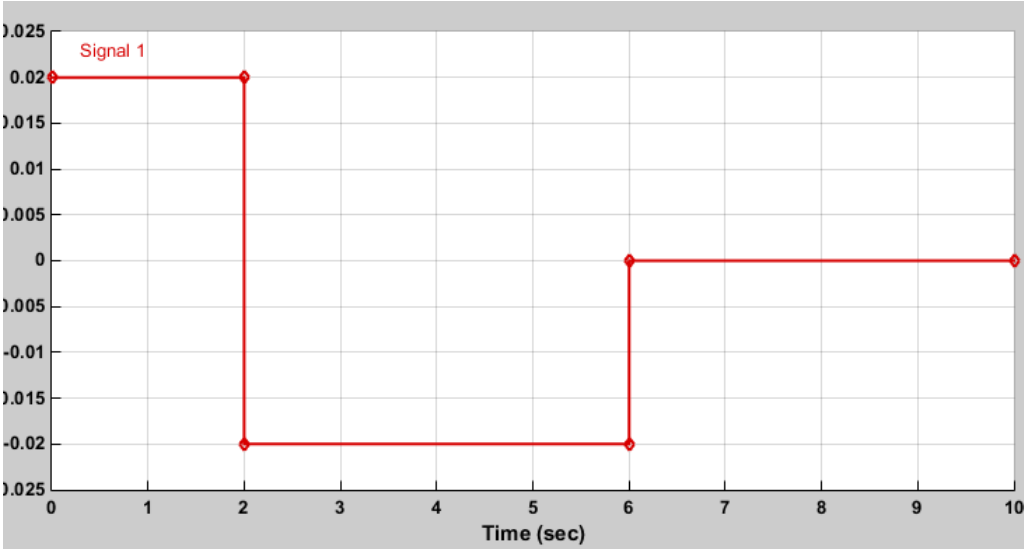
| Library | Block |
|---|---|
| **Simulink** > **Sources** | `Signal Builder` |
| **Simscape** > **Utilities** | `Simulink-PS Converter` |

The Simulink-PS Converter block converts the Simulink input signal into a physical signal compatible with SimMechanics blocks. It also provides signal filtering, which enables you to smooth discontinuous signals.

**3** Connect the blocks as shown in the figure.



**4** In the Signal Builder block dialog box, draw the input signal as shown in the figure. This signal starts with a positive torque followed by a negative torque. The positive torque causes the pinion to rotate counterclockwise about the base frame +Z axis and the rack to translate along the follower frame +Z axis.

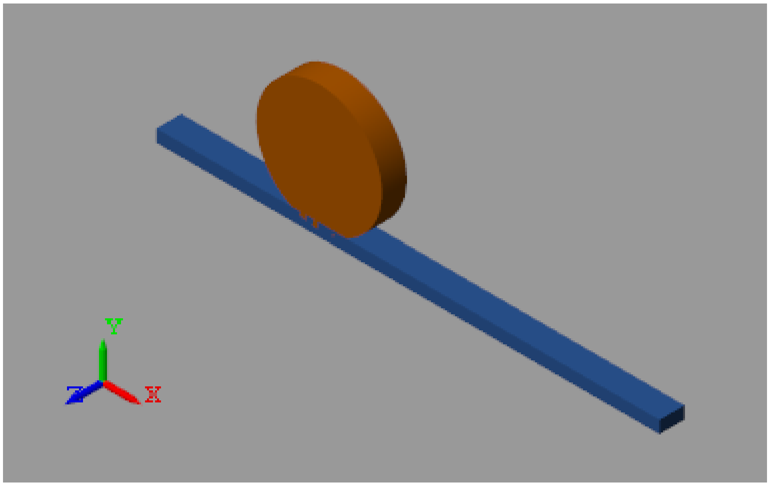


5 In the Simulink-PS Converter block dialog box, in the **Input Handling** tab, specify second-order filtering with a time constant of `0.1 s`. This filter helps to smooth the discontinuities of the input signal.

| Parameter | Select or Enter |
|---|---|
| **Filtering and derivatives** | `Filter input` |
| **Input filtering order** | `Second-order filtering` |
| **Input filtering time constant (in seconds)** | `0.1` |

## Simulate Model

Run the simulation. You can do this by selecting **Simulation** > **Run**. Mechanics Explorer plays a physics-based animation of the rack and pinion assembly. To better see motion during playback, select the frame button in the Mechanics Explorer tool bar.



## Open Complete Model

To view a complete model of the rack and pinion mechanism, at the MATLAB command prompt enter:

```
smdoc_rack_pinion_c
```

For an example using a helper function to generate a rough involute tooth profile, enter

```
smdoc_rack_pinion_d
```

# Model Planetary Gear Train

**In this section...**

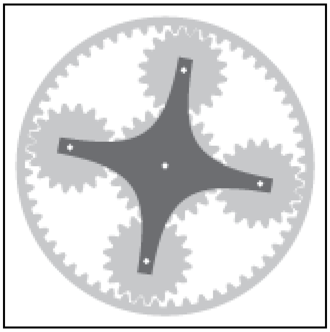
## Model Overview

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 SimMechanics, 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 rigid bodies:

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

Each rigid 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.



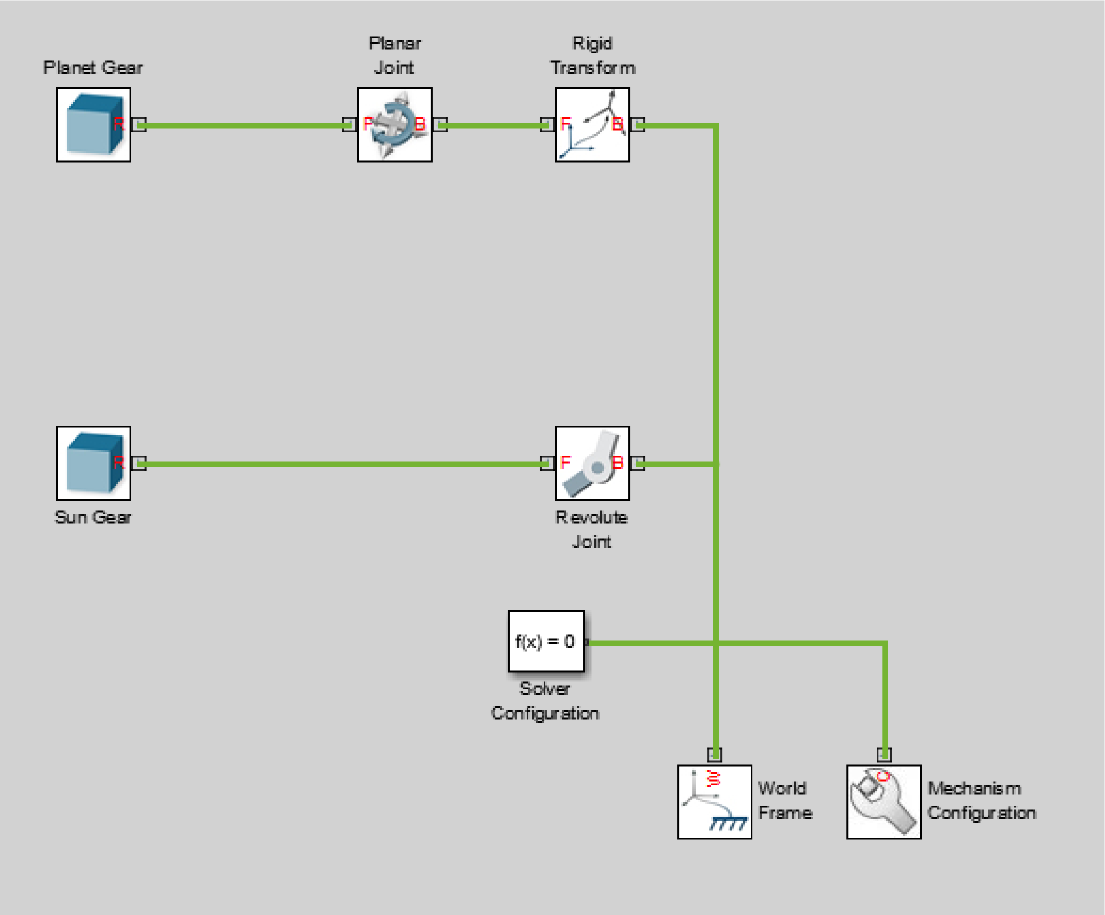
## Model Sun-Planet Gear Set

Model the gear rigid 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*Su`<br>Select units of `cm`. |
| **Geometry > Length** | Enter `T`. Select units of `cm`. |

| Parameter | Setting |
|---|---|
| **Inertia** > **Density** | Enter Rho. |
| **Graphic** > **Visual Properties** > **Color** | Enter Sun.RGB. |

The function simmechanics.demohelpers.gear_profile produces a rough approximation of an involute gear profile.

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*Pl<br>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.

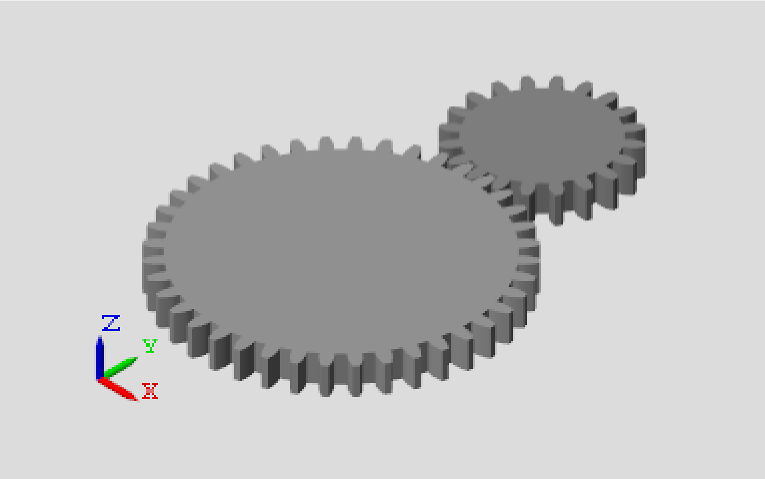
| Parameter | Setting |
|---|---|
| **Translation** > **Method** | Select Standard Axis. |
| **Translation** > **Axis** | Select +Y. |
| **Translation** > **Offset** | Enter Sun.R + Planet.R. Select units of cm. |

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`.

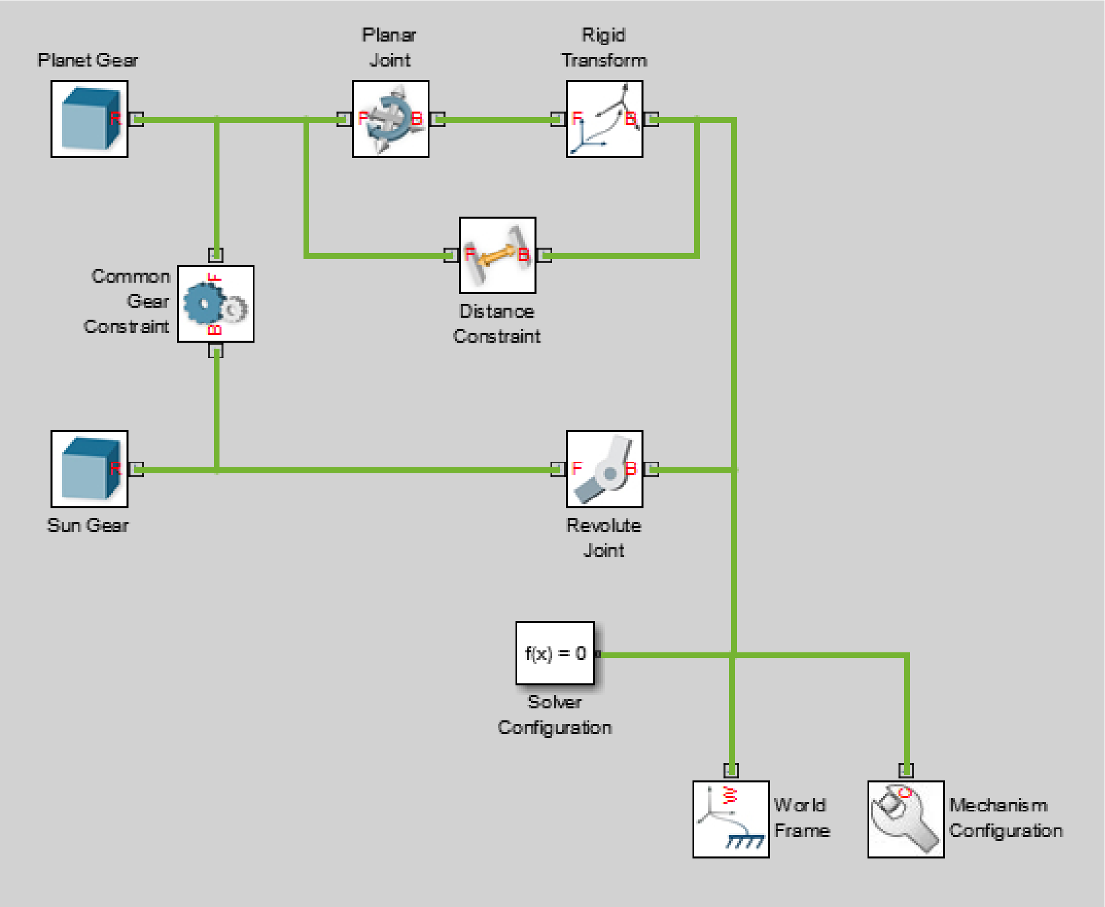
## Constrain Sun-Planet Gear Motion

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`.

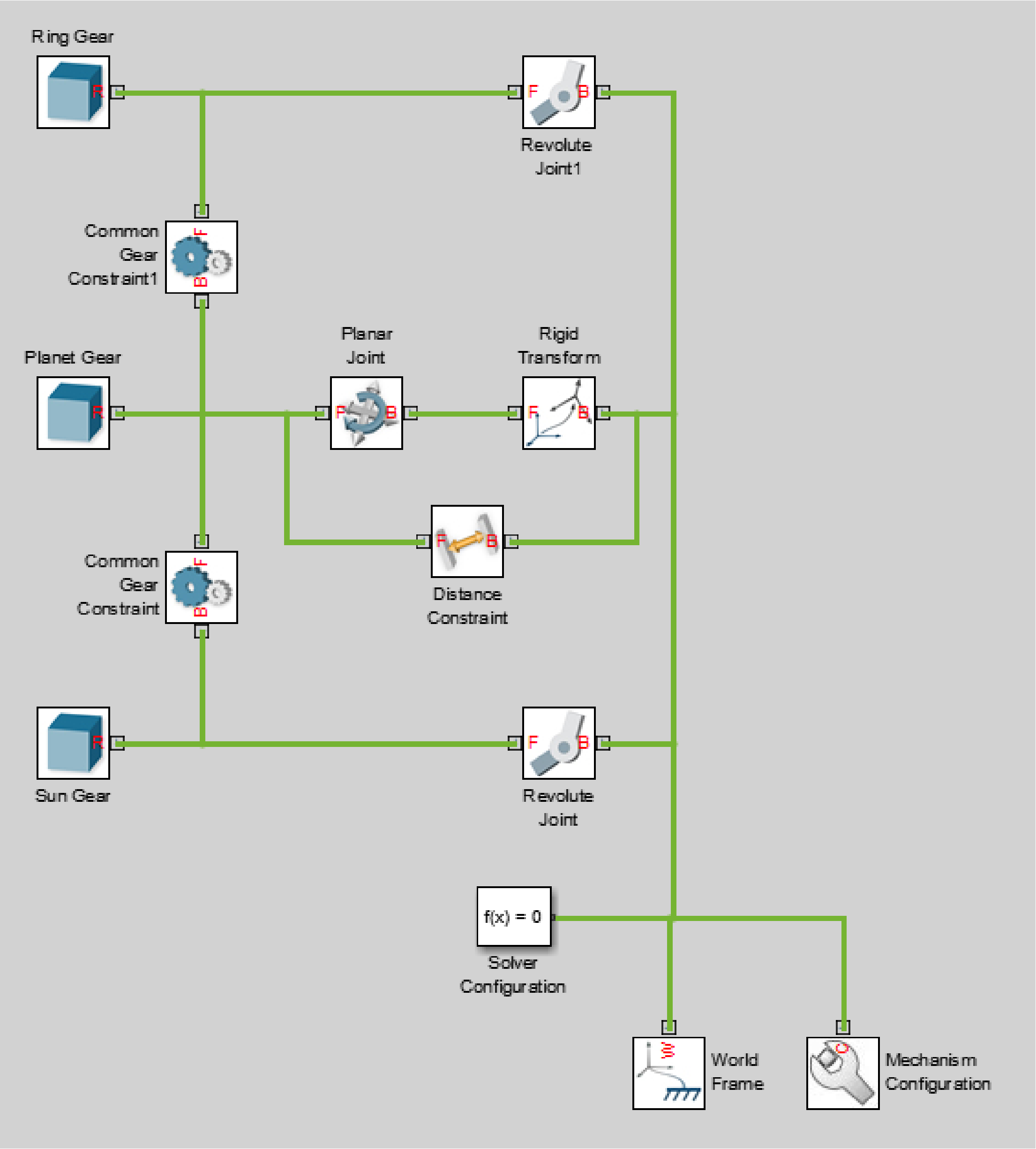
## Add Ring Gear

Model the ring gear rigid 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
R
F
B
Revolute Joint1
Common Gear Constraint1
F
B
Planet Gear
R
Planar Joint
F
B
Rigid Transform
F
B
F
B
Distance Constraint
Common Gear Constraint
F
B
Sun Gear
R
F
B
Revolute Joint
f(x) = 0
Solver Configuration
W
World Frame
C
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`. |

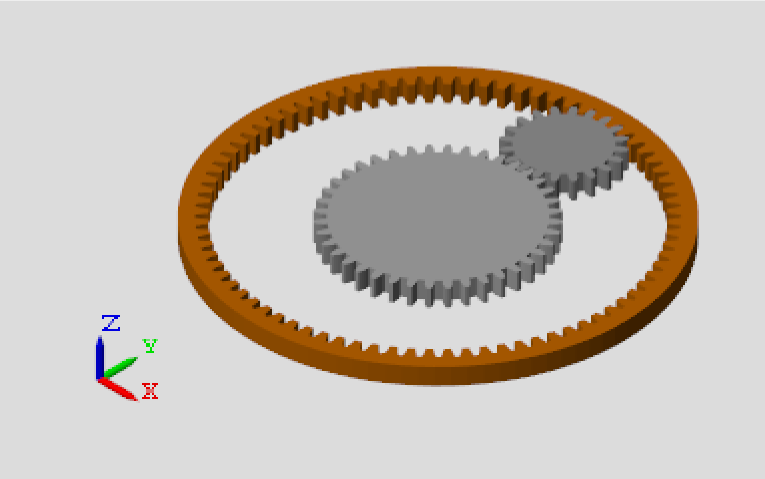
4 In the Common Gear Constraint1 block dialog box, specify these parameters.

| 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.



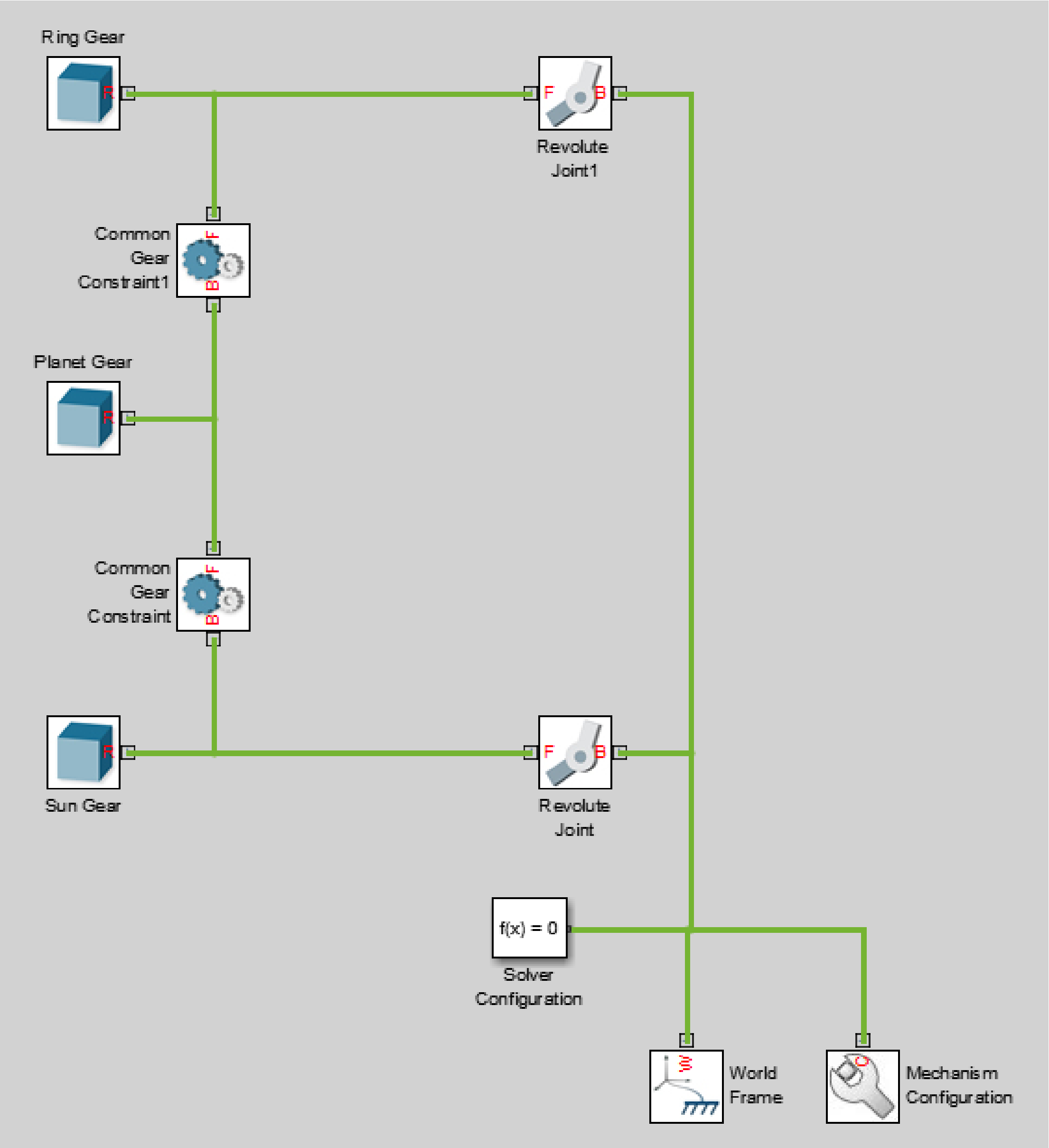
You can open a copy of the resulting model. At the MATLAB command line, enter `smdoc_planetary_gear_c`.

## 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

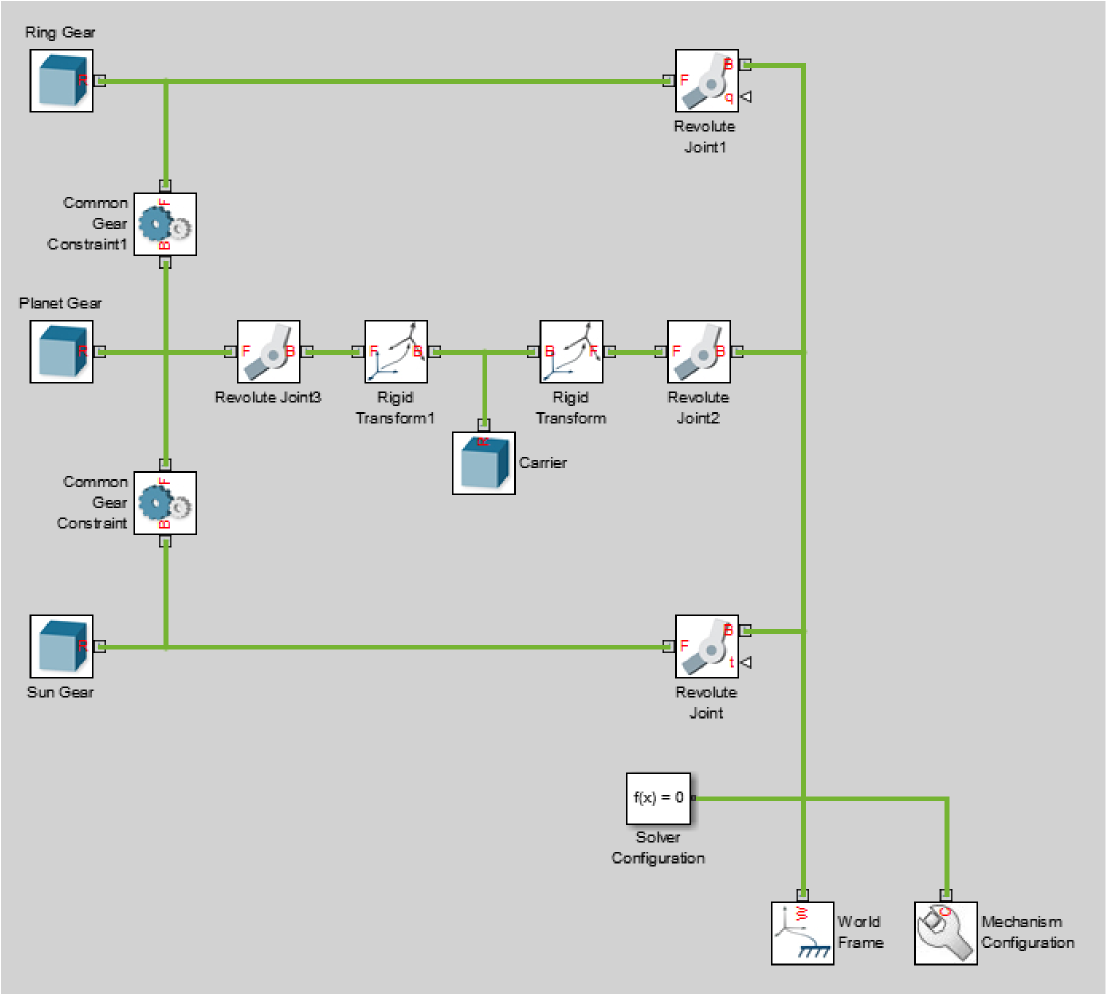

Ring Gear
R
F
B
Revolute Joint1
Common Gear Constraint1
F
B
Planet Gear
R
Common Gear Constraint
F
B
Sun Gear
R
F
B
Revolute Joint
f(x) = 0
Solver Configuration
W
World Frame
C
Mechanism Configuration

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`. |
| **Geometry > Cross-Section** | Enter `Carrier.CS`. Select units of `cm`. |
| **Geometry > Length** | Enter `Carrier.T`. |

| Parameter | Setting |
|---|---|
| **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. |

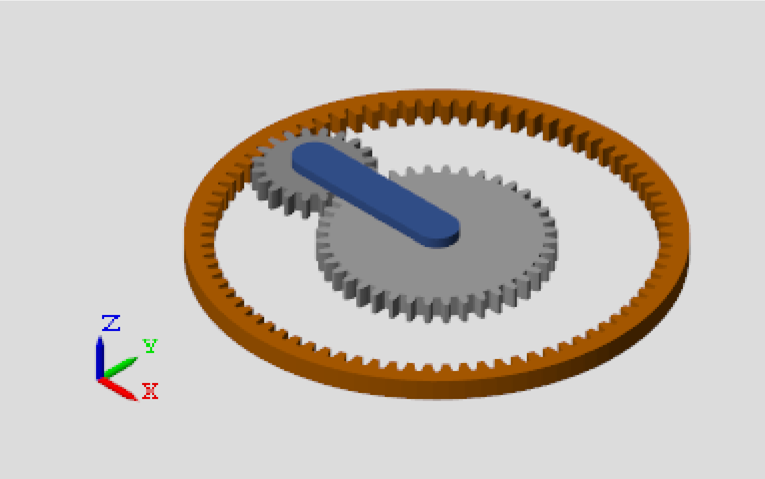
6 In the Rigid Transform1 block dialog box, specify these parameters.

| 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)];
```

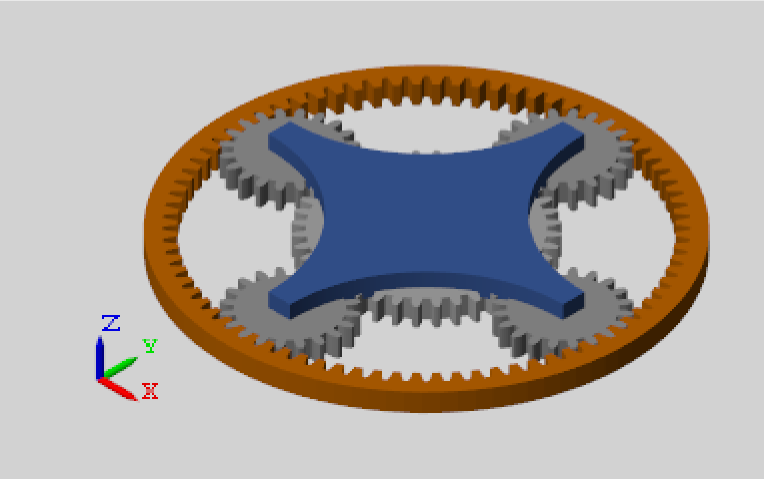
8 Simulate the model. To induce motion, try adjusting the velocity state targets in the joint block dialog boxes. Notice that the gear carrier now performs the task of the Distance Constraint block.



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 rigid 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`.

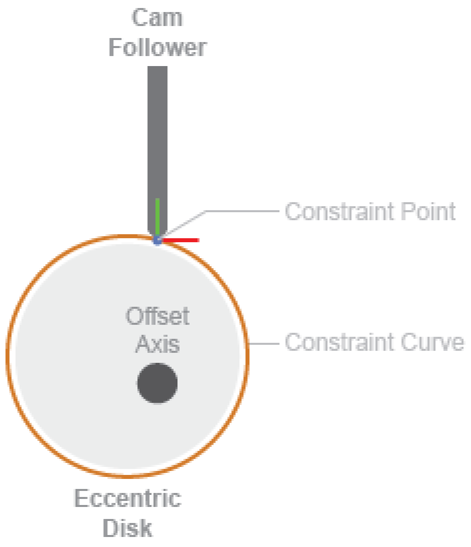

Z
Y
X

# Model Cam Mechanism

**In this section...**

## Model Overview

This tutorial shows how to model an eccentric cam mechanism. The mechanism consists of an eccentric disk (the cam) with a lever (the cam follower) mounted on its periphery. The distance between the rotation axis and perimeter of the cam varies with rotation angle, causing the follower to translate in a reciprocating motion.



**Cam Mechanism Schematic**

To work, the cam mechanism must constrain the follower tip (a point) to lie on the cam periphery (a curve). This type of constraint is known as point-on-curve. The same constraint is at work, for example, in a roller coaster cart bound to the perimeter of a track. You model this constraint using the `Point On Curve Constraint` block.

Any frame origin associated with a frame port can be a constraint point. Any curve associated with a geometry port can be a constraint curve. In this example, a frame origin positioned at the follower tip provides the constraint point. A circular spline curve defined in a `Spline` block provides the constraint curve.

## Geometry Ports

Geometry ports are analogues of frame ports. In the same way that frame ports identify frames on bodies, geometry ports identify curves and surfaces. You use these ports to apply kinematic constraints between the frames, curves, and surfaces that the ports represent.

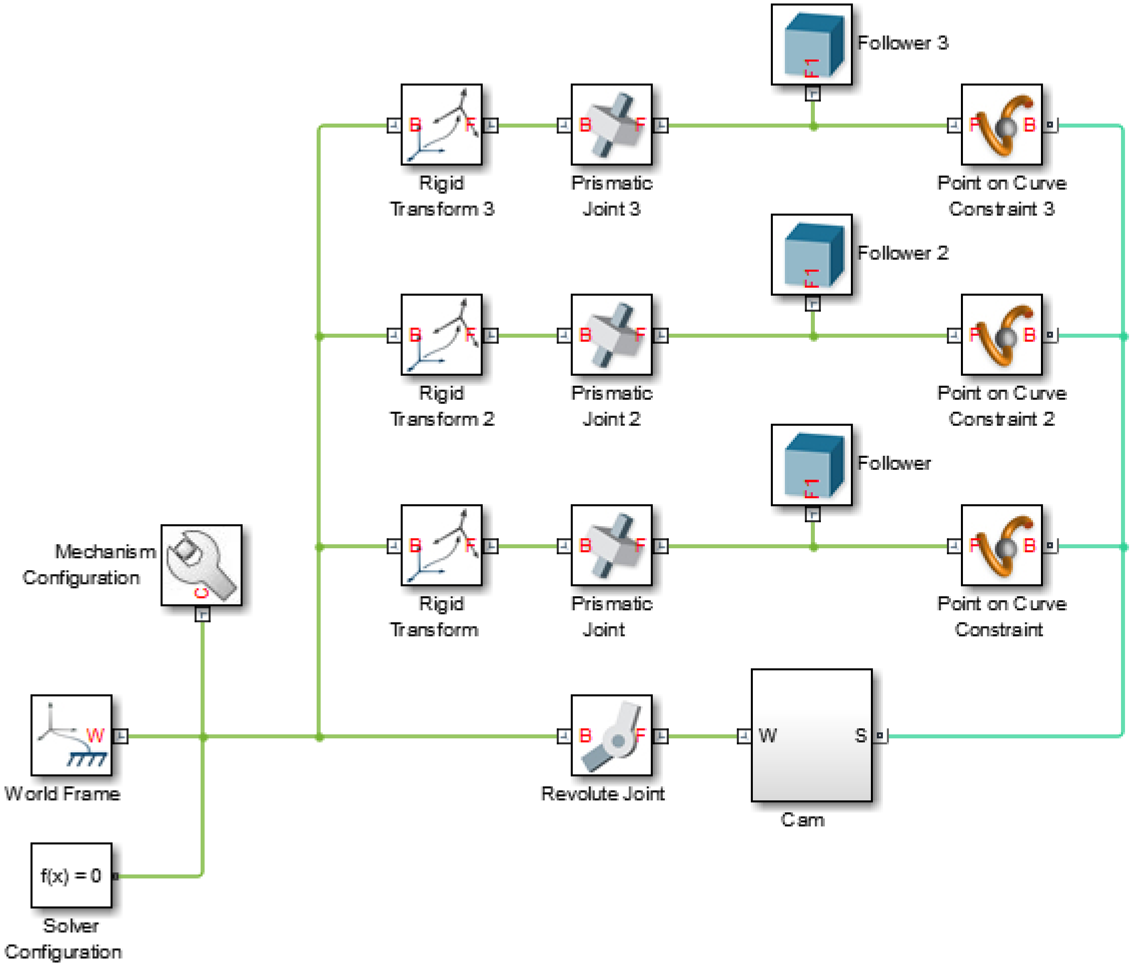
You can then apply kinematic constraints between the frames, curves, and surfaces given by these ports.

If a block has an intrinsic curve or surface definition, its geometry port makes that definition available to other blocks. If a block does not have such a definition, its geometry port enables you to reference one through a geometry connection line.

In this example, the Spline block provides an intrinsic curve definition. You specify this curve in the Spline block dialog box. The Point On Curve Constraint block, which does not have an intrinsic curve definition, then references this curve through a geometry connection line to the Spline block.

You can branch a geometry connection line, for example, so that it joins one Spline block to several Point On Curve Constraint blocks. Such a connection enables you to constrain various cam followers to the same cam or roller coaster carts to the same track.

However, branched or not, a geometry connection line must have exactly one geometry definition. If two blocks with intrinsic geometry definitions attach to the same geometry connection line, SimMechanics ignores one. If no such block connects to a geometry connection line, the model does not simulate. The figure shows a valid branched geometry line between one cam and three point-on-curve constraints.



## Spline Curves

The Spline block enables you to model a smooth, continuous curve. In mechanical systems, such curves are generally limited to contours on bodies. To preserve the parallel between your model and the system it represents, use the Spline block in the body subsystem that the spline curve is based on.

In this example, the spline curve represents the cam profile, a 2-D circle on the periphery of the cam. For this reason, you place the Spline block in the subsystem that represents the cam body. This approach treats the spline curve as part not of the constraint definition but of the body definition.

Treating the constraint curve as part of the cam body subsystem enables you to replace one cam subsystem for another without having to change anything else in the model. It also enables you to parameterize both the cam solid properties and spline curve in terms of MATLAB variables defined in a common subsystem mask.

To create the spline curve, the Spline block applies smooth interpolation between the data points you specify in the block dialog box. The interpolation ensures that the curve and its first two derivatives are continuous at each point. These constraints enable you to specify relatively complex curves with a only small number of interpolation points.

Because spline curves need only a small number of interpolation points, they are more computationally efficient than other curve types. However, if you specify a sufficiently large number of interpolation points, spline curves can slow down simulation. Try starting with a small number of interpolation points and gradually adding more until you reach the curve precision you need.

## Point On Curve Constraints

The Point On Curve Constraint block merely applies a constraint between a point and a curve. It defines neither the constraint point nor the constraint curve themselves. You must identify the constraint point externally by connecting the frame port of the Point On Curve Constraint block to another frame port in your model. The frame origin associated with the frame port or line is the constraint point.

Similarly, you must identify the constraint curve externally by connecting the geometry port of the Point On Curve Constraint block to another geometry port in your model. The geometry connection must be to a block with intrinsic curve definition, such as Spline. The curve associated with the geometry port is the constraint curve.

## Model Eccentric Cam

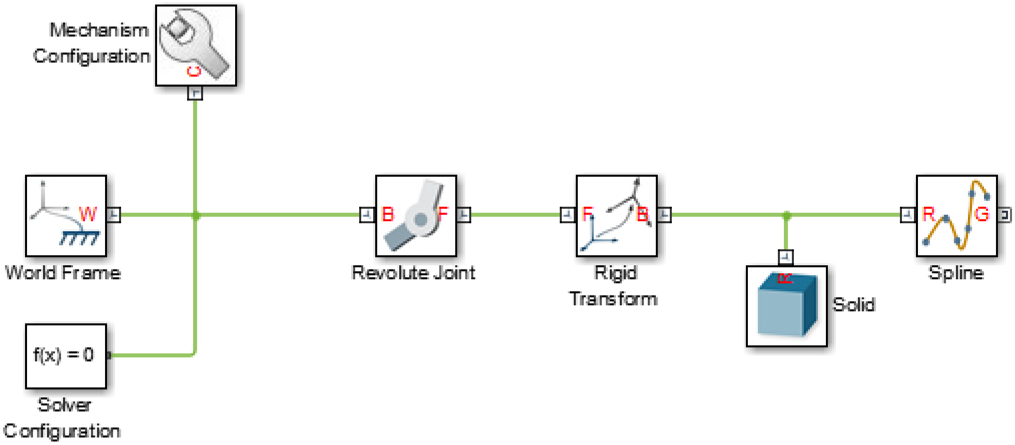
Add and connect blocks.

1 At the MATLAB command prompt, enter `smnew`. MATLAB opens the SimMechanics library and a model template with commonly used blocks. Remove all but the `Mechanism Configuration`, `World Frame`, and `Solver Configuration` blocks.

**2** Add these blocks to the model canvas.

| Library | Block | Purpose |
|---|---|---|
| Joints | `Revolute Joint` | Provide the cam one rotational degree of freedom relative to the World frame. |
| Frames and Transforms | `Rigid Transform` | Specify translational offset between cam center of mass and rotation axis. |
| Body Elements | `Solid` | Provide the solid properties of the cam body, including its geometry, inertia, and color. |
| Curves and Surfaces | `Spline` | Provide the cam profile for referencing in the Point On Curve Constraint block. |

**3** Connect the blocks as shown. Ensure that the base frame port of the Rigid Transform block faces the reference frame port of the Solid block.



The Spline block provides the curve coordinates relative to its reference frame port. The origin of this frame coincides with the `[0,0]` coordinate.

Specify the block parameters.

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

| Parameter | Value |
|---|---|
| **Translation > Method** | `Standard Axis` |
| **Translation > Axis** | `+Y` |
| **Translation > Offset** | `camOffset`, units of `in` |

The string `camOffset` is the translational offset between the cam center of mass and rotation axis specified as a MATLAB variable. You later initialize this and other MATLAB variables in the model workspace.

2 In the Solid block dialog box, specify these parameters.

| Parameter | Value |
|---|---|
| **Geometry > Shape** | `Cylinder` |
| **Geometry > Radius** | `r`, units of `in` |
| **Geometry > Length** | `t`, units of `in` |
| **Inertia > Density** | `rho` |
| **Graphic > Visual Properties > Color** | `rgbCam` |

The strings `r`, `t`, `rho`, `rgbCam` are the cam properties specified as MATLAB variables.

3 In the Spline curve block, specify these parameters.

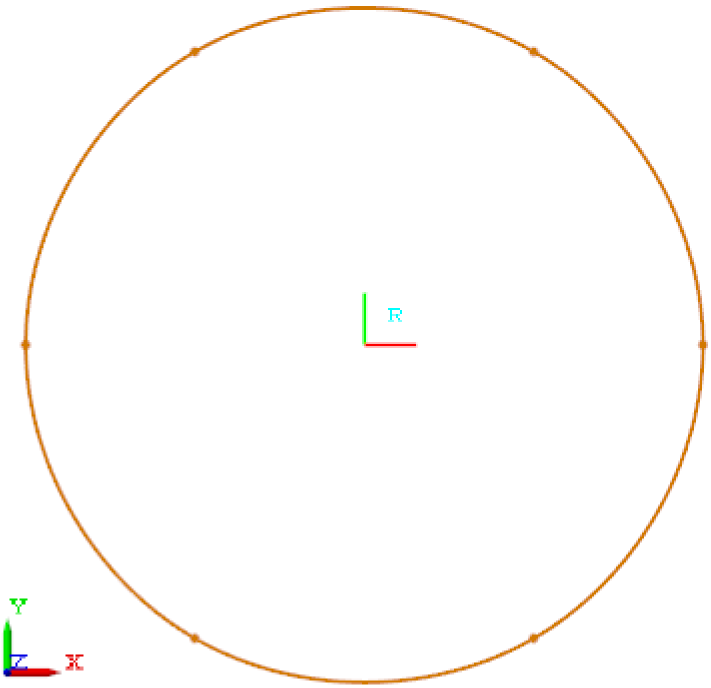
| Parameter | Value |
|---|---|
| **Interpolation Points** | `camProfile`, units of `in` |
| **Graphic > Visual Properties > Color** | `rgbSpline` |

The string `camProfile` is the spline curve specified as a MATLAB variable. The string `rgbSpline` is the color of the spline curve.

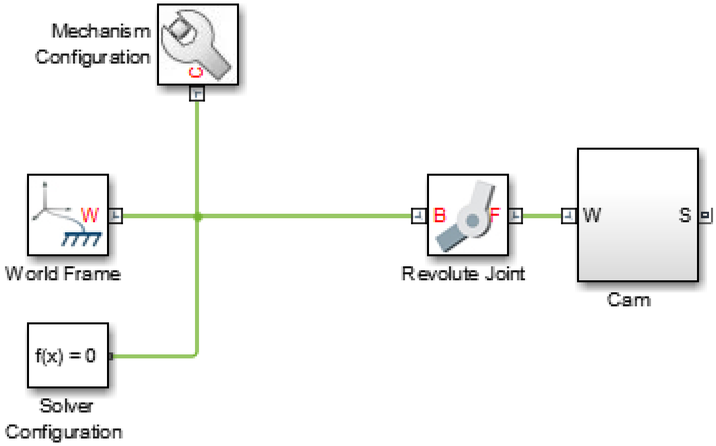
4 In the model workspace, define the MATLAB variables referenced in the block dialog boxes by entering this code:

```
% Cam parameters
r = 1; % Cam radius
t = 0.3; % Cam thickness
rho = 2700; % Aluminum density, kg/m^3
rgbCam = [1,1,1]; % Cam color
camOffset = r/3; % Distance from rotation axis to CM

% Spline parameters
rgbSpline = [210,120,0]/255; % Spline color
n = 6; % Number of interpolation points
theta = linspace(0,2*pi*(n-1)/n,n)'; % Angle vector

% Spline coordinates
x = r*cos(theta); % Interpolation-point x coordinates
y = r*sin(theta); % Interpolation-point y coordinates
camProfile = [x,y]; % Curve coordinate matrix
```

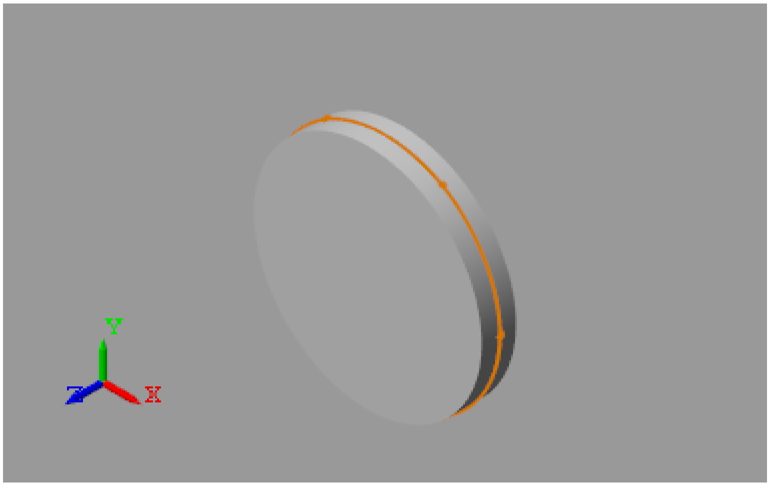
The spline portion of the code specifies the circular curve shown in the figure. You can view the spline curve in the visualization pane of the Spline block dialog box. The figure shows a top view of the curve.



Drag a selection box around the Rigid Transform, Solid, and Spline blocks. Then, select the Create Subsystem action button. Name the new Subsystem block Cam. This block represents the cam body.



To visualize the cam body, update the diagram. You can do this from the Simulink menu bar, by selecting **Simulation** > **Update diagram**. Mechanics Explorer opens with a visualization of the model. To obtain the view shown, in the Mechanics Explorer toolstrip, set the **View convention** parameter to `Y Up (XY Front)`. Then, select the **Isometric View** button.



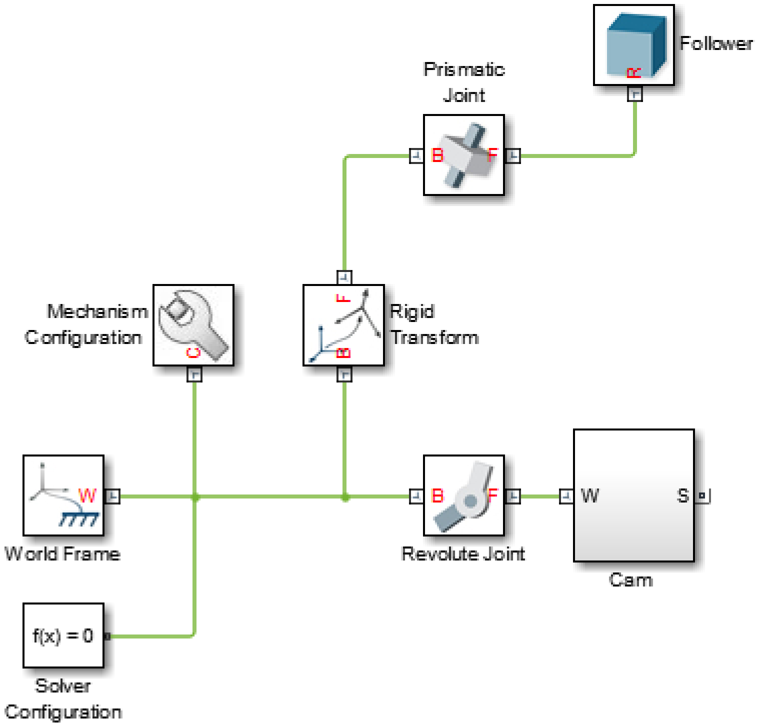
## Model Cam Follower

Add and connect blocks.

1 Add these blocks to the model.

| Library | Block | Purpose |
|---|---|---|
| Joints | `Prismatic Joint` | Provide the cam follower one translational degree of freedom relative to the World frame. |
| Frames and Transforms | `Rigid Transform` | Specify the relative orientation of the cam follower. |
| Body Elements | `Solid` | Provide the solid properties of the cam follower, including its geometry, inertia, and color. |

2 Connect the blocks as shown. Ensure the base frame port of the Rigid Transform block connects to the World frame line. Name the Solid block Follower.



Specify the block parameters.

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

| Parameter | Value |
|---|---|
| **Rotation > Method** | `Standard Axis` |
| **Rotation > Axis** | `+X` |
| **Rotation > Angle** | `-90` |

This rotation transform ensures that the translational axis of the cam follower is in the rotation plane of the cam.

2 In the Solid block dialog box, specify these parameters.

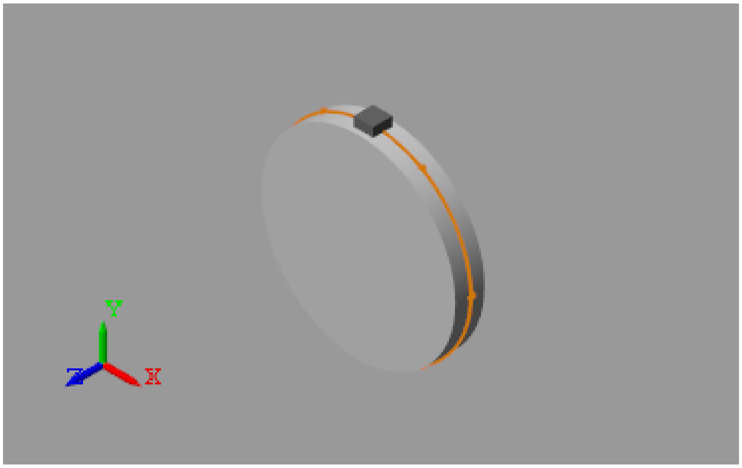
| Parameter | Value |
|---|---|
| **Geometry > Shape** | `Brick` |
| **Geometry > Dimensions** | `sizeFollower`, units of `in`. |
| **Inertia > Density** | `rho` |
| **Graphic > Visual Properties > Color** | `rgbFollower` |

The strings `sizeFollower`, `rho`, and `rgbFollower` are the cam follower properties specified as MATLAB variables.

3 In the model workspace, initialize the cam follower properties by adding this code:

```
% Follower parameters
sizeFollower = [0.2 0.2 1.5]; % Follower dimensions
rgbFollower = [0.5,0.5,0.5]; % Follower color
```

Update the diagram. Mechanics Explorer shows the updated model visualization. The cam and follower bodies overlap as the follower tip is not yet constrained to the cam periphery.



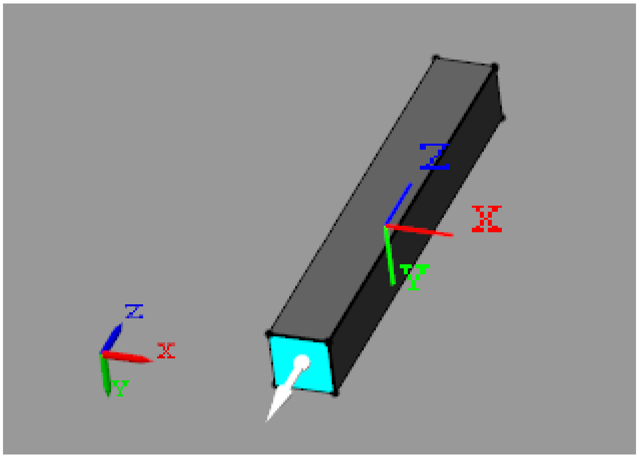
## Interactively Create Frame at Follower Tip

By default, the reference frame of a brick solid such as Follower is located at the center of mass. To apply the cam constraint to the bottom plane of the brick shape, you must create a new frame. You can create new frames interactively in the Solid block dialog box using the frame-creation interface.

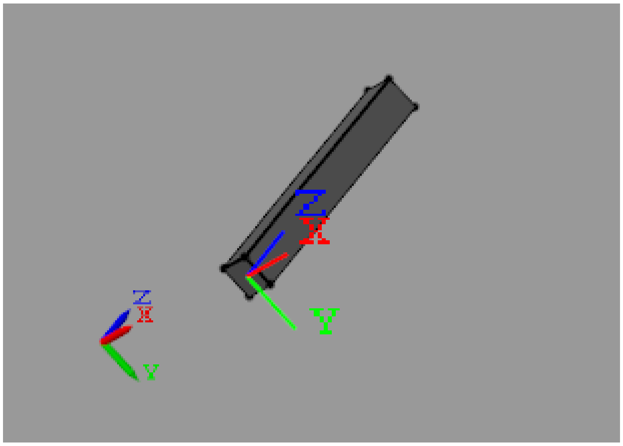
1 In the Solid block dialog box, expand the **Frames** area and select the **Create** button. If you make any changes to the block parameters, you must first select the Update Visualization button ⇄.

2 In the frame-creation interface, under **Frame Origin**, select **Based on Geometric Feature**. This option enables you to place the frame origin at the center of the selected geometric feature, be it a plane, a line, or a point. If you select a point, the frame origin coincides with that point.

3 In the visualization pane, rotate the solid and select its bottom plane. This plane is normal to the -z axis. The visualization pane highlights the selected plane. An arrow shows the center and normal vector of this plane.



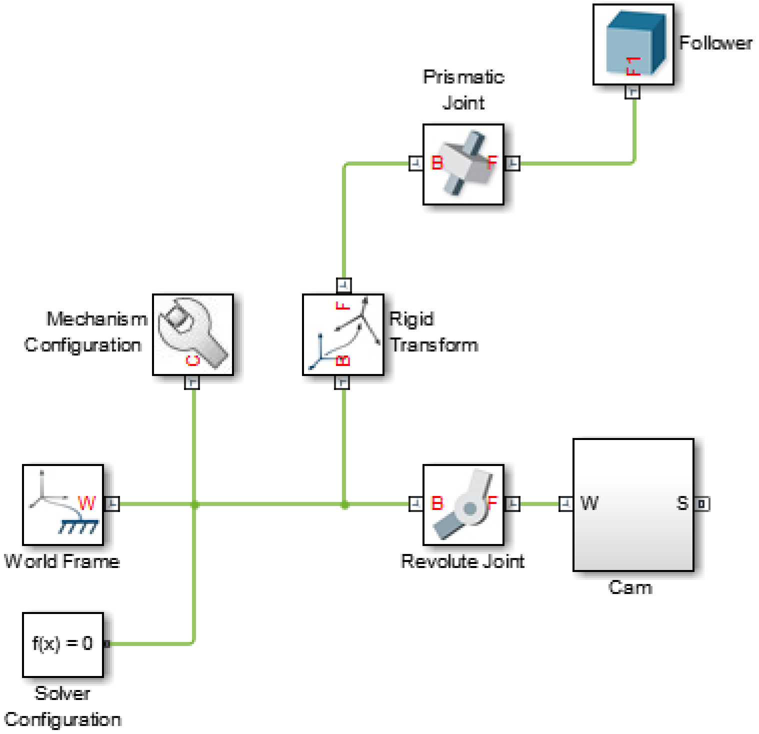
4 Under **Frame Origin**, select the **Use Selected Feature** button. This button sets the center of the selected surface as the origin of the new frame. By default, the frame orientation is that of the solid reference frame.



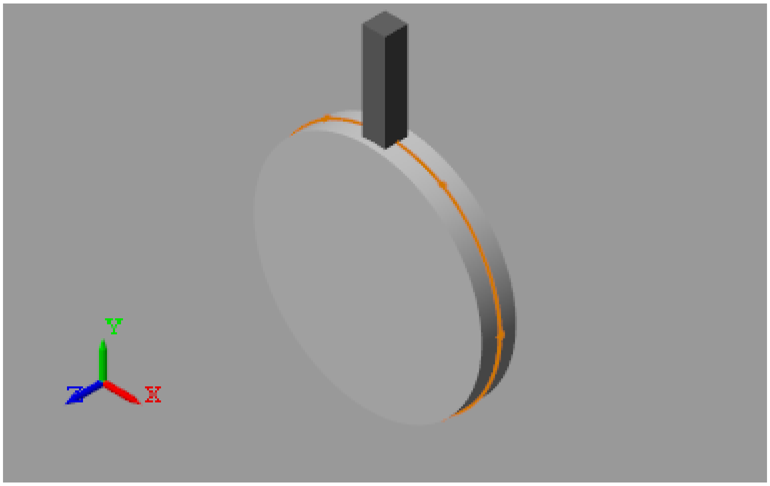
5 Select **Save**. The block saves the new frame definition without committing it to the model. If you close the block dialog box without first selecting **Apply** or **OK**, the block discards the new frame definition.

6 In the **Frames** area, clear the **Show Port R** check box. The block hides the reference frame port. You do not need this port in this model. Select **OK** to commit your changes to the model.

7 Rotate the Follower block and connect its new frame port, labeled F1, to the model as shown.



Update the diagram. Mechanics Explorer shows the updated model visualization. The model assembles with the cam follower in a new position, though the cam and follower bodies are still unconstrained.

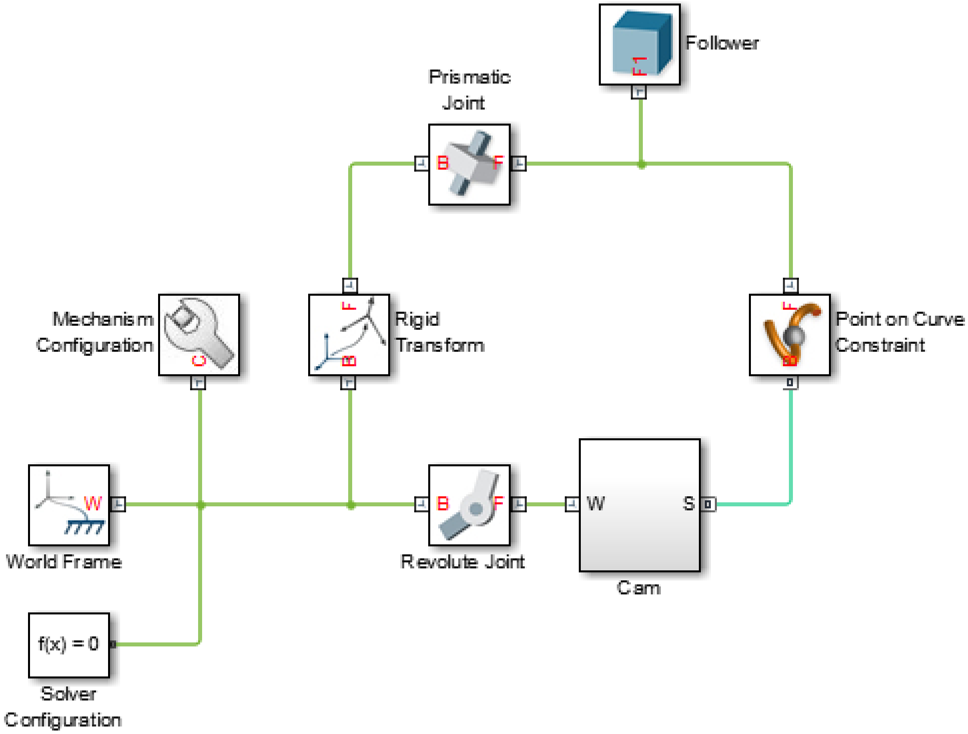


## Constrain Cam and Follower

The model now has the bodies it needs, cam and follower, each with the correct degrees of freedom. To complete the model, you need only constrain the two bodies.

1 From the Constraints library, drag a Point On Curve Constraint block.

2 Connect the block as shown. Ensure that the base geometry port connects to the geometry port of the Cam subsystem block. The base geometry port identifies the constraint curve.



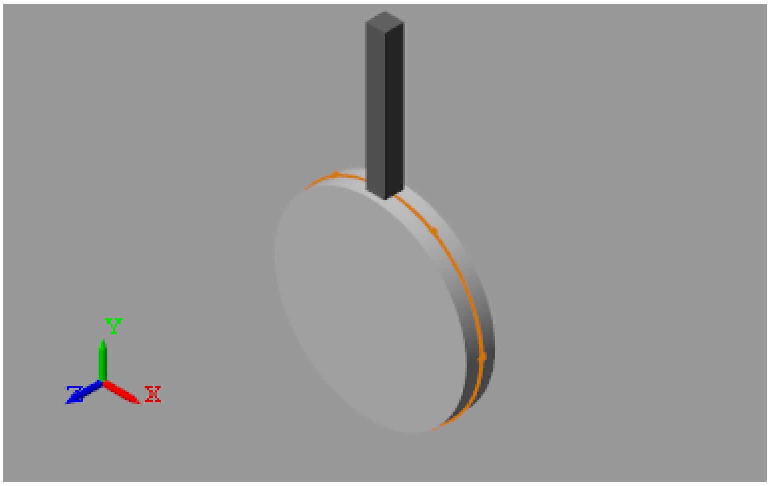
Set the cam in motion by specifying a nonzero initial velocity. Set gravity to zero to ensure uniform motion of the cam.

1 In the Revolute Joint block dialog box, specify these parameters.

| Parameter | Setting |
|---|---|
| **State Targets > Specify Velocity Target** | Select check box. |
| **State Targets > Specify Velocity Target > Value** | Enter `0.5`. Select units of `rev/s`. |

2 In the Mechanism Configuration block, set the **Uniform Gravity** parameter to `None`.

Simulate the model. You can do this from the Simulink menu bar by selecting **Simulation** > **Run**. Mechanics Explorer shows a physics-based animation of the cam mechanism.



# Internal Mechanics, Actuation and Sensing

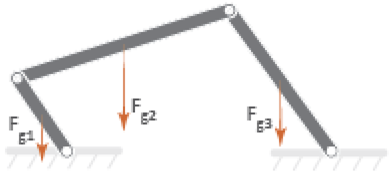
# Modeling Gravity

**In this section...**

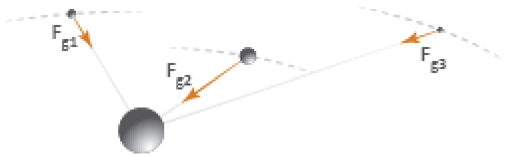
## Gravity Models

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 SimMechanics, you can add gravity to systems like these using three gravity models:

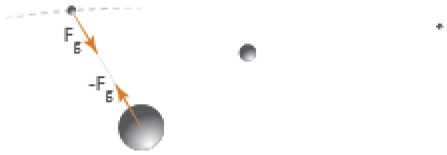
- Uniform gravity, as experienced by most earthbound systems. The force on each body due to uniform gravity depends only on its mass. This force is the same everywhere in space for a given body, though it can vary in time. You model uniform gravity using the `Mechanism Configuration` block.



- 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

$$F_g = -G\frac{Mm}{R^2}$$

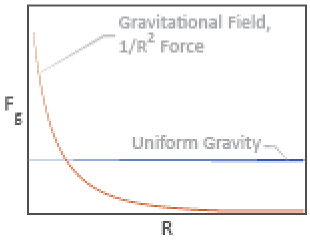
with $G$ being the gravitational constant. This is the force that you model when you represent gravity through Gravitational Field or Inverse Square Law Force blocks. If the distance between source and target masses is constant, the gravitational force reduces to a simpler form,

$$F_g = -mg$$

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 “SimMechanics Bodies” on page 2-2 for more information on how SimMechanics 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.

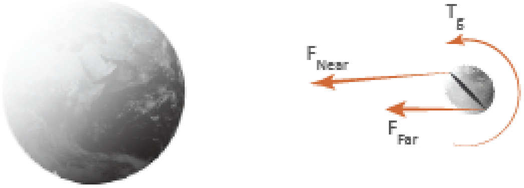
If you want to model the effects of gravity on a point other than a body’s center of mass, you can add a frame at the desired location and apply a gravitational force directly at that frame. You model the force using the Inverse Square Law Force block. This force points along the imaginary line between 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.

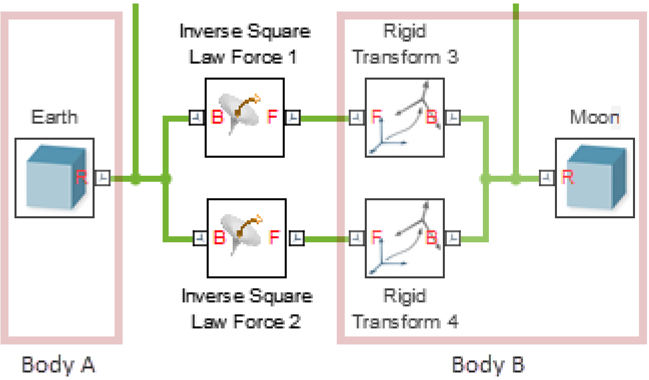
| Block | Position | Direction |
|---|---|---|
| Mechanism Configuration | Center of Mass | Specified gravity vector |
| Gravitational Field | Center of Mass | Field origin-center of mass line |
| Inverse Square Law Force | Connection frames | Base-follower frame line |

## 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.

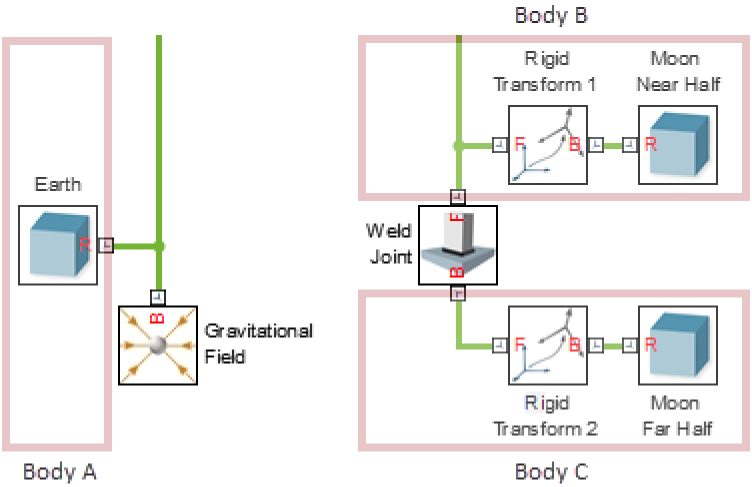


You can model such torques in SimMechanics by modeling the different gravitational forces acting on a body. You do this using the Inverse Square Law Force or Gravitational Field block. If you use the Inverse Square Law Force block, you must create additional frames in each body whose response to gravitational 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.



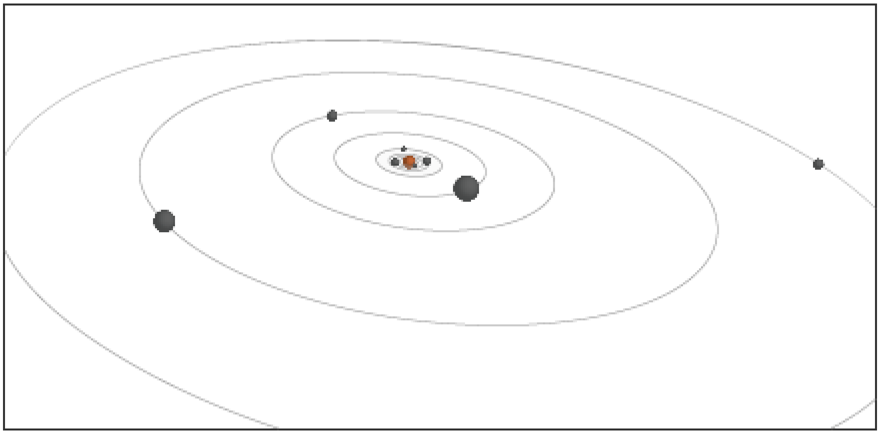
**Torque on the moon due to dissimilar gravitational forces at the elongated ends**

# Model Planetary Orbit Due to Gravity

**In this section...**

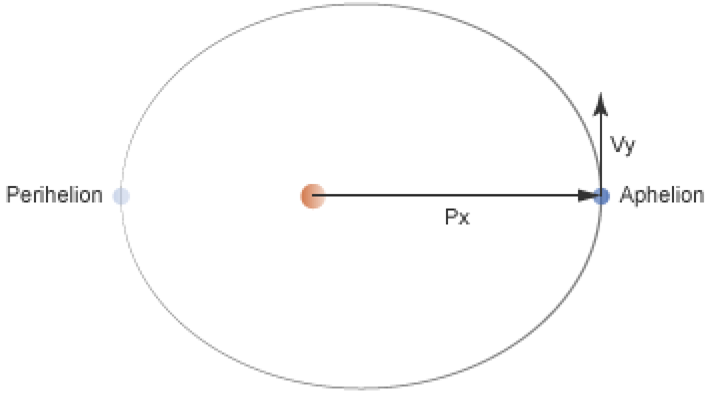
## Model Overview

In this tutorial, you model planet orbit in our solar system due to gravity. The planets are treated as spheres, each with three translational and zero rotational degrees of freedom with respect to a fixed world frame. Cartesian Joint blocks provide these degrees of freedom. Gravitational Field blocks generate the attractive forces that keep the planets in orbit.



For simplicity, the tutorial assumes that each planet is at aphelion at simulation time zero. Aphelion is the point of greatest distance from the sun. It corresponds to the minimum orbital velocity of a given planet.

Placing the planets simultaneously at aphelion while neglecting orbital parameters such as inclination and longitude of ascension node causes them to align on a single axis. In the tutorial, aphelion lies on the X axis at a distance Px from the sun. The planet velocity at aphelion is orthogonal to the line joining the planet to the sun. It points along the Y axis with magnitude Vy.

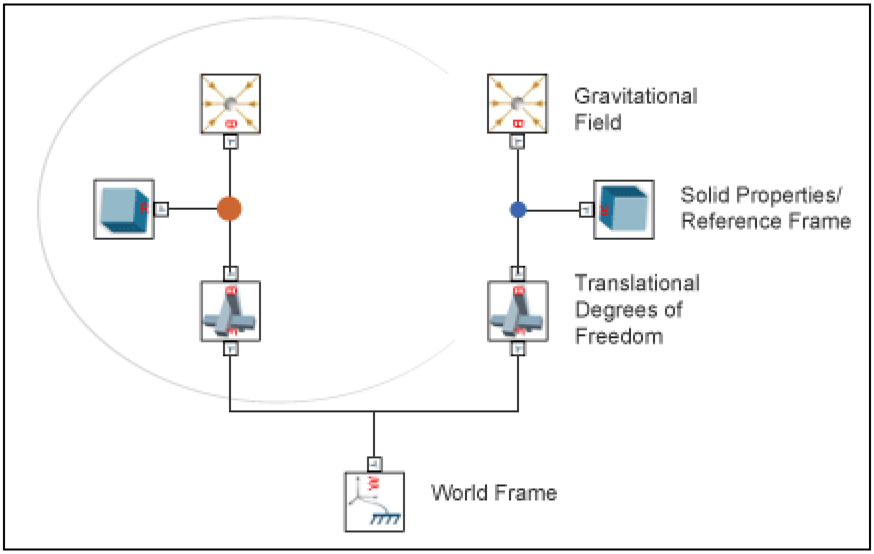


The procedure in this tutorial shows how you can model the gravitational interaction between Earth and the sun. In this procedure, you add two Cartesian Joint blocks. One block, connected between the Earth reference frame and the World frame, provides three translational degrees of freedom to Earth. The other block, connected between the Sun reference frame and the World frame, provides the same to the sun, allowing it too to move under the gravitational pull of Earth. Planet spin is ignored.

Similarly, you add two Gravitational Field blocks to the model. One block, connected to the sun reference frame, represents the gravitational field of the sun. This field exerts an attractive pull on Earth, but it has no effect on the sun itself. To represent the attractive pull of Earth on the sun, you connect the second Gravitational Field block to the Earth reference frame.

When using the Gravitational Field block, you must set the **Uniform Gravity** parameter of the Mechanism Configuration block to `None`. This step prevents conflicting gravity sources in your model. If you forget to take this step, SimMechanics throws an error and the model does not simulate.

The figure shows the key blocks in this model and their connections. Note that you must add other blocks, such as Solver Configuration, for the model to simulate.


Gravitational Field
Solid Properties/ Reference Frame
Translational Degrees of Freedom
World Frame

## Build Model

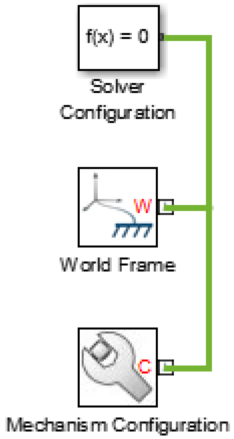
Start by modeling the gravitational interaction between the sun and the earth. This model shows the approach to use when you add the remaining planets to the model. Except where noted, the blocks you use are from the SimMechanics Second Generation library.

### Start Model

1. Start a new model.
2. Add these blocks to the model.

| Library | Block |
|---|---|
| Utilities | `Mechanism Configuration` |
| Frames and Transforms | `World Frame` |
| Simscape Utilities | `Solver Configuration` |

3. Connect the blocks as shown.



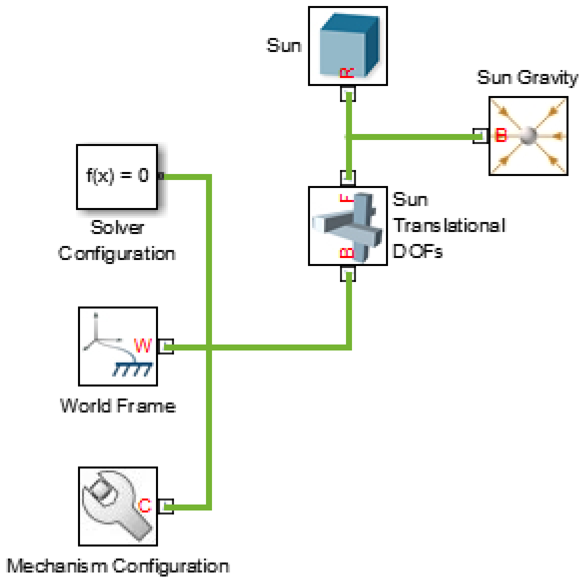
4 In the Mechanism Configuration block dialog box, set **Uniform Gravity** to `None`. This change enables you to specify gravity using a Gravitational Field block instead.

### Add Sun

1 Add these blocks to the model. The blocks represent the sun: its solid properties, its gravitational pull on other bodies, and its degrees of freedom with respect to the (fixed) World frame.

| Library | Block |
|---|---|
| Body Elements | `Solid` |
| Forces and Torques | `Gravitational Field` |
| Joints | `Cartesian Joint` |

2 Connect the blocks as shown. Note that each line joining two or more frame ports represents a single frame. The R, B, and F port frames of the Sun, Sun Gravity, and Sun Translational DOFs blocks are coincident with each other.



**3** In the Sun block dialog box, specify these parameters. These variables are the solid properties of the sun. You define their numerical values later in the model workspace.

| Parameter | Select or Enter |
|---|---|
| **Geometry** > **Shape** | `Sphere` |
| **Geometry** > **Radius** | `Sun.R` |
| **Inertia** > **Based on** | `Mass` |
| **Inertia** > **Mass** | `Sun.M` |
| **Graphic** > **Visual Properties** > **Color** | `Sun.RGB` |

**4** In the Sun Gravity block dialog box, for **Mass**, enter `Sun.M`. The block uses this mass to calculate the gravitational force on the planets as a function of distance.

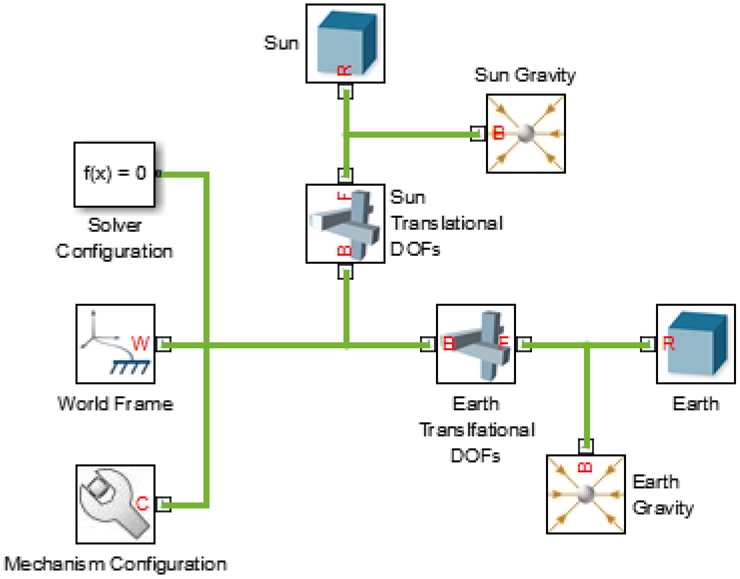
In the previous two steps, you specified the sun mass in two blocks, Sun and Sun Gravity, using a single variable, `Sun.M`. This approach ensures the two blocks, both of which represent the sun, use the same mass.

## Add Earth

1 Add these blocks to the model. The blocks represent Earth: its solid properties, its gravitational pull on other bodies, and its degrees of freedom with respect to the (fixed) World frame.

| Library | Block |
|---|---|
| Body Elements | `Solid` |
| Forces and Torques | `Gravitational Field` |
| Joints | `Cartesian Joint` |

2 Connect the blocks as shown. The R, B, and F port frames of the Earth, Earth Gravity, and Earth Translational DOFs blocks are coincident with each other.



3 In the Earth block dialog box, specify these parameters. These variables are the solid properties of Earth. You define their numerical values later in the model workspace.

| Parameter | Select or Enter |
|---|---|
| **Geometry** > **Shape** | `Sphere` |
| **Geometry** > **Radius** | `Earth.R` |
| **Inertia** > **Based on** | `Mass` |
| **Inertia** > **Mass** | `Earth.M` |
| **Graphic** > **Visual Properties** > **Color** | `Earth.RGB` |

4 In the Earth Gravity block dialog box, for **Mass**, enter `Earth.M`. This variable is the earth mass you specified in the Earth block. The block uses this mass to calculate the gravitational force as a function of distance.

5 In the Earth Translational DOFs block dialog box, specify these parameters. The state target variables set the initial position and velocity of Earth with respect to the initial position of the sun (originally coincident with the fixed World frame).

| Parameter | Select or Enter |
|---|---|
| **X Prismatic Primitive (Px)** > **State Targets** | a Select the **Specify Position** check box.<br>b In **Value**, enter `Earth.Px`.<br>c Select the **Specify Velocity** check box.<br>d In **Value**, enter `Earth.Vx`. |
| **Y Prismatic Primitive (Py)** > **Specify Position Target** | a Select the **Specify Position** check box.<br>b In **Value**, enter `Earth.Py`.<br>c Select the **Specify Velocity** check box.<br>d In **Value**, enter `Earth.Vy`. |

### Define Model Variables

1 In the Simulink editor, select **Tools** > **Model Explorer**.

2 In the Model Hierarchy pane of Model Explorer, expand the node for your model and select **Model Workspace**.

3 In the Model Workspace pane, in **Data Source**, select `MATLAB Code`.

4. In the **MATLAB Code** field, enter:

   ```
   %% Sun Parameters
   % Scale sun size for visualization purposes
   SunScaling = 1e2;

   % Specify the solid properties of the sun
   Sun.M = 1.99e30;
   Sun.R = 6.96e8*SunScaling;
   Sun.RGB = [1, 0.6, 0];

   %% Earth Parameters
   % Scale Earth size for visualization purposes
   TerrestrialPlanetScaling = 1.2e3; % Scale the size of Earth

   % Specify the solid properties of Earth
   Earth.M = 5.97e24;
   Earth.R = 6.05e6*TerrestrialPlanetScaling;
   Earth.RGB = [0.5, 0.8, 1];

   % Specify initial position and velocity of Earth
   Earth.Px = 1.52e11;
   Earth.Py = 0;
   Earth.Vx = 0;
   Earth.Vy = 2.93e4;
   ```

   The scaling factors impact visualization only. They have no impact on model dynamics. These factors become more important as you add planets to the solar system model. As the distances between the sun and the outer planets increase, the apparent sizes of these bodies decrease, sometimes to just a few pixels. The scaling factors artificially increase the apparent sizes of the sun and the planets so that they remain visible in Mechanics Explorer.

5. Click **Reinitialize from Source**.

## Configure Simulation Parameters

Large gravitational systems such as the Earth-sun pair typically have long periods of revolution. To capture multiple Earth revolutions during simulation, increase the simulation stop time to several years. To ensure an adequate balance between simulation speed and accuracy, adjust the solver step sizes.

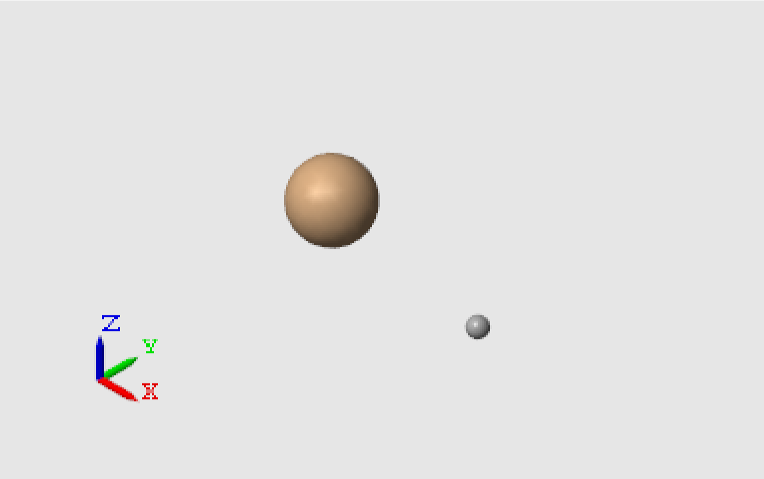
1. In the Simulink editor, select **Simulation** > **Model Configuration Parameters**.

2 In **Stop time**, enter `10*365*24*60*60`. This value equals 10 years in seconds. It allows simulation to run for 10 consecutive Earth revolutions about the sun.

3 In **Max step size**, enter `24*60*60`. This value equals one day in seconds.

## Simulate Model

Update the block diagram, e.g., by pressing **Ctrl+D** with the model window active. Mechanics Explorer opens with a static 3-D display of the model in its initial state. Adjust the viewpoint and zoom level to optimize visualization on your screen.



Run the simulation, e.g., by pressing **Ctrl+T**. Mechanics Explorer plays a physics-based animation of the solar system. At the default value of the base playback speed, planet motion is hardly noticeable. Increase this value to watch Earth complete one revolution about the sun in 10 seconds:

1 In Mechanics Explorer, select **Tools** > **Animation Settings**.

2 In **Base(1X) Playback Speed**, enter `3153600`. This number equals one 10th of a year in seconds.

3 Click **Pause** and then **Play**. Mechanics Explorer applies the previous changes. You can now watch Earth revolve about the sun once every ten seconds.

## Add Remaining Planets

Complete the solar system model by adding the remaining planets. Follow the approach that you used to model the Earth-sun system.

The table shows the relevant parameters for each planet. To better visualize the planets, consider scaling the radius parameters. For example, you can multiply the terrestrial planet radii by a common scaling factor (e.g., `TerrestrialPlanetScaling`), and the gas giant planets by another common scaling factor (e.g., `GasGiantScaling`).

| Planet | Radius (m) | Mass (kg) | X Position Target (m) | X Velocity Target (m/s) | Y Position Target (m) | Y Velocity Target (m/s) |
|---|---|---|---|---|---|---|
| Mercury | 2.44e6 | 3.30e23 | 6.98e10 | 0 | 0 | 3.89e4 |
| Venus | 6.05e6 | 4.87e24 | 1.09e11 | 0 | 0 | 3.48e4 |
| Mars | 3.39e6 | 6.42e23 | 2.49e11 | 0 | 0 | 2.2e4 |
| Jupiter | 6.99e7 | 1.90e27 | 8.17e11 | 0 | 0 | 1.24e4 |
| Saturn | 5.82e7 | 5.68e26 | 1.51e12 | 0 | 0 | 9.09e3 |
| Uranus | 2.54e7 | 8.68e25 | 3.00e12 | 0 | 0 | 6.49e3 |
| Neptune | 2.46e7 | 1.02e26 | 4.55e12 | 0 | 0 | 5.37e3 |

If you struggle with the remaining planets, you can open a completed model of the solar system. At the MATLAB command prompt, enter:

- `smdoc_solar_system_wfield_a` — Model of the inner solar system. This model includes the sun and terrestrial planets only.
- `smdoc_solar_system_wfield_b` — Model of the complete solar system. This model adds to the previous model the gas giant planets.

These models include Rigid Transform blocks to account for orbit inclination and longitude of ascension node.

Note that gravity is an inverse-square law force. That is, the force magnitude decays with the square distance between a body and the field source mass. As such, you can model the gravitational force between two bodies using the Inverse Square Law Force block. However, this block applies a force only between its two port frames. Use this approach to neglect secondary gravitational interactions, e.g., if orbital perturbations due to planet-planet interactions are irrelevant to your application.

Open these models to see how to model the solar system using the Inverse Square Law Force block instead:

- `smdoc_solar_system_wforce_a` — Model of the solar system with terrestrial planets.
- `smdoc_solar_system_wforce_b` — Model of the solar system with terrestrial and gas-giant planets.

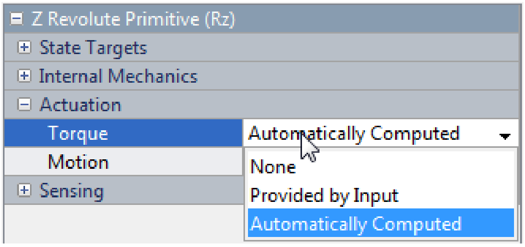
These models account only for the gravitational interactions between sun-planet pairs.

# Joint Actuation

## Actuation Modes

Joint blocks provide two actuation parameters. These parameters, **Force/Torque** and **Motion**, govern how the joint behaves during simulation. Depending on the parameter settings you select, a joint block can accept either actuation parameter as input or automatically compute its value during simulation.

An additional setting (`None`) allows you to set actuation force/torque directly to zero. The joint primitive is free to move during simulation, but it has no actuator input. Motion is due indirectly to forces and torques acting elsewhere in the model, or directly to velocity state targets.



Z Revolute Primitive (Rz)

State Targets

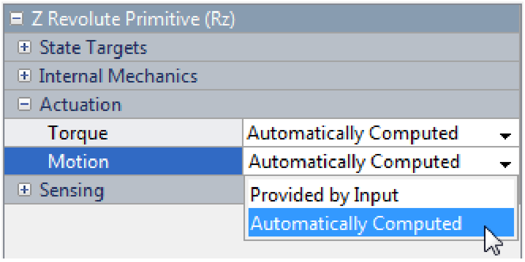
Internal Mechanics

Actuation

Torque: Automatically Computed

Motion

Sensing



Z Revolute Primitive (Rz)

State Targets

Internal Mechanics

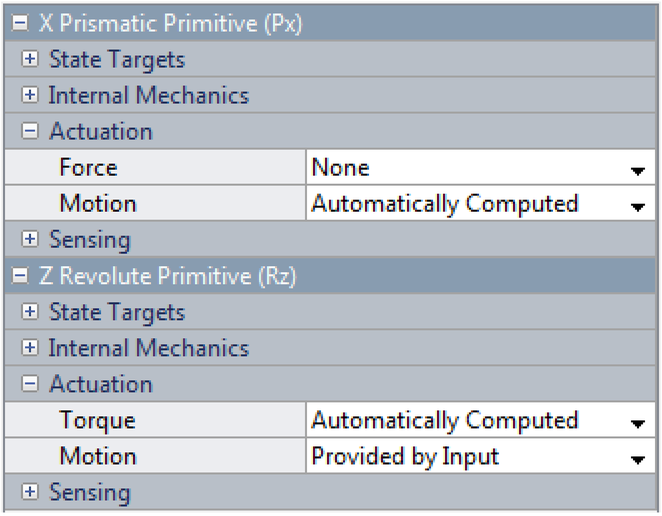
Actuation

Torque: Automatically Computed

Motion: Automatically Computed

Sensing

Like all joint block parameters, you select the actuation parameter settings for each joint primitive separately. Different joint primitives in the same block need not share the same actuation settings. Using a `Pin Slot Joint` block, for example, you can provide motion input and have actuation torque automatically computed for the **Z Revolute Primitive (Rz)**, while having motion automatically computed with no actuation force for the **X Prismatic Primitive (Px)**.



⊟ X Prismatic Primitive (Px)

⊞ State Targets

⊞ Internal Mechanics

⊟ Actuation

| | |
|---|---|
| Force | None |
| Motion | Automatically Computed |

⊞ Sensing

⊟ Z Revolute Primitive (Rz)

⊞ State Targets

⊞ Internal Mechanics

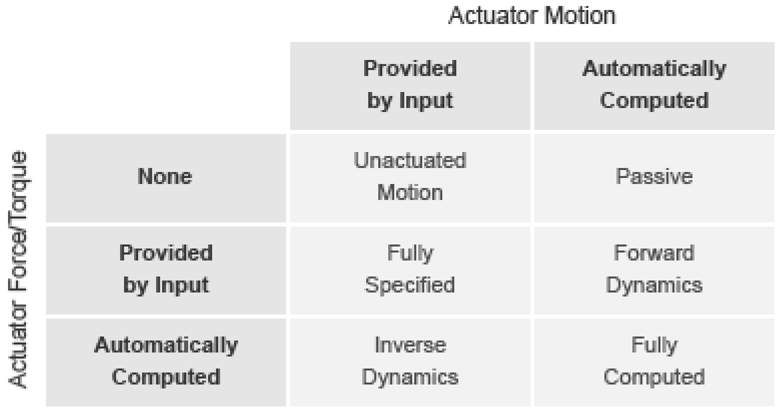
⊟ Actuation

| | |
|---|---|
| Torque | Automatically Computed |
| Motion | Provided by Input |

⊞ Sensing

By combining different **Force/Torque** and **Motion** actuation settings, you can achieve different joint actuation modes. Forward dynamics and inverse dynamics modes are two common examples. You actuate a joint primitive in forward dynamics mode by providing actuation force/torque as input while having motion automatically computed. Conversely, you actuate a joint primitive in inverse dynamics mode by providing motion as input while having actuation force/torque automatically computed.

Other joint actuation modes, including fully computed and fully specified modes, are possible. The table summarizes the different actuation modes that you can obtain by manipulating the actuation parameter settings.



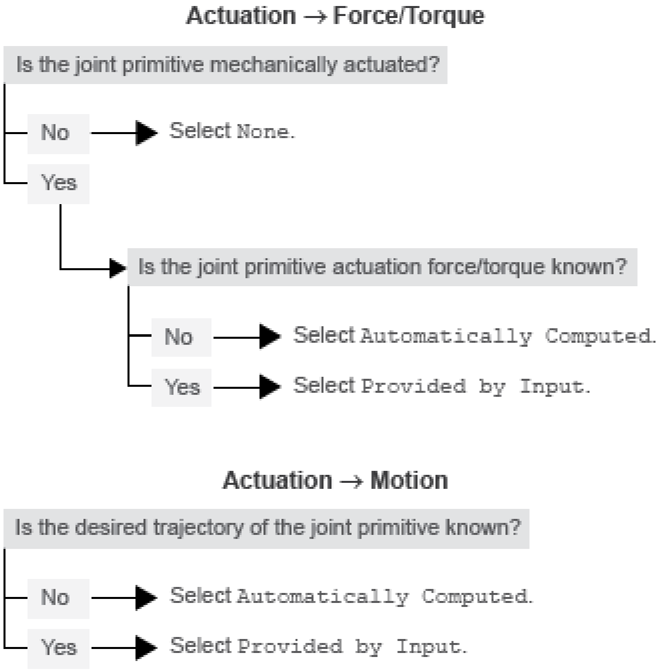
| | | Actuator Motion | |
|---|---|---|---|
| | | Provided by Input | Automatically Computed |
| Actuator Force/Torque | None | Unactuated Motion | Passive |
| | Provided by Input | Fully Specified | Forward Dynamics |
| | Automatically Computed | Inverse Dynamics | Fully Computed |

### Joint Actuation Modes

More generally, thinking of joint actuation in terms of the specified or calculated quantities—i.e., force/torque and motion—provides a more practical modeling approach. You may not always know the appropriate mode for a joint but, having planned the model beforehand, you should always know the answers to two questions:

- Is the joint primitive mechanically actuated?
- Is the desired trajectory of the joint primitive known?

By selecting the joint actuation settings based on the answers to these questions, you can ensure that each joint is properly set for your application. The figure shows the proper settings depending on your answers.


Actuation → Force/Torque
Is the joint primitive mechanically actuated?
No
Select None.
Yes
Is the joint primitive actuation force/torque known?
No
Select Automatically Computed.
Yes
Select Provided by Input.
Actuation → Motion
Is the desired trajectory of the joint primitive known?
No
Select Automatically Computed.
Yes
Select Provided by Input.

**Selecting Joint Primitive Actuation Settings**

## Motion Input

The motion input of a joint primitive is a timeseries object specifying that primitive's trajectory. For a prismatic primitive, that trajectory is the position coordinate along the primitive axis, given as a function of time. The coordinate provides the position of the follower frame origin with respect to the base frame origin. The primitive axis is resolved in the base frame.

For a revolute primitive, the trajectory is the angle about the primitive axis, given as a function of time. This angle provides the rotation of the follower frame with respect to the base frame about the primitive axis. The axis is resolved in the base frame.

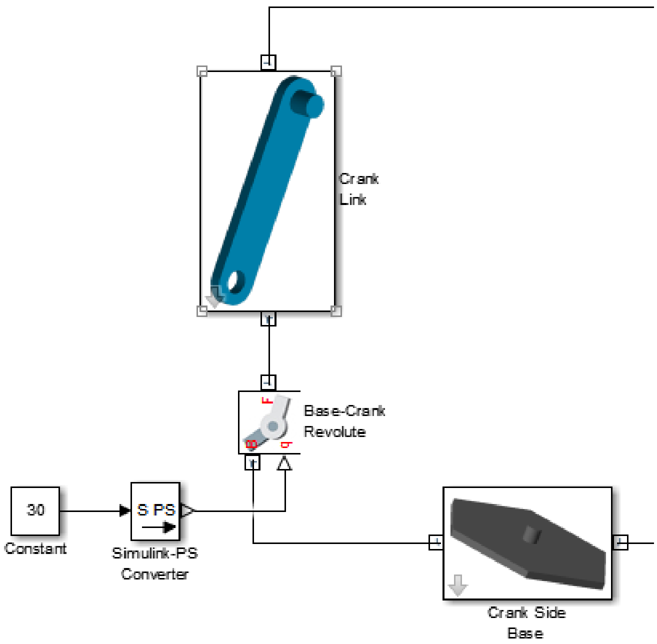
Spherical joint primitives provide no motion actuation options. You can specify actuation torque for these primitives, but you cannot prescribe their trajectories. Those trajectories are always automatically computed from the model dynamics during simulation.

### Zero Motion Prescription

Unlike **Actuation** > **Force/Torque**, the **Actuation** > **Motion** parameter provides no zero input option, corresponding to a fixed joint primitive during simulation. You can, however, prescribe zero motion the same way you prescribe all other types of motion: using Simscape and Simulink blocks.

In SimMechanics, motion input signals are position-centric. You specify the joint primitive position and, if filtered to the second-order, the `Simulink-PS Converter` block smooths the signal while providing its two time-derivatives automatically. This behavior makes zero motion prescription straightforward: just provide a constant signal to the motion actuation input port of the joint primitive and simulate.

The figure shows an example of zero-motion prescription. A Simulink Constant block provides a constant position value. A Simulink-PS Converter block converts this Simulink signal into a Simscape signal compatible with the motion actuation input port of the Base-Crank Revolute Joint block. Assuming that assembly and simulation are successful, this joint will maintain a fixed angle of 30 degrees, corresponding to the value set in the Simulink Constant block and the units set in the Simulink-PS Converter block.



## Input Handling

When prescribing a joint primitive trajectory, it is practical to specify a single input, the position, and filter that input using a Simulink-PS Converter block. This filter, which must of second-order, automatically provides the two time derivatives of the motion input. Because it also smooths the input signal, the filter can help prevent simulation issues due to sudden changes or discontinuities, such as those present when using a Simulink `Step` block.

Filtering smooths the input signal over a time scale of the order of the input filtering time constant. The larger the time constant, the greater the signal smoothing, and the more distorted the signal tends to become. The smaller the time constant, the closer the filtered signal is to the input signal, but also the greater the model stiffness—and, hence, the slower the simulation.

As a guideline, the input filtering time constant should be only as small as the smallest relevant time scale in a model. By default, its value is 0.001 s. While appropriate for many models, this value is often too small for SimMechanics models. For faster simulation, start with a value of 0.01 s. Decrease this value for greater accuracy.

If you know the two time derivatives of the motion input signal, you can specify them directly. This approach is most convenient for simple trajectories with simple derivatives. You must, however, ensure that the two derivative signals are compatible with the position signal. If they are not, even when simulation proceeds, results may be inaccurate.

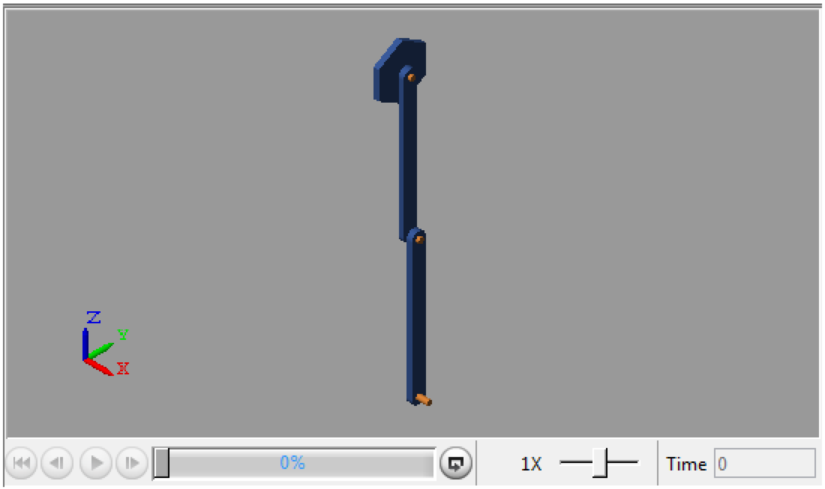
## Assembly and Simulation

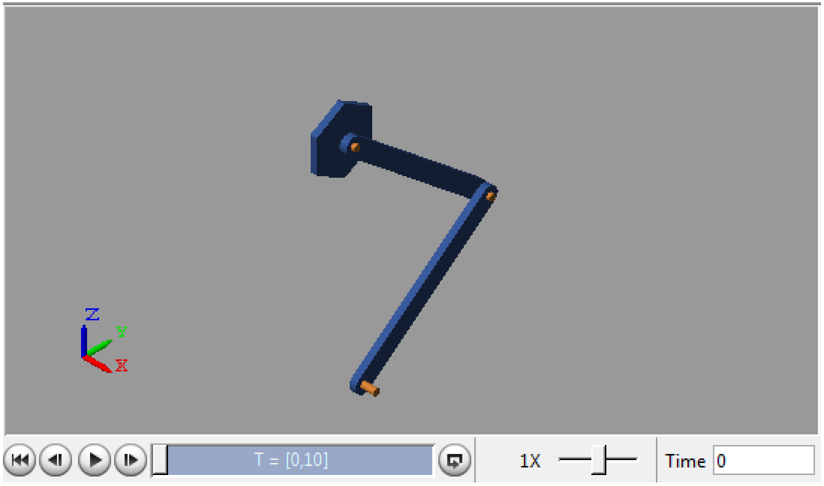
SimMechanics joints with motion inputs start simulation **(Ctrl+T)** at the initial position dictated by the input signal. This initial position may differ from the assembled state, which is governed by an assembly algorithm optimized to meet the joint state targets, if any. Even in the absence of joint state targets, the assembled state may differ from that at simulation time zero.

**Note:** You obtain the assembled state each time you update the block diagram, e.g., by pressing **Ctrl+D**. You obtain the initial simulation state each time you run the simulation, e.g., by pressing **Ctrl+T**, and pausing at time zero.

Due to the discrepancy between the two states, Model Report provides accurate initial state data only for models lacking motion inputs. For models possessing motion inputs, that data is accurate only when the initial position prescribed by the motion input signal exactly matches the initial position prescribed in the joint state targets.

Similarly, Mechanics Explorer displays the initial joint states accurately only for models lacking motion inputs. As it transitions from the assembled state to the initial simulation state, Mechanics Explorer may show a sudden jump if a model contains motion inputs that are incompatible with the joint state targets. You can eliminate the sudden change by making the initial position prescribed by joint motion inputs equal to the initial position prescribed by the joint state targets.





## Related Examples

- "Prescribe Joint Motion in Planar Manipulator Model" on page 4-108
- "Prescribe Joint Motion in Four-Bar Model" on page 4-103
- "Specify Motion Input Derivatives" on page 4-26

# Specify Motion Input Derivatives

If filtering the input signal using the Simulink-PS Converter block, you need only to provide the position signal. The block automatically computes the derivatives. You must, however, select second-order filtering in the block dialog box:

1. Open the dialog box of the Simulink-PS Converter block and click **Input Handling**.
2. In **Filtering and derivatives**, select `Filter input`.
3. In **Input filtering order**, select `Second-order filtering`.
4. In **Input filtering time constant (in seconds)**, enter the characteristic time over which filter smooths the signal. A good starting value is `0.01` seconds.

If providing the input derivatives directly, you must first compute those derivatives. Then, using the Simulink-PS Converter block, you can provide them to the target joint block. To specify the input derivatives directly:

1. Open the Simulink-PS Converter block receiving the input signal and click the **Input Handling** tab.
2. In **Filtering and derivatives**, select `Provide input derivative(s)`.
3. To specify both derivatives, in **Input derivatives**, select `Provide first and second derivatives`.

The block displays two additional physical signal ports, one for each derivative.

## Related Examples

- “Prescribe Joint Motion in Planar Manipulator Model” on page 4-108
- “Prescribe Joint Motion in Four-Bar Model” on page 4-103

## More About

- “Joint Actuation” on page 4-18

# Joint Actuation Limitations

**In this section...**

## Closed Loop Restriction

Each closed kinematic loop must contain at least one joint block without motion inputs or computed actuation force/torque. This condition applies even if one of the joints acts as a virtual joint, e.g., the bushing joint in the "Prescribe Joint Motion in Planar Manipulator Model" on page 4-108 example. The joint without motion inputs or automatically computed actuation forces/torques can still accept actuation forces/torques from input.

In models not meeting this condition, you can replace a rigid connection line between two Solid blocks with a Weld Joint block. Since the Weld Joint block represents a rigid connection, this approach leaves the model dynamics unchanged. The advantage of this approach lies in its ability to satisfy the SimMechanics closed-loop requirement without altering model dynamics.

## Motion Actuation Not Available in Spherical Primitives

Spherical joint primitives provide no motion actuation parameters. You can prescribe the actuation torque acting on the spherical primitive, but not its desired trajectory. For models requiring motion prescription for three concurrent rotational degrees of freedom, use joint blocks with three revolute primitives instead. These blocks include `Gimbal Joint`, `Bearing Joint`, and `Bushing Joint`.

## Redundant Actuation Mode Not Supported

Redundant actuation, in which the end effector trajectory of a high-degree-of-freedom linkage is prescribed, is not allowed. Such linkages possess more degrees of freedom than are necessary to uniquely position the end effector and, as such, have no single solution.

Models that have more degrees of freedom with automatically computed actuation forces/torques than with prescribed motion inputs cause simulation errors.

## Model Report and Mechanics Explorer Restrictions

In models with motion input, the assembled state achieved by updating the block diagram (**Ctrl+D**) does not generally match the initial simulation state at time zero **(Ctrl+T**). This discrepancy is visible in Mechanics Explorer, where it can cause a sudden state change at time zero when simulating a model after updating it. It is also reflected in Model Report, whose initial state data does not generally apply to the simulation time zero when a model has motion inputs.

## Motion-Controlled DOF Restriction

The number of degrees of freedom with prescribed trajectories must equal the number of degrees of freedom with automatically computed force or torque. In models not meeting this condition, simulation fails with an error.

## Related Examples

- “Prescribe Joint Motion in Planar Manipulator Model” on page 4-108
- “Prescribe Joint Motion in Four-Bar Model” on page 4-103
- “Specify Motion Input Derivatives” on page 4-26

## More About

- “Joint Actuation” on page 4-18

# Actuating and Sensing Using Physical Signals

| In this section... |
|---|
| |
| |
| |

Some SimMechanics blocks provide physical signal ports for actuation input or sensing output. These ports accept or output only Simscape physical signals. If you wish to connect these ports to Simulink blocks, you must use the Simscape converter blocks. The table summarizes the converter blocks that Simscape provides. You can find both blocks in the Simscape Utilities library.

| Block | Summary |
|---|---|
| PS-Simulink Converter | Convert Simscape physical signal into Simulink signal |
| Simulink-PS Converter | Convert Simulink signal into Simscape physical signal |

## Exposing Physical Signal Ports

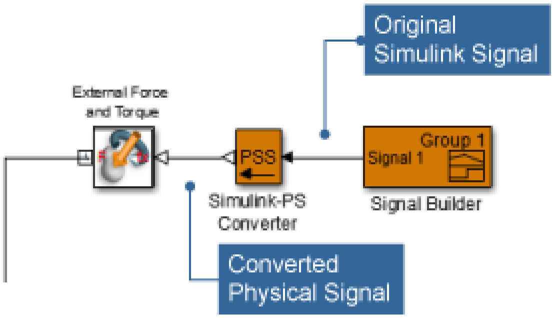
In SimMechanics, most physical signal ports are hidden by default. To expose them, you must select an actuation input or sensing output from the block dialog box. Blocks that provide physical signal ports include certain Forces and Torques blocks as well as Joint blocks. Each port has a unique label that identifies the actuation/sensing parameter. For the ports that a block provides, see the reference page for that block.

## Providing Actuation Signals

To provide an actuation signal based on Simulink blocks, you use the `Simulink-PS Converter` block:

1. Specify the desired actuation signal using Simulink blocks.
2. Connect the Simulink signal to the input port of a Simulink-PS Converter block.
3. Connect the output port of the Simulink-PS Converter block to the input port of the SimMechanics block that you want to provide the actuation signal to.

In the figure, the connection line that connects to the input port of the Simulink-PS Converter block represents the original Simulink signal. The connection line that connects to the output port of the same block represents the converted physical signal. This is the signal that you must connect to the actuation ports in SimMechanics blocks.

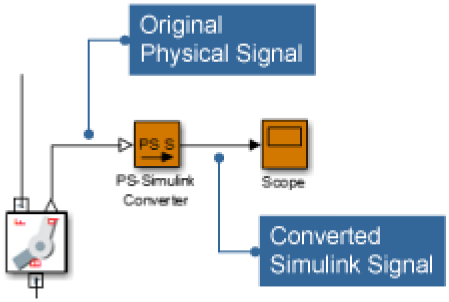


## Extracting Sensing Signals

To connect the sensing signal of a SimMechanics block to a Simulink block, you use the `PS-Simulink Converter` block:

1. Connect the SimMechanics sensing port to the input port of a PS-Simulink Converter block.
2. Connect the output port of the PS-Simulink Converter block to the Simulink block of your choice.

The figure shows how you can connect a SimMechanics sensing signal to a Simulink Scope block.


Original
Physical Signal
PS S
PS-Simulink
Converter
Scope
Converted
Simulink Signal

# Forces and Torques Between Arbitrary Bodies

| In this section... |
|---|
| "Force and Torque Blocks" on page 4-32 |
| "Actuating Rigid Bodies" on page 4-32 |

## Force and Torque Blocks

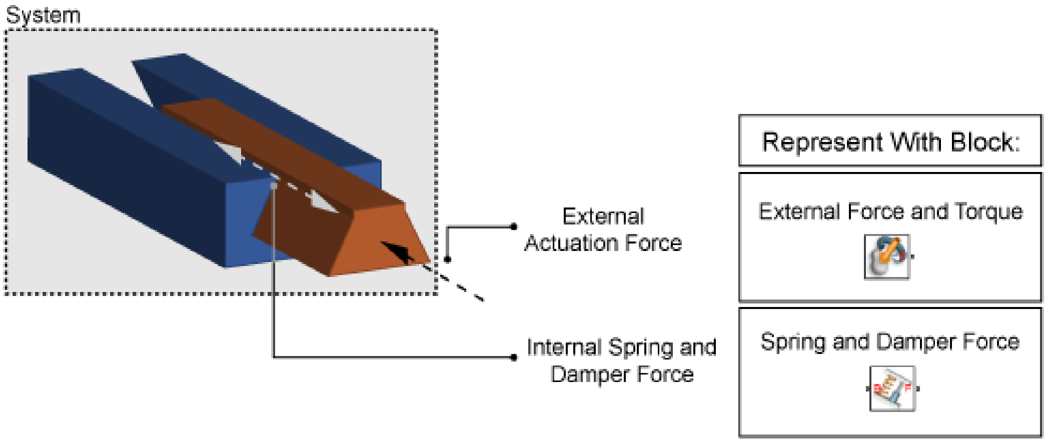
You can apply different forces and torques to a model. The table summarizes the different forces and torques that you can represent using SimMechanics blocks. For detailed information about these blocks, see the block reference pages.

| Block | Description |
|---|---|
| `External Force and Torque` | General force and torque acting on a frame origin due to an external source. You specify the force magnitude directly using a physical signal. |
| `Gravitational Field` | Gravitational field of a point mass. The field exerts on every rigid body a force proportional to the distance between the rigid-body center of mass and the Gravitational Field port frame. |
| `Internal Force` | General force between two frames. You specify the force magnitude directly using a physical signal. |
| `Inverse Square Law Force` | Force with an inverse dependence on the square distance between two frame origins. Examples include gravitational and electrostatic forces. |
| `Spring and Damper Force` | Force with a linear dependence on the relative position and velocity between two frame origins. |

## Actuating Rigid Bodies

You can actuate a rigid body directly using blocks from the Forces and Torques library. Use the External Force and Torque block to represent an actuation input that arises outside your modeled system. Use the remaining blocks to represent forces that are internal to your system.

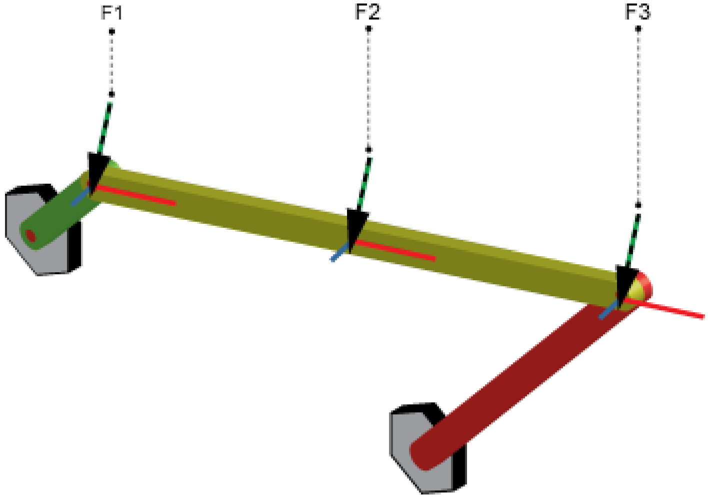
The figure illustrates external and internal forces acting on a mechanical system. An external force provides the actuation input to the system. This can be a constant or a general time-dependent input. A spring and damper force acting between the two bodies in the system accounts for energy storage and dissipation. You represent the actuation input using the External Force and Torque block. You represent the internal spring and damper force using the Spring and Damper Force block.



The Forces and Torques blocks contain frame ports. These ports identify the rigid body frames the forces/torques act on. If the block represents an internal force, the block contains two frame ports. Connect these ports to the two rigid bodies the force/torque acts on. If the block represents an external force or torque, the block contains one frame port. Connect this port to the rigid body frame the external force or torque acts on.

The frame origin identifies the point of application for a force or torque. The frame axes identify the directions of the X, Y, and Z force/torque vector components that you specify. Changing the frame position changes also the force/torque application point. Likewise, changing the frame orientation changes also the force/torque direction.

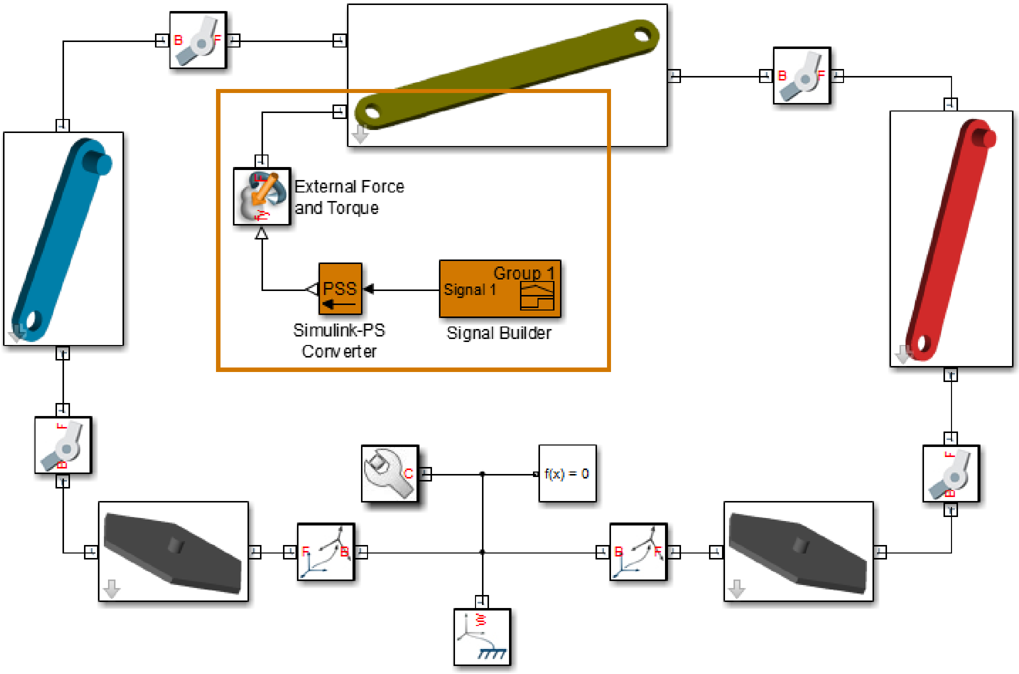
The figure shows three external forces that you can apply to the rocker link of a four-bar mechanism—F1, F2, and F3. Forces F1 and F3 act at the ends of the link, while force F2 acts at its mass center.



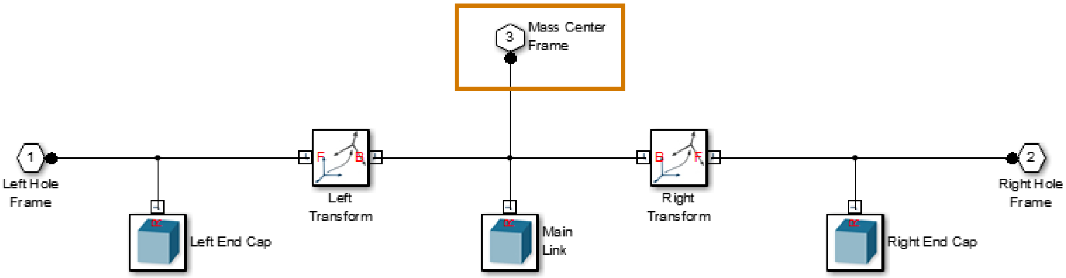
To represent one of these forces in a SimMechanics model, you first define the frame to apply that force to. Example "Represent Binary Link Frame Tree" on page 1-33 shows you how to do this. Then, in the block diagram for your model, connect the frame port of an External Force and Torque block to the frame entity that represents that frame—frame port, line, or node. For more information, see "Representing Frames" on page 1-6.

Finally, in the block dialog box, select the force components that you want to specify. For example, to specify a force acting along the -Y axis of the frame it connects to, select **Force** > **Force (Y)**. Then, use the physical signal port that the block exposes to input the value of that force component. That value is negative for a force acting along the -Y axis.

The figure shows the modified block diagram of a four-bar model that is present in your SimMechanics installation. You can open the original model by typing `sm_four_bar` at the MATLAB command line.



The rectangular frame in the image highlights the blocks that you can use to apply an external force. The frame port that the External Force and Torque block connects to represents the binary link mass center. The block diagram of the binary link subsystem provides this frame. The figure shows the block diagram.



In the External Force and Torque block, physical signal port fy identifies the force component that the block represents—in this case, a force in the Y direction of the frame that the block connects to.

## Related Examples

- “Actuate Joint in Four-Bar Model” on page 4-71

## More About

- “Joint Actuation” on page 4-18
- “Actuating and Sensing Using Physical Signals” on page 4-29
- “Representing Frames” on page 1-6

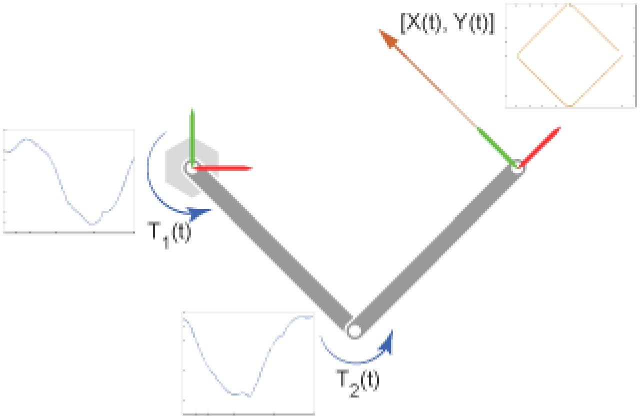
# Sensing

**In this section...**

## Sensing Overview

Sensing enables you to perform analytical tasks on a model. For example, you can perform inverse dynamic analysis on a robotic manipulator model. By prescribing the end-effector trajectory and sensing the joint actuation forces and torques, you can obtain the time-varying profile of each joint actuation input.

The variables you prescribe, the model inputs, and those you sense, the model outputs, determine which types of analysis you can perform. By changing the model inputs and outputs, you can perform numerous other analysis types. For example, to perform forward kinematic analysis on the robotic manipulator model, you can prescribe the manipulator joint trajectories and sense the resulting end-effector trajectory.



## Variables You Can Sense

To support various analytical tasks, SimMechanics software provides a wide range of variables that you can sense. Each variable belongs to either of two categories:

- Motion variables — Linear and angular position, velocity, and acceleration. Linear variables are available in different coordinate systems, including Cartesian, spherical, and cylindrical. Angular variables are available in different formats, including quaternion, axis-angle, and transform matrix.
- Force and torque variables — Actuation, constraint, and total forces and torques acting at a joint, as well as certain forces and torques acting outside of a joint.

## Blocks with Sensing Capability

The entire sensing capability spans multiple SimMechanics blocks. Two types of blocks provide motion sensing:

- Joint blocks — Motion sensing between the base and follower port frames of a joint block. Variables that you can sense are organized by joint primitive (prismatic, revolute, or spherical).
- Transform Sensor block — Motion sensing between any two frames in a model. This block provides the most comprehensive motion sensing capability in SimMechanics.

Two types of blocks provide force and torque sensing:

- Joint blocks — Actuation, constraint, and total force and torque sensing between the base and follower port frames. Actuation force and torque sensing is arranged by joint primitive.
- Constraint blocks — Constraint force and torque between the base and follower port frames.
- Certain Forces and Torques blocks — Total force the block exerts between the base and follower port frames. Only certain Forces and Torques blocks provide this type of sensing. Blocks that do include `Spring and Damper Force` and `Inverse Square Law Force`.

## Sensing Output Format

Each sensing output is in a physical signal format. You can convert physical signals into Simulink signals using Simscape converter blocks, e.g., for plotting purposes using the

Scope block. For information on how to use physical signals in SimMechanics models, see “Actuating and Sensing Using Physical Signals” on page 4-29.

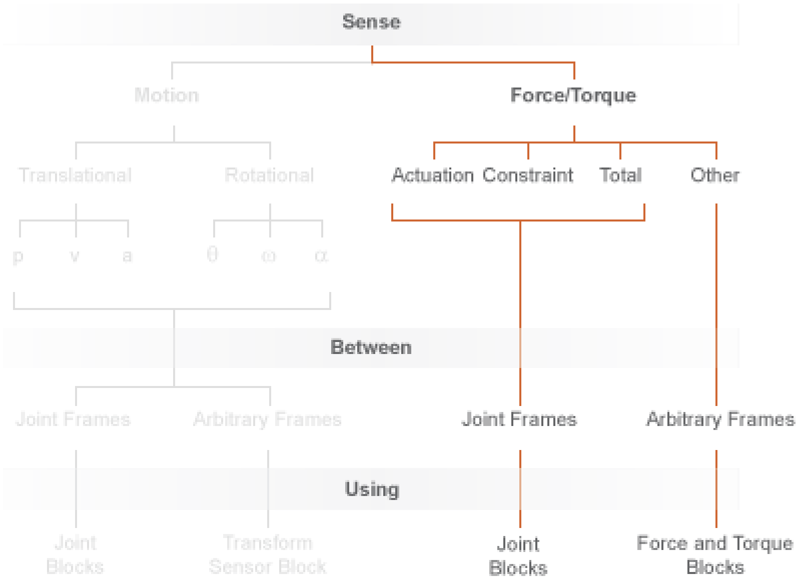
# Force and Torque Sensing

**In this section...**

## Blocks with Force and Torque Sensing

Blocks with force and torque sensing appear in two SimMechanics libraries:

- Forces and Torques — Sense the magnitude of certain forces not explicitly provided by input. Blocks with force sensing include `Inverse Square Law Force` and `Spring and Damper Force`. Each block can sense only the magnitude of its own force.
- Joints — Sense various forces and torques acting directly at a joint. All joint blocks provide force and torque sensing. However, the specific force and torque types that you can sense vary from joint to joint. Force and torque sensing is available strictly between the rigid bodies the joint connects.



**Force and Torque Sensing in SimMechanics**

## Joint Forces and Torques You can Sense

Forces and torques that you can sense at a joint fall into two categories:

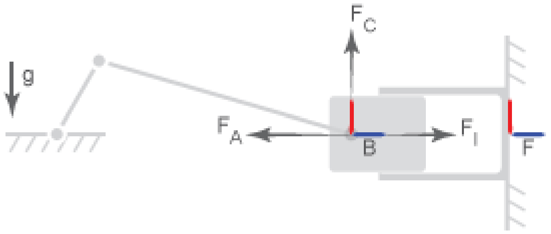
- Joint primitive forces and torques. Each such force or torque is individually computed for a given joint primitive. Joint actuator forces and torques belong to this category.
- Composite forces and torques. Each such force or torque is computed in aggregate for an entire joint. Constraint and total forces and torques belong to this category.

The table summarizes the different joint forces and torques.

| Force/Torque Type | Acts On | Measures |
|---|---|---|
| Actuator | Individual joint primitives | Force or torque driving an individual joint primitive. The sensed force or torque can be provided by input |

| Force/Torque Type | Acts On | Measures |
|---|---|---|
| | | or it can be automatically computed based on joint motion inputs in a model. |
| Constraint | Entire joints | Aggregate constraint force or torque opposing motion normal to the joint degrees of freedom. By definition, these forces and torques act orthogonally to the joint primitive axes. |
| Total | Entire joints | Net sum of all forces or torques acting between the joint port frames. These include actuator, internal, and constraint forces and torques. |

The figure shows a basic example of these forces acting on a crank-slider piston.



In the figure:

- $F_A$ is the actuator force, which drives the piston toward the crank link.
- $F_I$ is the internal spring and damper force, which resists motion of the piston with respect to the chamber.
- $F_C$ is the constraint force, which opposes the effect of gravity on the piston, preventing it from falling.

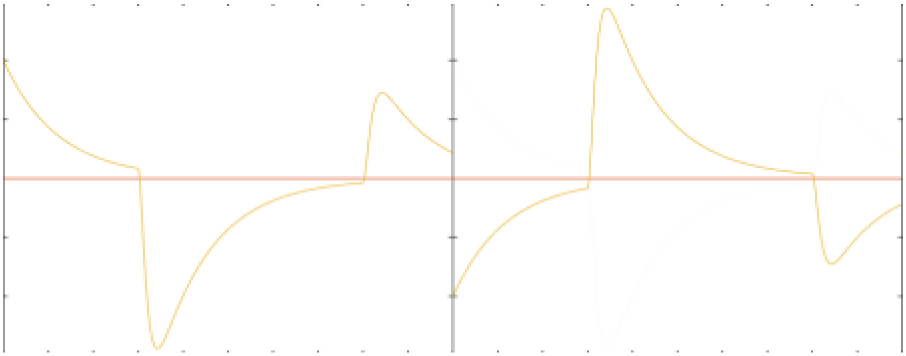
The total force equals the net sum of $F_A$, $F_I$, and $F_C$.

## Force and Torque Measurement Direction

In accordance with Newton's third law of motion, a force or torque acting between two joint port frames accompanies an equal and opposite force or torque. If the base port frame of a Prismatic Joint block exerts a force on the follower port frame, then the follower port frame exerts an equal force on the base frame. When sensing composite forces and torques in joint blocks, you can specify which of the two to sense:

- Follower on base — Sense the force or torque that the follower port frame exerts on the base port frame.
- Base on follower — Sense the force or torque that the base port frame exerts on the follower port frame.

The figure shows the effect of reversing the measurement direction. Reversing this direction changes the measurement sign.



# Motion Sensing

**In this section...**

In SimMechanics, you can sense the spatial relationship between two frames using two types of blocks:

- `Transform Sensor` — Sense the spatial relationship between any two frames in a model. Parameters that you can sense with this block include position, velocity, and acceleration of the linear and angular types. This block provides the most extensive motion sensing capability in the SimMechanics libraries.
- Joint blocks — Sense the spatial relationship between the base and follower frames of a Joint block. Parameters that you can sense with a Joint block include the position and its first two time derivatives (velocity and acceleration) for each joint primitive.

These blocks output a physical signal for each measurement that you specify. You can use the sensing output of these blocks for analysis or as input to a control system in a model.

## Sensing Spatial Relationship Between Joint Frames

To sense the spatial relationship between the base and follower frames of a Joint block, you can use the Joint block itself. For each joint primitive, the dialog box provides a **Sensing** menu with basic parameters that you can measure. These parameters include the position, velocity, and acceleration of the follower frame with respect to the base frame. If the sensing menu of the dialog box does not provide the parameters that you wish to sense, use the Transform Sensor block instead. See "Sensing Spatial Relationship Between Arbitrary Frames" on page 4-46.

The sensing capability of a joint block is limited to the base and follower frames of that joint block. Every measurement provides the value of a parameter for the joint follower frame with respect to the joint base frame. If sensing the spatial relationship with a spherical joint primitive, you can also select the frame to resolve the measurement in. To sense the spatial relationship between any other two frames, use the Transform Sensor block instead.

If the joint primitive is of the revolute or spherical type, the parameters correspond to the rotation angle, angular velocity, and angular acceleration, respectively. If the joint primitive is of the prismatic type, the parameters correspond to the offset distance, linear velocity, and linear acceleration, respectively.

Regardless of joint primitive type, each parameter that you select applies only to the joint primitive it belongs to. For example, selecting **Position** in the **Z Revolute Primitive (Rz)** > **Sensing** menu exposes a physical signal port that outputs the rotation angle of the follower frame with respect to the base frame *about the base frame Z axis*.

The table lists the port label for each parameter that you can sense using a joint block. The first column of the table identifies the parameters that you can select. The remaining three columns identify the port labels for the three joint primitive menus that the dialog box can contain: **Spherical**, **Revolute**, and **Prismatic**.

**Note:** For parameter descriptions, see the reference pages for `Spherical Joint`, `Revolute Joint`, and `Prismatic Joint` blocks.

| Parameter | Spherical | Revolute | Prismatic |
|---|---|---|---|
| Position | Q | q | p |
| Velocity | w | w | v |
| Velocity (X/Y/Z) | wx/wy/wz | N/A | N/A |
| Acceleration | b | b | a |
| Acceleration (X/Y/Z) | bx/by/bz | N/A | N/A |

A joint block can contain multiple revolute and prismatic joint primitives. For blocks with multiple primitives of the same type, the port labels include an extra letter identifying the joint primitive axis. For example, the **Position** port label for the Z prismatic primitive of a Cartesian Joint block is pz.

### Select Joint Parameters To Sense

To select the spatial relationship parameters that you wish to sense:

1 Open the dialog box for the joint block to sense the spatial relationship across.

2 In the **Sensing** menu of the block dialog box, select the parameters to sense.

The block exposes one physical signal port for each parameter that you select. The label of each port identifies the parameter that port outputs.

## Sensing Spatial Relationship Between Arbitrary Frames

To sense the spatial relationship between two arbitrary frames in a model, you use the Transform Sensor block. The dialog box of this block provides a set of menus that you can use to select the parameters to sense. These parameters include position, velocity, and acceleration of the linear and angular types.

Every measurement provides the value of a parameter for the follower frame with respect to the base frame, resolved in the measurement frame that you choose. You can connect the base and follower frame ports of the Transform Sensor block to any two frames in a model. To measure a parameter for a different frame, connect the follower frame port to the frame line or port that identifies that frame. Likewise, to measure a parameter for the same frame but with respect to a different frame, connect the base frame port to the frame line or port that identifies that frame. Finally, to resolve a measurement in a different frame, select a different measurement frame in the block dialog box. For more information about measurement frames, see “Measurement Frames” on page 4-62. For more information about frame lines and ports, see “Representing Frames” on page 1-6.

Selecting a parameter from the block dialog box exposes the corresponding physical signal port in the block. Use this port to output the measurement for that parameter. To identify the port associated with each parameter, each port uses a unique label.

The table lists the port labels for each angular parameter that you can sense. The first column of the table identifies the parameters that you can select. The remaining three columns identify the port labels for the three angular parameter menus in the dialog box: **Rotation**, **Angular Velocity**, and **Angular Acceleration**. Certain parameters belong to one menu but not to others. N/A identifies the parameters that do not belong to a given menu—e.g. Angle, which is absent from the Angular Velocity.

**Note:** For parameter descriptions, see the `Transform Sensor` reference page.

| Parameter | Rotation | Angular Velocity | Angular Acceleration |
|---|---|---|---|
| Angle | q | N/A | N/A |
| Axis | axs | N/A | N/A |
| Quaternion | Q | Qd | Qdd |
| Transform | R | Rd | Rdd |

| Parameter | Rotation | Angular Velocity | Angular Acceleration |
|---|---|---|---|
| Omega X/Omega Y/ Omega Z | N/A | wx/wy/wz | N/A |
| Alpha X/Alpha Y/ Alpha Z | N/A | N/A | bx/by/bz |

The table lists the port labels for each linear parameter that you can sense. As in the previous table, the first column identifies the parameters that you can select. The remaining three columns identify the port labels for the three linear parameter menus in the dialog box: **Translation**, **Velocity**, and **Acceleration**.

| Parameter | Rotation Port | Angular Velocity Port | Angular Acceleration Port |
|---|---|---|---|
| X/Y/Z | x/y/z | vx/vy/vz | ax/ay/az |
| Radius | rad | vrad | arad |
| Azimuth | azm | vazm | aazm |
| Distance | dst | vdst | adst |
| Inclination | inc | vinc | ainc |

### Select Transform Sensor Parameters To Sense

To select the spatial relationship parameters that you wish to sense:

1 Open the Transform Sensor dialog box.

2 Expand the menu for the parameter group that parameter belongs to.

E.g. **Rotation** for parameter **Angle**.

3 Select the check box for that parameter.

The block exposes one physical signal port for each parameter that you select. The label of each port identifies the parameter that port outputs.

## Related Examples

- “Sense Motion in Double-Pendulum Model” on page 4-65
- “Actuate Joint in Four-Bar Model” on page 4-71

## More About

- “Rotational Measurements” on page 4-49
- “Translational Measurements” on page 4-54
- “Measurement Frames” on page 4-62

# Rotational Measurements

**In this section...**

## Rotation Sensing Overview

You can measure frame rotation in different formats. These include axis-angle, quaternion, and transform. The different formats are available through the `Transform Sensor` block and, to a limited extent, in joint blocks [1]. The choice of measurement format depends on the model. Select the format that is most convenient for the application.

## Measuring Rotation

Rotation is a relative quantity. The rotation of one frame is meaningful only with respect to another frame. As such, blocks with rotation sensing capability require two frames to make a measurement: measured and reference frames. In these blocks, the follower frame port identifies the measured frame; the base frame port identifies the reference frame of the measurement.

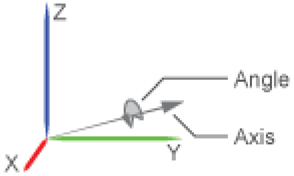
SimMechanics defines the rotation formats according to standard conventions. In some cases, more than one convention exists. This is the case, for example, of the quaternion. To properly interpret rotation measurements, review the definitions of the rotation formats.

## Axis-Angle Measurements

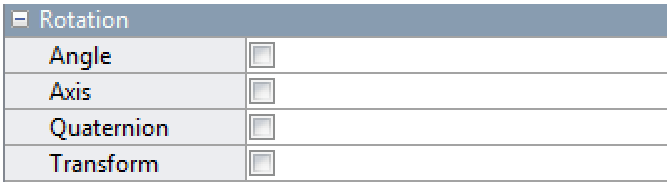
Axis-angle is one of the simpler rotation measurement formats. This format uses two parameters to completely describe a rotation: axis vector and angle. The usefulness of the axis-angle format follows directly from Euler's rotation theorem. According to the

[1] Weld Joint is an exception

theorem, any 3–D rotation or rotation sequence can be described as a pure rotation about a single fixed axis.



To measure frame rotation in axis-angle format, use the Transform Sensor block. The block dialog box contains separate **Axis** and **Angle** parameters that you can select to expose the corresponding physical signal (PS) ports (labeled axs and q, respectively). Because the axis-angle parameters are listed separately, you can choose to measure the axis, the angle, or both.



⊟ Rotation

| | |
|---|---|
| Angle | □ |
| Axis | □ |
| Quaternion | □ |
| Transform | □ |

The axis output is a 3–D unit vector in the form [$a_x$, $a_y$, $a_z$]. This unit vector encodes the rotation direction according to the right-hand rule. For example, a frame spinning in a counterclockwise direction about the +X axis has rotation axis [1 0 0]. A frame spinning in a clockwise direction about the same axis has rotation axis [-1 0 0].

The angle output is a scalar number in the range 0–π. This number encodes the extent of rotation about the measured axis. By default, the angle is measured in radians. You can change the angle units in the `PS-Simulink Converter` block used to interface with Simulink blocks.

## Quaternion Measurements

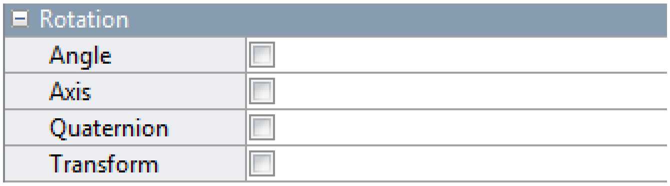
The quaternion is a rotation representation based on hypercomplex numbers. This representation uses a 4-vector containing one scalar ($S$) and three vector components ($V_x$,

$V_y$, $V_z$). The scalar component encodes the rotation angle. The vector components encode the rotation axis.

A key advantage of quaternions is the singularity-free parameter space. Mathematical singularities, present in Euler angle sequences, result in the loss of rotational degrees of freedom. This phenomenon is known as gimbal lock. In SimMechanics, gimbal lock causes numerical errors that lead to simulation failure. The absence of singularities means that quaternions are more robust for simulation purposes.

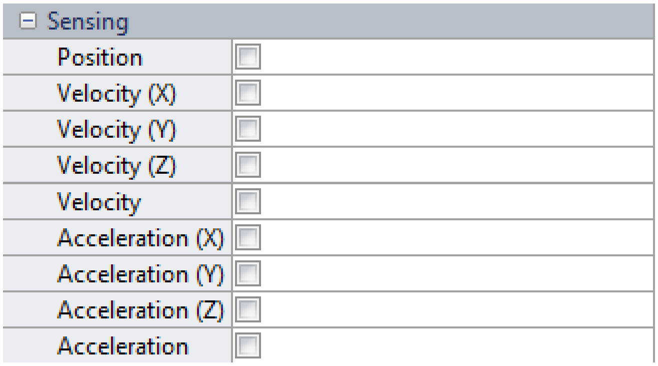
To measure frame rotation in quaternion format, use:

- Transform Sensor block, if measuring rotation between two general frames. The **Rotation** menu of the dialog box contains a **Quaternion** option that you can select to expose the corresponding physical signal port (labeled Q).



| ⊟ Rotation | |
|---|---|
| Angle | □ |
| Axis | □ |
| Quaternion | □ |
| Transform | □ |

- Joint block possessing spherical primitive, if measuring 3–D rotation between the two joint frames. The **Sensing** menu of the dialog box contains a **Position** option that you can select to expose the corresponding physical signal port (also labeled Q). For more information, see `Spherical Joint` block reference page.



| ⊟ Sensing | |
|---|---|
| Position | □ |
| Velocity (X) | □ |
| Velocity (Y) | □ |
| Velocity (Z) | □ |
| Velocity | □ |
| Acceleration (X) | □ |
| Acceleration (Y) | □ |
| Acceleration (Z) | □ |
| Acceleration | □ |

The quaternion output is a 4-element row vector $Q = \left( S \quad \mathbf{V} \right)$, where:

$$S = \cos\left(\theta/2\right)$$

and

$$\mathbf{V} = [V_x V_y V_z]\sin\left(\frac{\theta}{2}\right)$$

θ is the rotation angle. The angle can take any value between 0–π. [$V_x$, $V_y$, $V_z$] is the rotation axis. Axis components can take any value between 0–1.

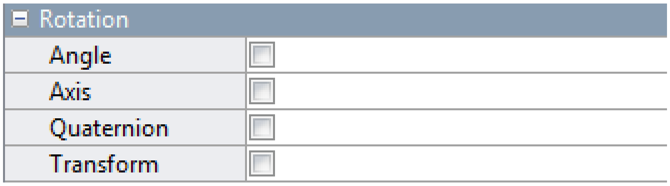
## Transform Measurements

The rotation transform is a 3×3 matrix that encodes frame rotation. In terms of base frame axes $[x, y, z]_{\mathrm{B}}$, the follower frame axes $[x, y, z]_{\mathrm{F}}$ are:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_B = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}_F$$

Each matrix column contains the coordinates of a follower frame axis resolved in the base frame. For example, the first column contains the coordinates of the follower frame X-axis, as resolved in the base frame. Similarly, the second and third columns contain the coordinates of the Y and Z-axes, respectively. Operating on a vector with the rotation matrix transforms the vector coordinates from the follower frame to the base frame.

You can sense frame rotation in terms of a rotation matrix using the Transform Sensor block. The dialog box for this block contains a **Transform** option that when selected exposes a physical signal port labeled R. Use this port to output the rotation matrix signal, for example, for processing and analysis in a Simulink subsystem—after converting the output physical signal to a Simulink signal through the PS-Simulink Converter block.



⊟ Rotation

| | |
|---|---|
| Angle | □ |
| Axis | □ |
| Quaternion | □ |
| Transform | □ |

## Related Examples

- “Sense Motion in Double-Pendulum Model” on page 4-65
- “Actuate Joint in Four-Bar Model” on page 4-71

## More About

- “Motion Sensing” on page 4-44
- “Translational Measurements” on page 4-54
- “Measurement Frames” on page 4-62

# Translational Measurements

**In this section...**

## Translation Sensing Overview

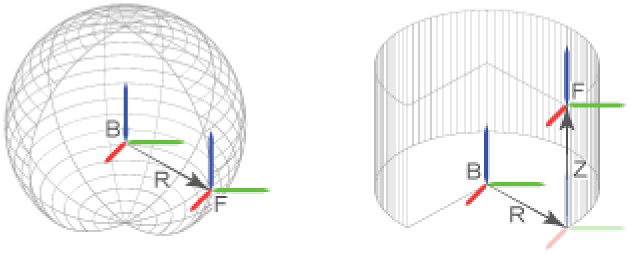
You can measure frame translation in different coordinate systems. These include Cartesian, cylindrical, and spherical systems. The different coordinate systems are available through the `Transform Sensor` block and, to a limited extent, through the Joint blocks. The choice of coordinate system depends on the model. Select the coordinate system that is most convenient for your application.

## Measuring Translation

Translation is a relative quantity. The translation of one frame is meaningful only with respect to another frame. As such, blocks with translation sensing capability require two frames to make a measurement: measured and reference frames. In these blocks, the follower frame port identifies the measured frame; the base frame port identifies the reference frame of the measurement.

Some measurements are common to multiple coordinate systems. One example is the Z-coordinate, which exists in both Cartesian and cylindrical systems. In the Transform Sensor dialog box, coordinates that make up more than one coordinate system appear only once. Selecting **Z** outputs translation along the Z-axis in both Cartesian and cylindrical coordinate systems.

Other measurements are different but share the same name. For example, radius is a coordinate in both spherical and cylindrical systems. The spherical radius is different from the cylindrical radius: the former is the distance between two frame origins; the latter is the distance between one frame origin and a frame Z-axis.

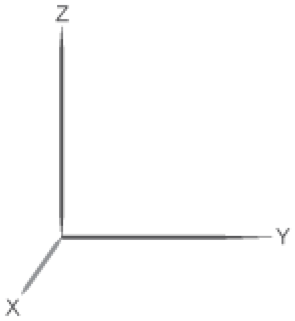


To differentiate between the two radial coordinates, SimMechanics uses the following convention:

- Radius — Cylindrical radial coordinate
- Distance — Spherical radial coordinate

## Cartesian Measurements

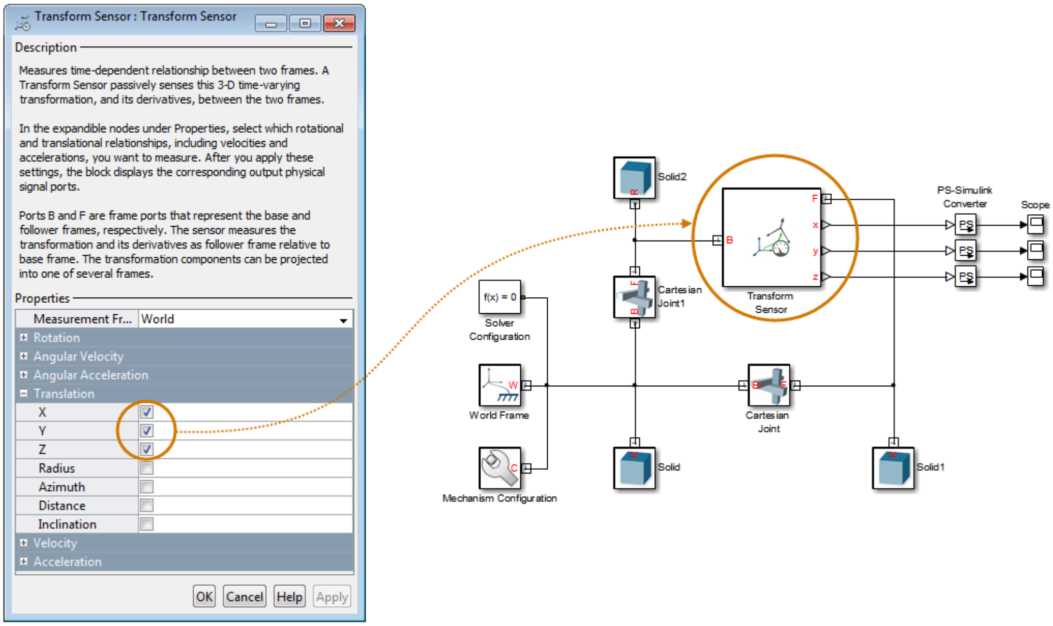
The Cartesian coordinate system uses three linear coordinates—X, Y, and Z—corresponding to three mutually orthogonal axes. Cartesian translation measurements have units of distance, with meter being the default. You can use the `PS-Simulink Converter` block to select a different physical unit when interfacing with Simulink blocks.



### Transform Sensor

You can select any of the Cartesian axes in the Transform Sensor for translation sensing. This is true even if translation is constrained along any of the Cartesian axes. Selecting the Cartesian axes exposes physical signal ports x, y, and z, respectively.

The figure shows a simple model using a Transform Sensor block to measure frame translation along all three Cartesian axes. The measurement gives the relative translation of the follower port frame with respect to the base port frame. These frames are, respectively, the Solid1 and Solid2 reference port frames. For more information, see "Representing Frames" on page 1-6.



Transform Sensor : Transform Sensor

Description

Measures time-dependent relationship between two frames. A Transform Sensor passively senses this 3-D time-varying transformation, and its derivatives, between the two frames.

In the expandible nodes under Properties, select which rotational and translational relationships, including velocities and accelerations, you want to measure. After you apply these settings, the block displays the corresponding output physical signal ports.

Ports B and F are frame ports that represent the base and follower frames, respectively. The sensor measures the transformation and its derivatives as follower frame relative to base frame. The transformation components can be projected into one of several frames.

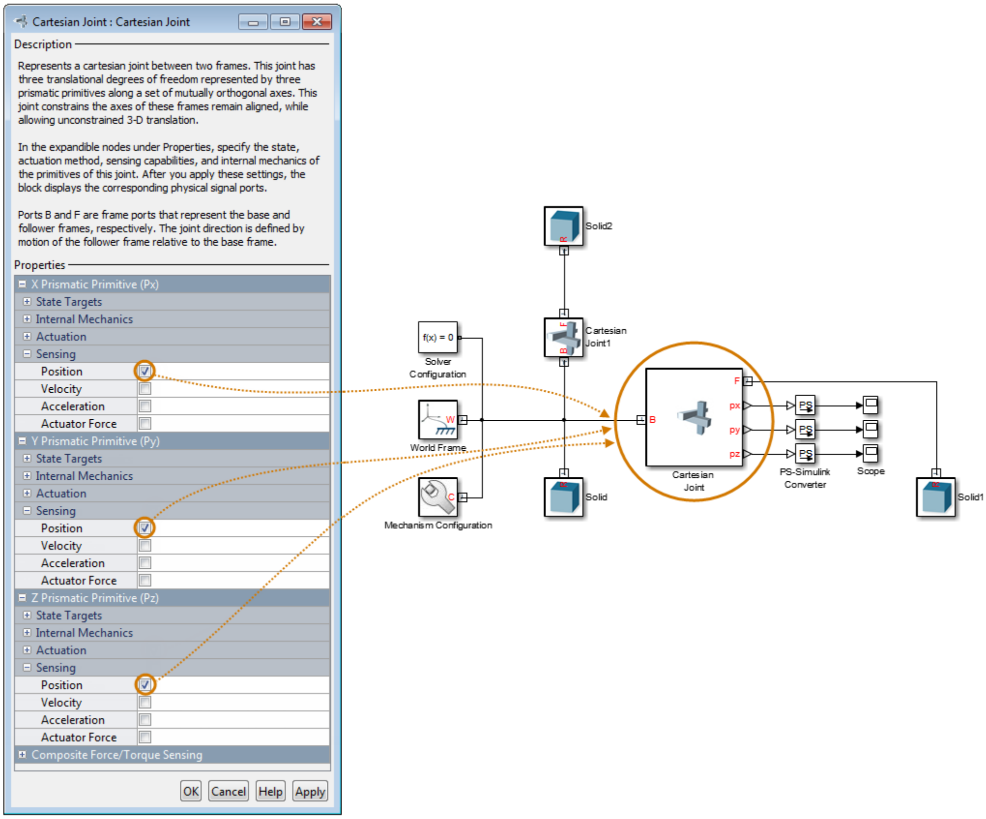
Properties

| | |
|---|---|
| Measurement Fr... | World |
| ⊞ Rotation | |
| ⊞ Angular Velocity | |
| ⊞ Angular Acceleration | |
| ⊟ Translation | |
| X | ☑ |
| Y | ☑ |
| Z | ☑ |
| Radius | □ |
| Azimuth | □ |
| Distance | □ |
| Inclination | □ |
| ⊞ Velocity | |
| ⊞ Acceleration | |

### Joints

With joint blocks, you can sense translation along each prismatic primitive axis. Selecting a sensing parameter from a prismatic primitive menu exposes the corresponding physical signal port. For example, if you select **Position** from the **Z Prismatic Primitive (Pz)** of a Cartesian Joint block, the block exposes physical signal port z.

The figure shows a simple model using a `Cartesian Joint` block to sense frame translation along the three Cartesian axes. The measurement gives the relative translation of the follower port frame with respect to the base port frame. These frames coincide with the Solid1 and Solid reference port frames.



Cartesian Joint : Cartesian Joint

Description

Represents a cartesian joint between two frames. This joint has three translational degrees of freedom represented by three prismatic primitives along a set of mutually orthogonal axes. This joint constrains the axes of these frames remain aligned, while allowing unconstrained 3-D translation.

In the expandible nodes under Properties, specify the state, actuation method, sensing capabilities, and internal mechanics of the primitives of this joint. After you apply these settings, the block displays the corresponding physical signal ports.

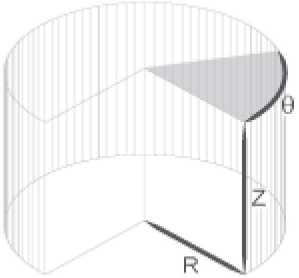
Ports B and F are frame ports that represent the base and follower frames, respectively. The joint direction is defined by motion of the follower frame relative to the base frame.

Properties

- X Prismatic Primitive (Px)
  - State Targets
  - Internal Mechanics
  - Actuation
  - Sensing
    - Position ☑
    - Velocity □
    - Acceleration □
    - Actuator Force □
- Y Prismatic Primitive (Py)
  - State Targets
  - Internal Mechanics
  - Actuation
  - Sensing
    - Position ☑
    - Velocity □
    - Acceleration □
    - Actuator Force □
- Z Prismatic Primitive (Pz)
  - State Targets
  - Internal Mechanics
  - Actuation
  - Sensing
    - Position ☑
    - Velocity □
    - Acceleration □
    - Actuator Force □
- Composite Force/Torque Sensing

## Cylindrical Measurements

The cylindrical coordinate system uses one angular and two linear coordinates. The linear coordinates are the cylinder radius, R, and length, Z. The angular coordinate is the azimuth, ϕ, about the length axis. Linear coordinates have units of distance, with meter being the default. The angular coordinate has units of angle, with radian being the default. You can use the PS-Simulink Converter block to select a different physical unit when interfacing with Simulink blocks.

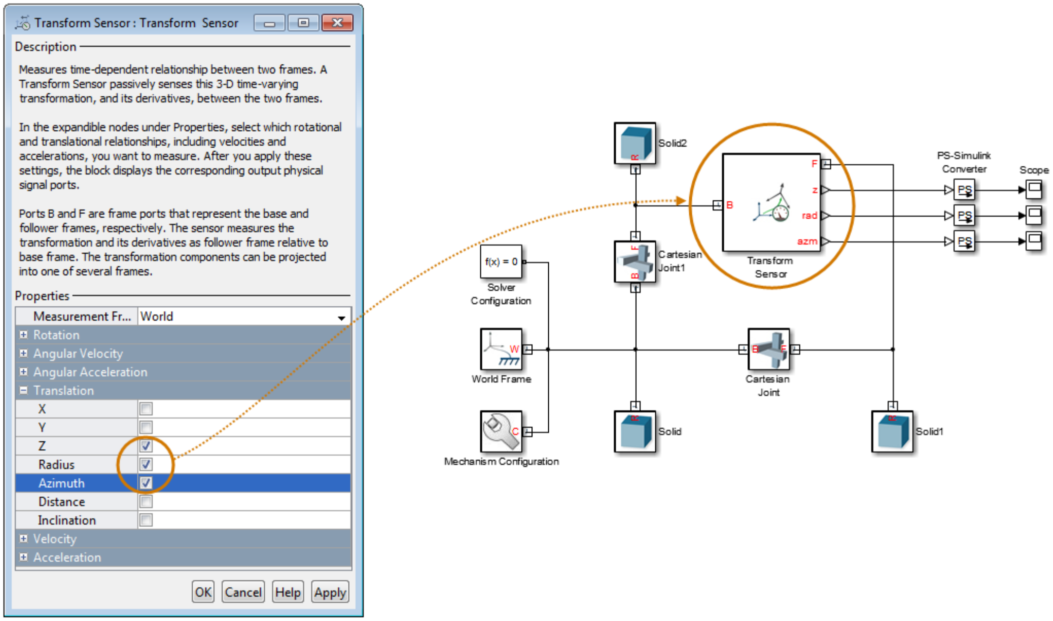


#### Transform Sensor

Only the Transform Sensor block can sense frame translation in cylindrical coordinates. In the dialog box of this block, you can select one or more cylindrical coordinates to measure. The cylindrical coordinates are named **Z**, **Radius**, and **Azimuth**. Selecting the cylindrical coordinates exposes physical signal ports z, rad, and azm, respectively.

**Note:** **Z** belongs to both Cartesian and cylindrical systems.

The figure shows a simple model using a Transform Sensor block to measure frame translation along all three cylindrical axes. The measurement gives the relative translation of the follower port frame with respect to the base port frame. These frames are, respectively, the Solid1 and Solid2 reference port frames.



Transform Sensor : Transform Sensor

Description

Measures time-dependent relationship between two frames. A Transform Sensor passively senses this 3-D time-varying transformation, and its derivatives, between the two frames.

In the expandible nodes under Properties, select which rotational and translational relationships, including velocities and accelerations, you want to measure. After you apply these settings, the block displays the corresponding output physical signal ports.

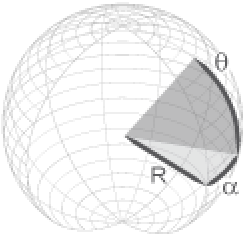
Ports B and F are frame ports that represent the base and follower frames, respectively. The sensor measures the transformation and its derivatives as follower frame relative to base frame. The transformation components can be projected into one of several frames.

Properties

| | |
|---|---|
| Measurement Fr... | World |
| ⊞ Rotation | |
| ⊞ Angular Velocity | |
| ⊞ Angular Acceleration | |
| ⊟ Translation | |
| X | □ |
| Y | □ |
| Z | ☑ |
| Radius | ☑ |
| Azimuth | ☑ |
| Distance | □ |
| Inclination | □ |
| ⊞ Velocity | |
| ⊞ Acceleration | |

## Spherical Measurements

The spherical coordinate system uses two angular coordinates and one linear coordinate. The linear coordinate is the spherical radius, R. The angular coordinates are the azimuth, ϕ, and inclination, θ. The linear coordinate has units of distance, with meter being the default. The angular coordinates have units of angle, with radian being the default. You can use the PS-Simulink Converter block to select a different physical unit when interfacing with Simulink blocks.



### Transform Sensor

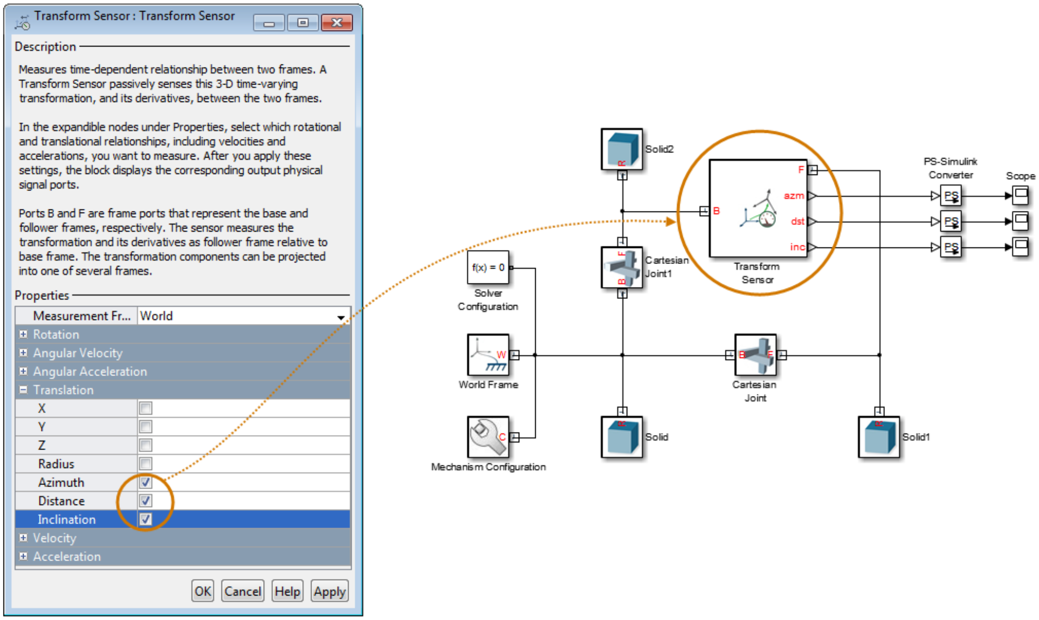
Only the Transform Sensor block can sense frame translation in spherical coordinates. In the dialog box of this block, you can select one or more spherical coordinates to measure. The spherical coordinates are named **Azimuth**, **Distance**, and **Inclination**. Selecting the spherical coordinates exposes physical signal ports azm, dst, and inc, respectively.

---

**Note: Azimuth** belongs to both cylindrical and spherical systems. **Distance** is the spherical radius.

---

The figure shows a simple model using a Transform Sensor block to measure frame translation along all three spherical axes. The measurement gives the relative translation of the follower port frame with respect to the base port frame. These frames are, respectively, the Solid1 and Solid2 reference port frames.



Transform Sensor : Transform Sensor

Description

Measures time-dependent relationship between two frames. A Transform Sensor passively senses this 3-D time-varying transformation, and its derivatives, between the two frames.

In the expandible nodes under Properties, select which rotational and translational relationships, including velocities and accelerations, you want to measure. After you apply these settings, the block displays the corresponding output physical signal ports.

Ports B and F are frame ports that represent the base and follower frames, respectively. The sensor measures the transformation and its derivatives as follower frame relative to base frame. The transformation components can be projected into one of several frames.

Properties

| | |
|---|---|
| Measurement Fr... | World |
| ⊞ Rotation | |
| ⊞ Angular Velocity | |
| ⊞ Angular Acceleration | |
| ⊟ Translation | |
| X | □ |
| Y | □ |
| Z | □ |
| Radius | □ |
| Azimuth | ☑ |
| Distance | ☑ |
| Inclination | ☑ |
| ⊞ Velocity | |
| ⊞ Acceleration | |

## Related Examples

- “Sense Motion in Double-Pendulum Model” on page 4-65
- “Actuate Joint in Four-Bar Model” on page 4-71

## More About

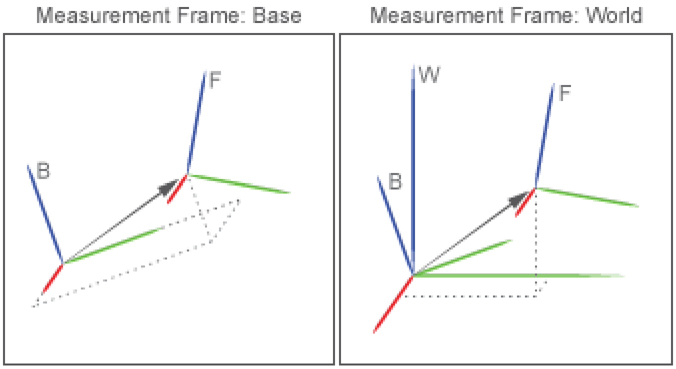
- “Motion Sensing” on page 4-44
- “Rotational Measurements” on page 4-49
- “Measurement Frames” on page 4-62

# Measurement Frames

You can sense the spatial relationship between two frames. When you do so, SimMechanics 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 SimMechanics 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. SimMechanics 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, SimMechanics resolves that translation vector along the axes of the base frame. If you select the World frame as the measurement frame, SimMechanics instead resolves the translation vector along the axes of the World frame. The translation vector remains the same, but the frame SimMechanics 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

SimMechanics 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.<br>A non-rotating frame is a virtual frame which, at every point in time, SimMechanics holds coincident with the rotating frame, but which has zero angular velocity with respect to the World frame.<br>Measurements that can differ between rotating and non-rotating frames are the linear velocity and linear acceleration. |

## Related Examples

- "Sense Motion in Double-Pendulum Model" on page 4-65
- "Actuate Joint in Four-Bar Model" on page 4-71

## More About

- "Motion Sensing" on page 4-44
- "Rotational Measurements" on page 4-49
- "Translational Measurements" on page 4-54

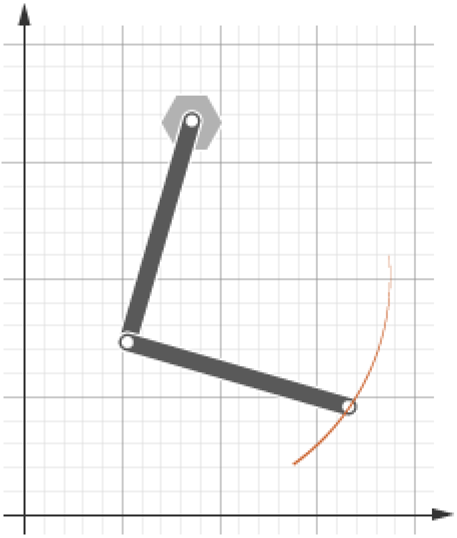
# Sense Motion in Double-Pendulum Model

**In this section...**

## Model Overview

The Transform Sensor block provides the broadest motion-sensing capability in SimMechanics 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 Double Pendulum” on page 3-14.
2. Drag these blocks into the model to sense motion.

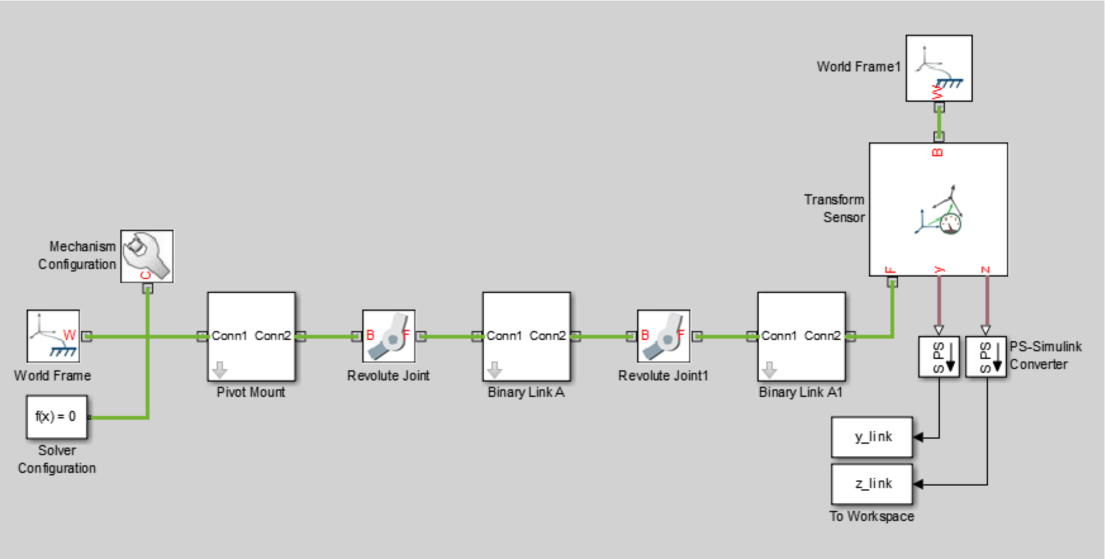
| Library | Block | Quantity |
|---|---|---|
| **SimMechanics > Second Generation > Frames and Transforms** | `Transform Sensor` | 1 |
| **SimMechanics > Second Generation > Frames and Transforms** | `World Frame` | 1 |
| **Simscape > Utilities** | `PS-Simulink Converter` | 2 |
| **Simulink > Sinks** | `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.

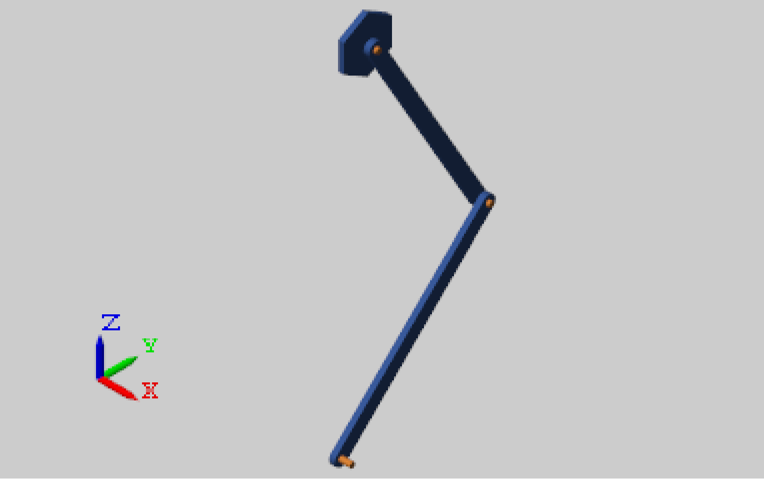
1 In the Revolute Joint block dialog box, select **State Targets** > **Specify Position Target**.

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.

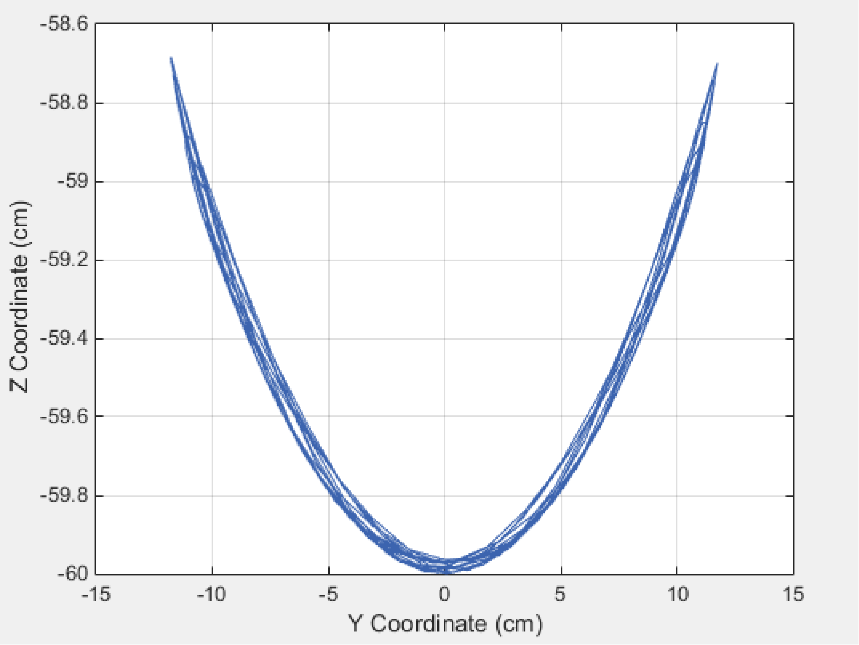


You can now plot the position coordinates of the lower link. To do this, at the MATLAB command line, enter:

```
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

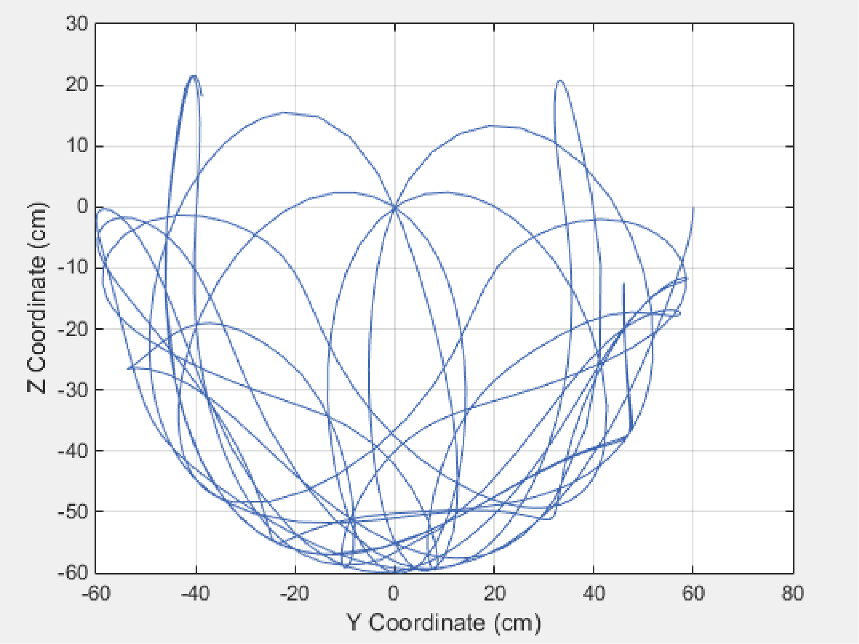
By adjusting the revolute joint state targets, you can simulate the model under chaotic conditions. One way to obtain chaotic motion is to rotate the top revolute joint by a large angle. To do this, in the Revolute Joint dialog box, change **State Targets** > **Position** > **Value** to 90 and click **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 4-108.

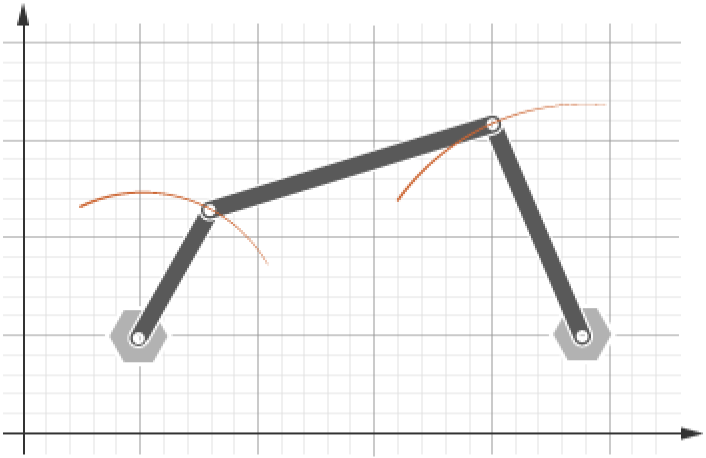
# Actuate Joint in Four-Bar Model

**In this section...**

## Model Overview

In SimMechanics, 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 Four Bar” on page 3-19, 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:

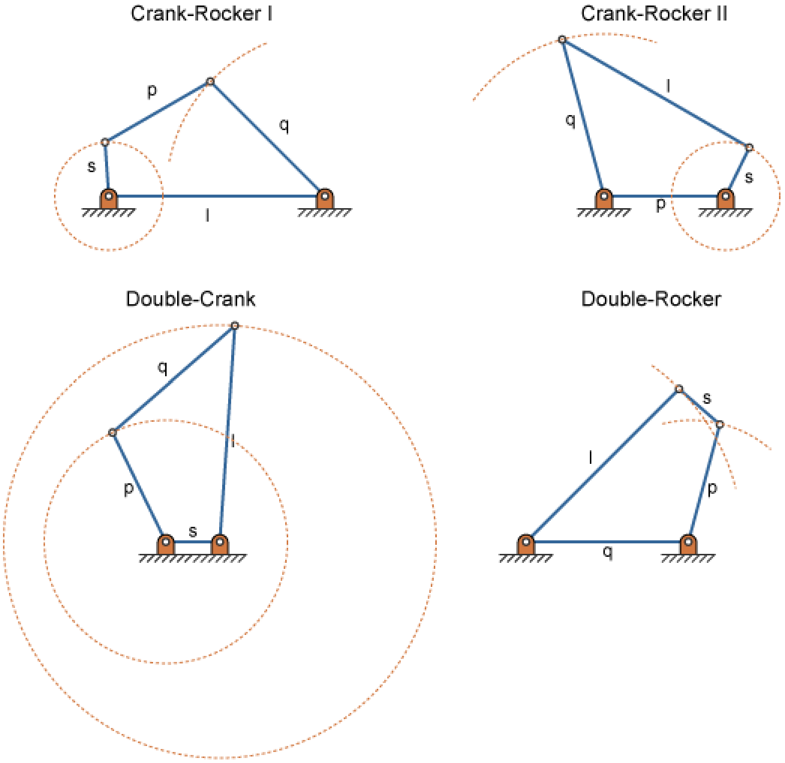
- s is the shortest link
- l is the longest link
- p and q are the two remaining links

### 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.



Crank-Rocker I

Crank-Rocker II

Double-Crank

Double-Rocker

## 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 SimMechanics can use. For more information, see “Actuating and Sensing Using Physical Signals” on page 4-29.

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

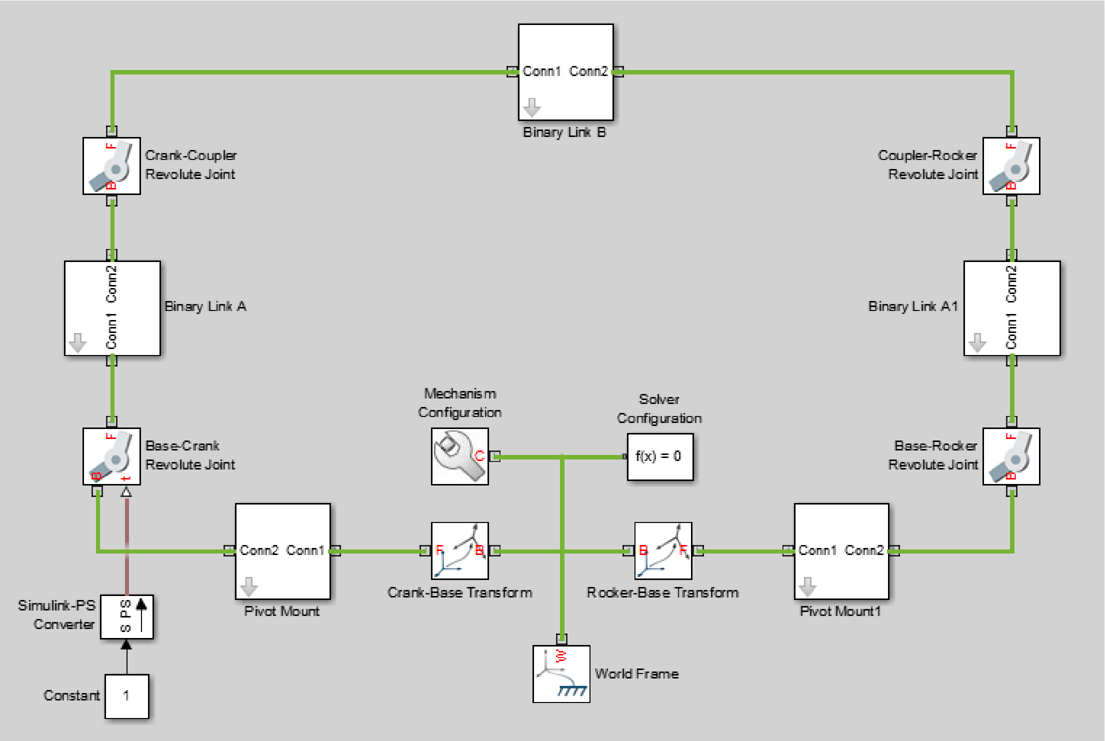
Provide the joint actuation input, specify the joint internal mechanics, and sense the position coordinates of the coupler link end frames.

### 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 Four Bar” on page 3-19.
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**, 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.

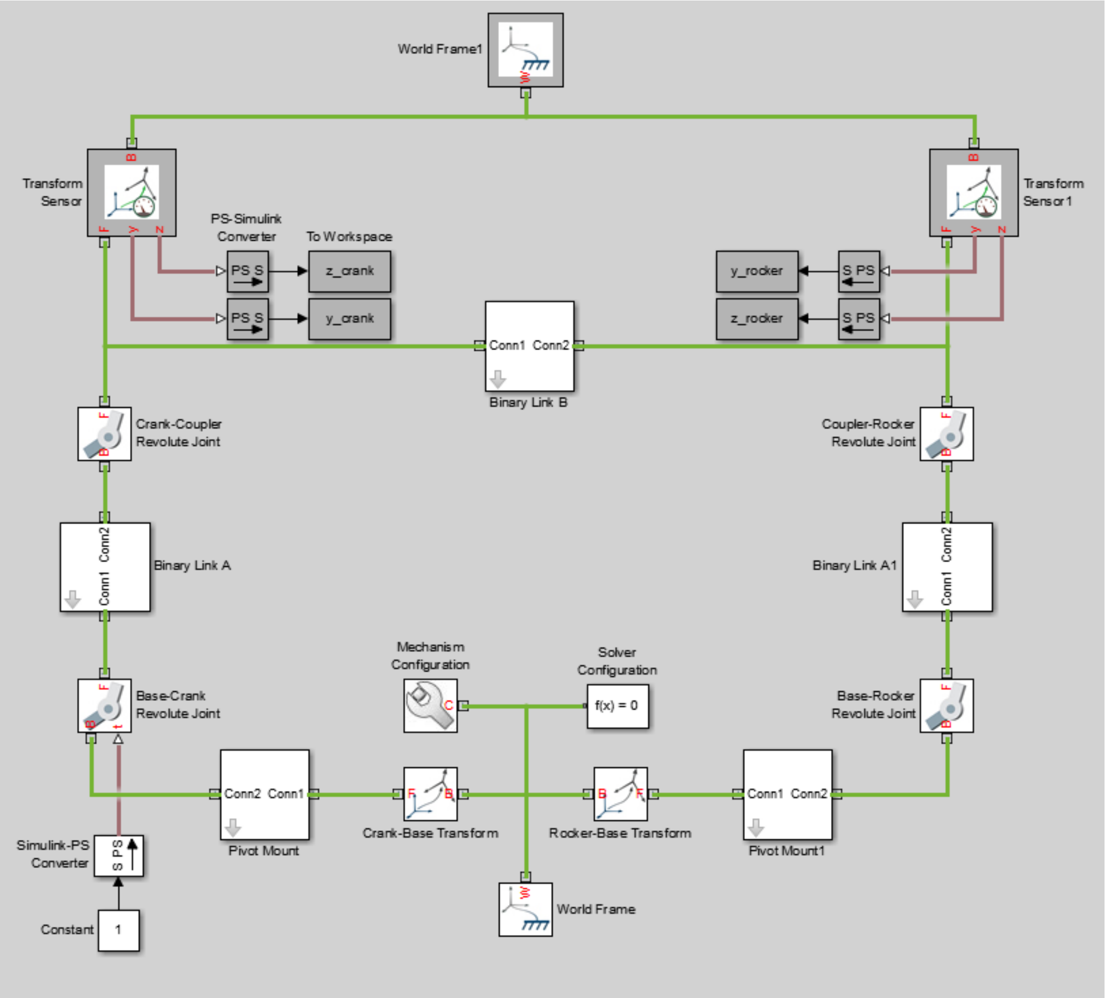
| Library | Block | Quantity |
|---|---|---|
| **SimMechanics > Frames and Transforms** | `Transform Sensor` | 2 |

| Library | Block | Quantity |
|---|---|---|
| **SimMechanics > 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 **Input 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:

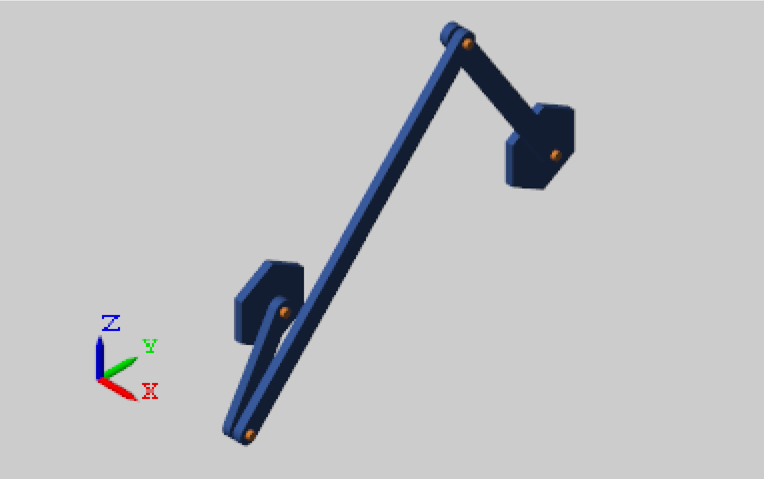
   - `y_crank`
   - `z_crank`
   - `y_rocker`
   - `z_rocker`

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

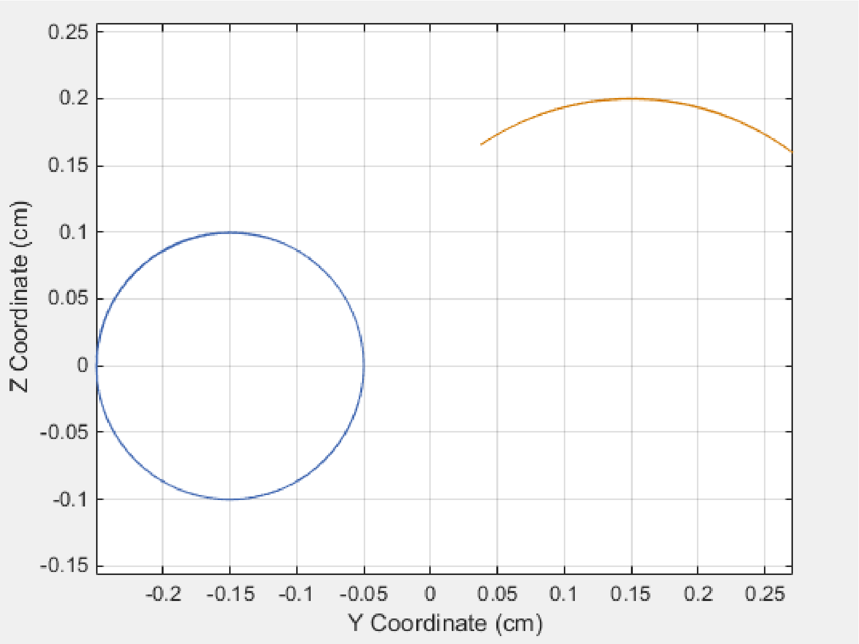
Run the simulation. You can do this in the Simulink tool bar by clicking the run button. Mechanics Explorer plays a physics-based animation of the four bar assembly.



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.

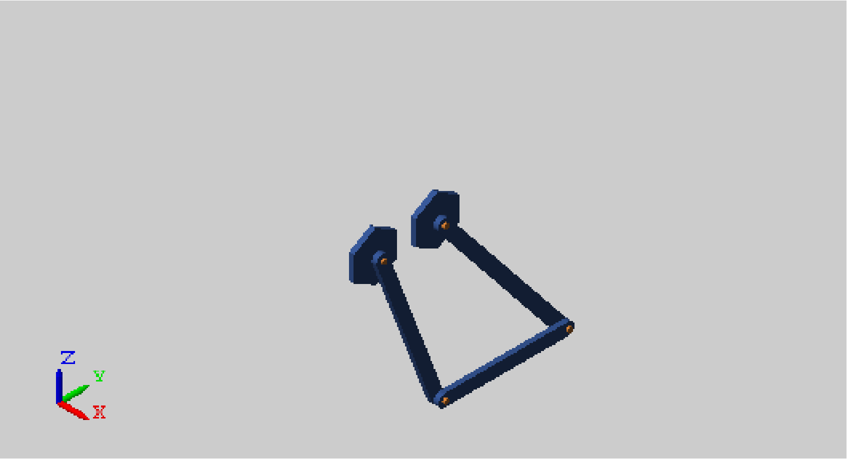


### Simulate Model in Double-Crank Mode

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 |

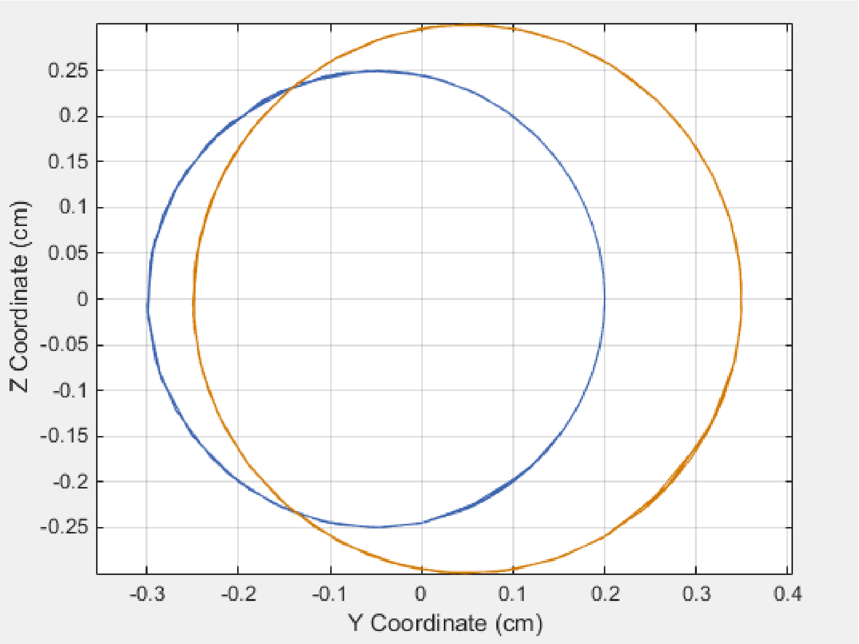
Update and simulate the model. The figure shows the updated visualization display in Mechanics Explorer.



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.

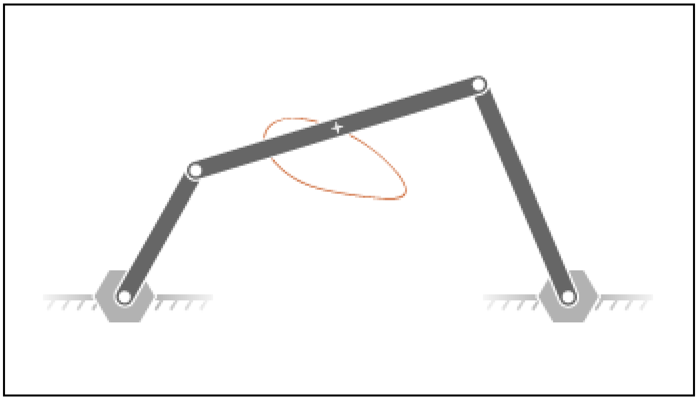

0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
-0.25
Z Coordinate (cm)
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
Y Coordinate (cm)

# Analyze Coupler Curves at Various Coupler Lengths

**In this section...**

## Model Overview

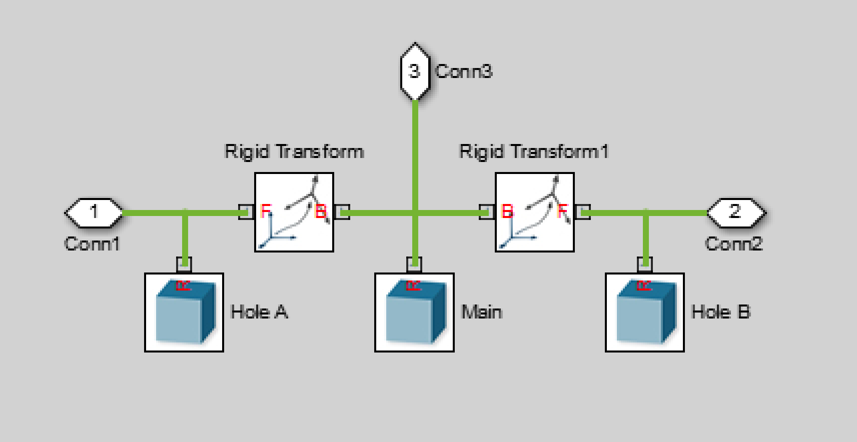
In this tutorial, you create a simple MATLAB script to simulate a four-bar model at various coupler lengths. The script uses the coupler motion coordinates, obtained using a Transform Sensor block, to plot the resulting coupler curve at each value of the coupler length. For information on how to create the four-bar model used in this tutorial, see "Model Four Bar" on page 3-19.



## 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 Four Bar" on page 3-19.

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.

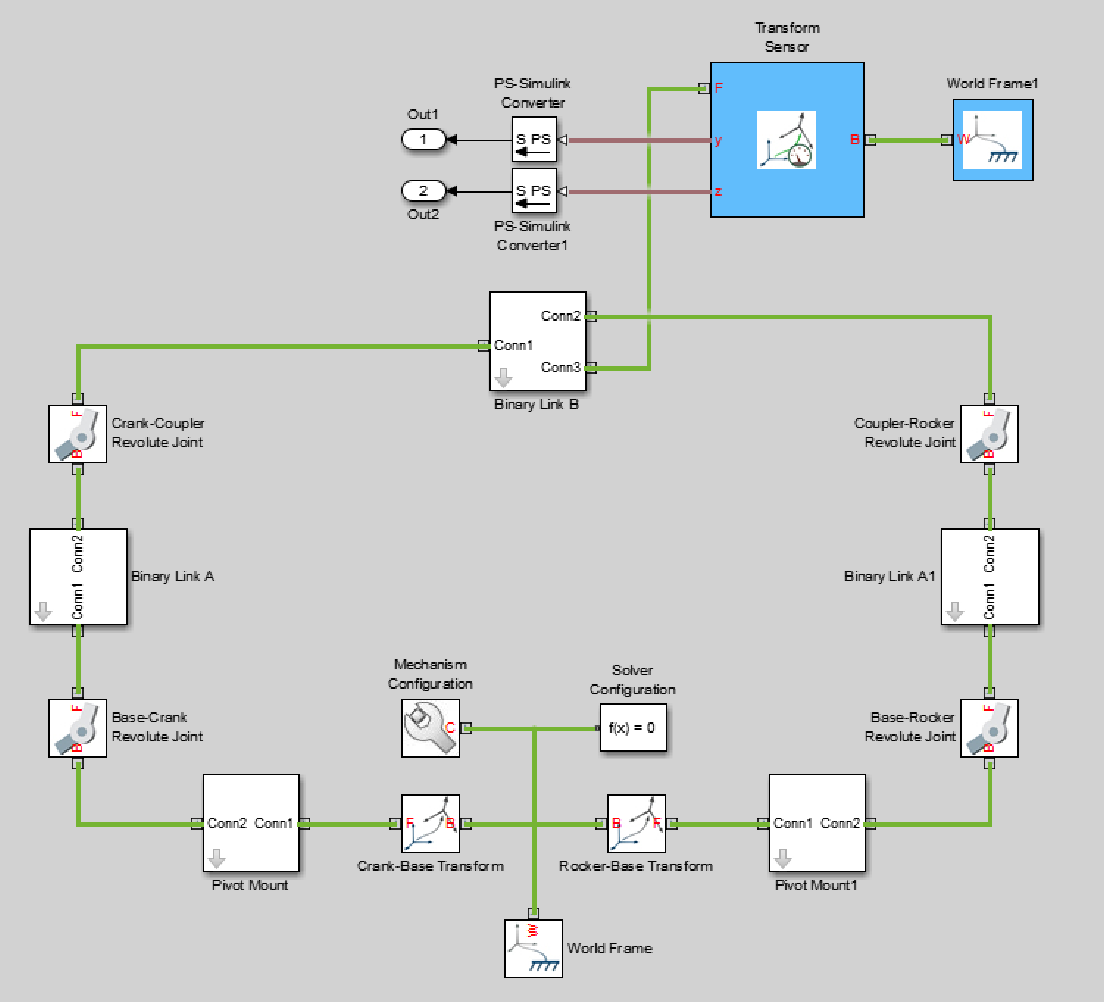
| Library | Block | Quantity |
|---|---|---|
| Frames and Transforms | `World Frame` | 1 |
| Frames and Transforms | `Transform Sensor` | 1 |
| Simscape Utilities | `PS-Simulink Converter` | 2 |
| Simulink Sinks | `Outport` | 2 |

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:

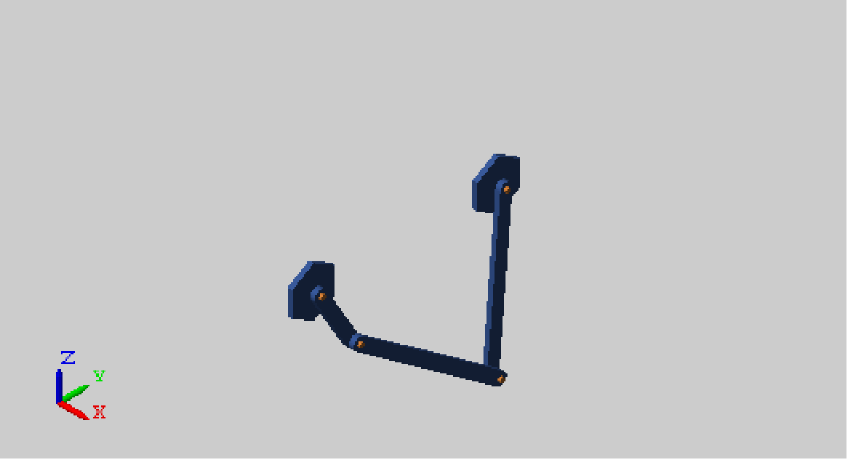
```
% Run simulation nine times, each time
% increasing coupler length by 1 cm.
% The original coupler length is 20 cm.
for i = (0:8);
    LCoupler = 20+i;

    % 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.

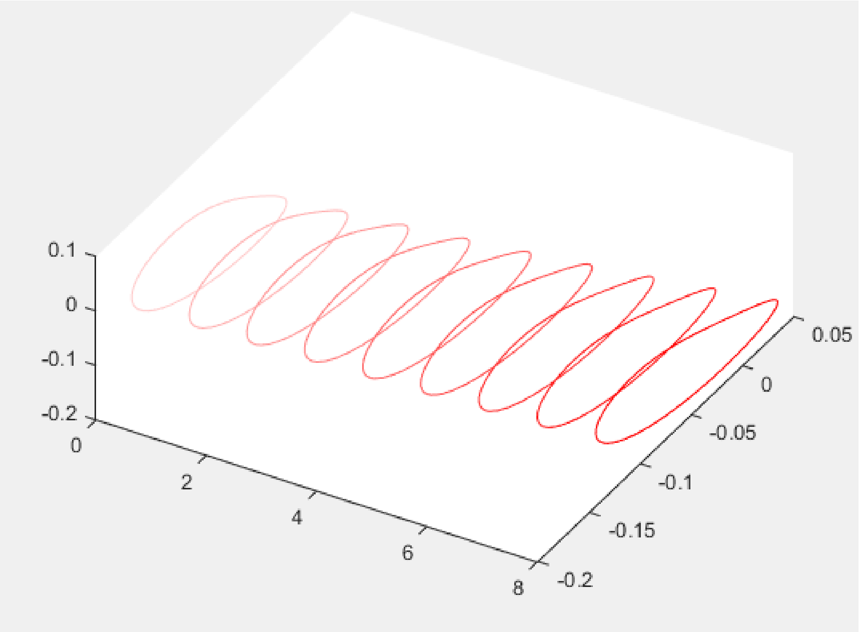
3 Save the script as sim_four_bar in the folder containing the four-bar model.

## 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.



SimMechanics 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 at Joints

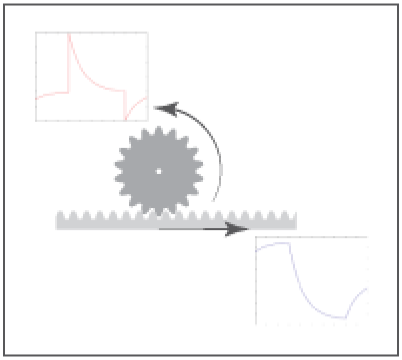
**In this section...**

## Overview

SimMechanics 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 attributable to:

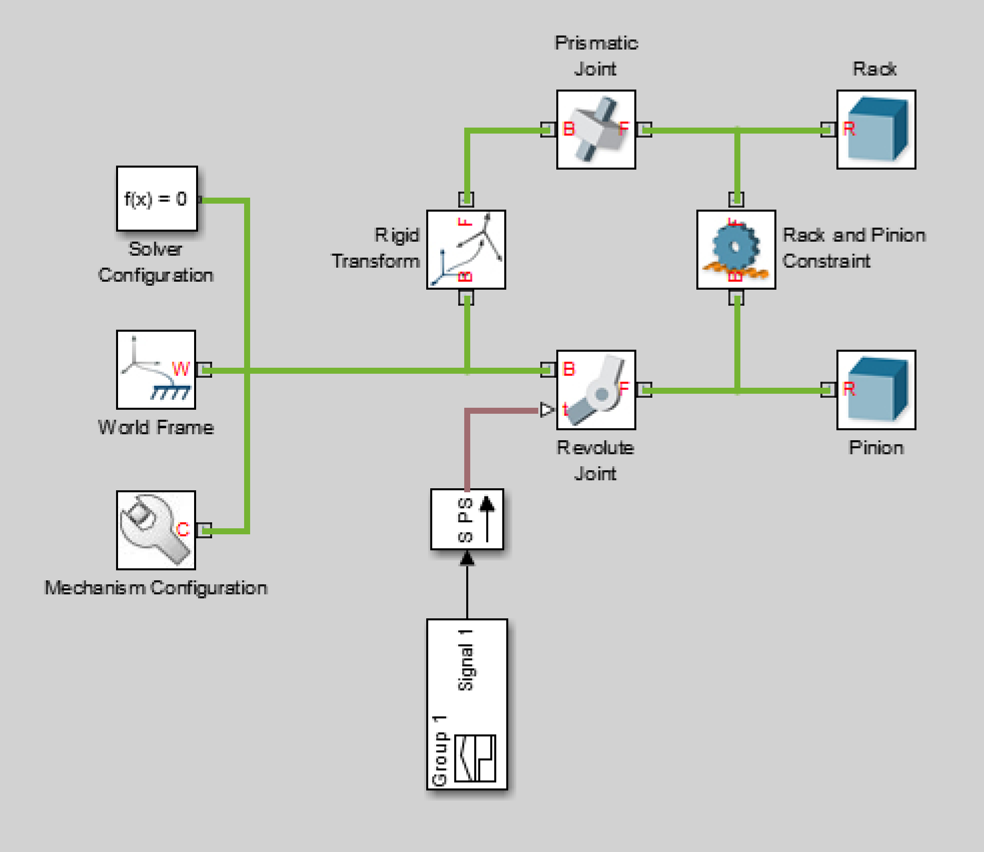
- Joint actuation inputs
- Joint constraints
- Joint actuation inputs, constraints, and internal mechanics combined

In this tutorial, you explore the different types of force and torque sensing that SimMechanics joint blocks provide.



## Open Model

At the MATLAB command prompt, enter `smdoc_rack_pinion_c`. SimMechanics opens a rack and pinion model that you can use to explore the force and torque sensing capability of joint blocks.

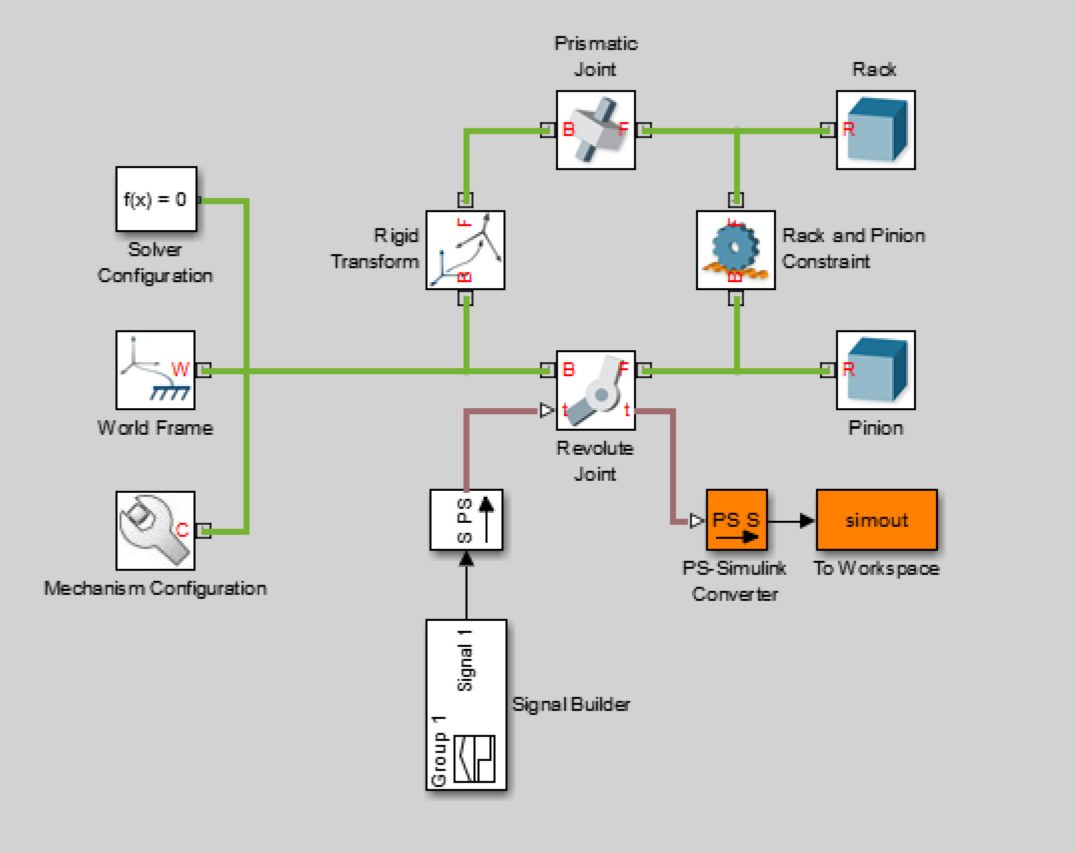


## Sense Actuation Torque

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.

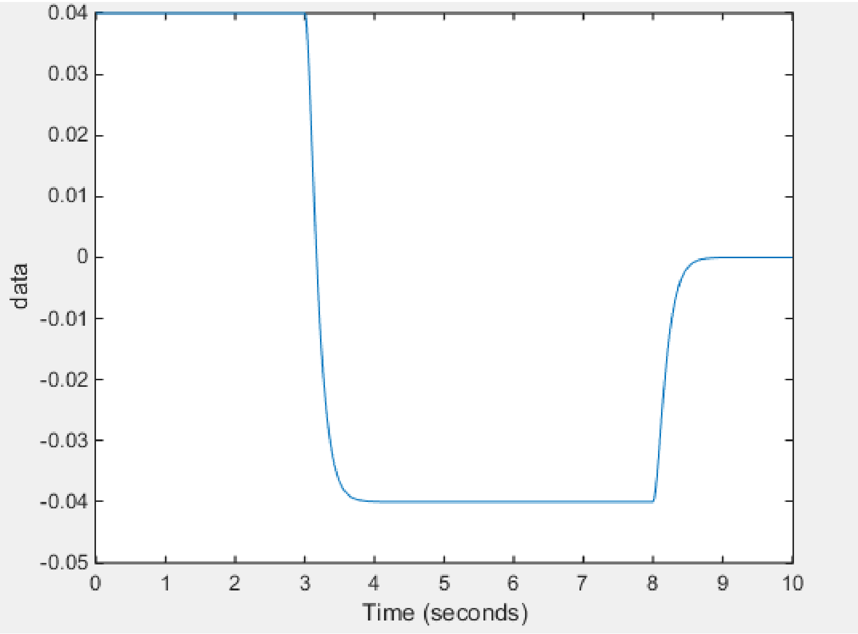


4 Simulate the model, e.g., by pressing **Ctrl+D**. The To Workspace block outputs the actuator torque signal into a time-series variable, simout, available in the MATLAB base workspace.

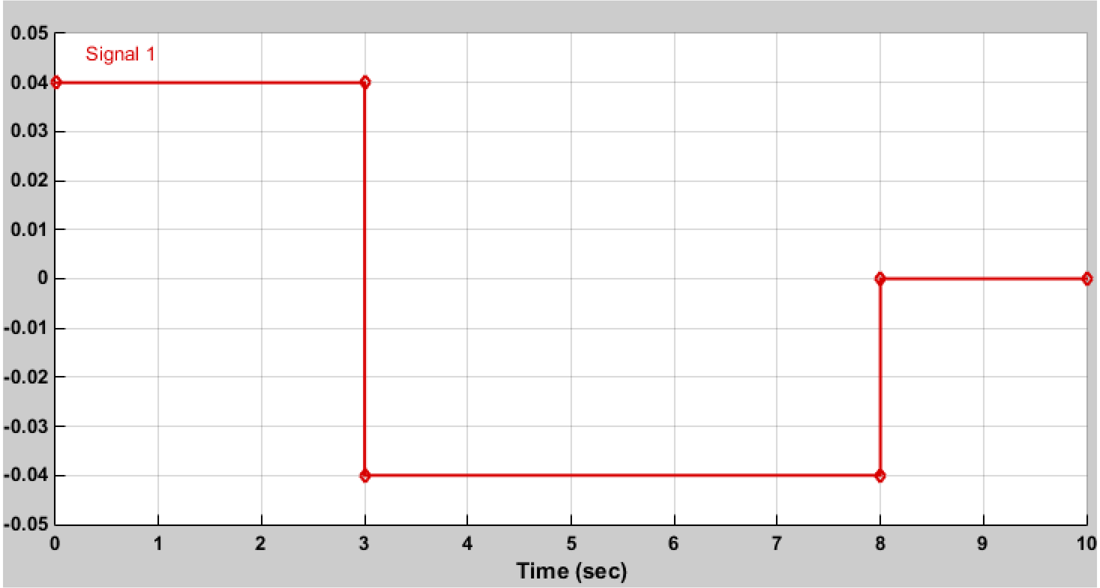
5 At the MATLAB command prompt, enter:

```
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.

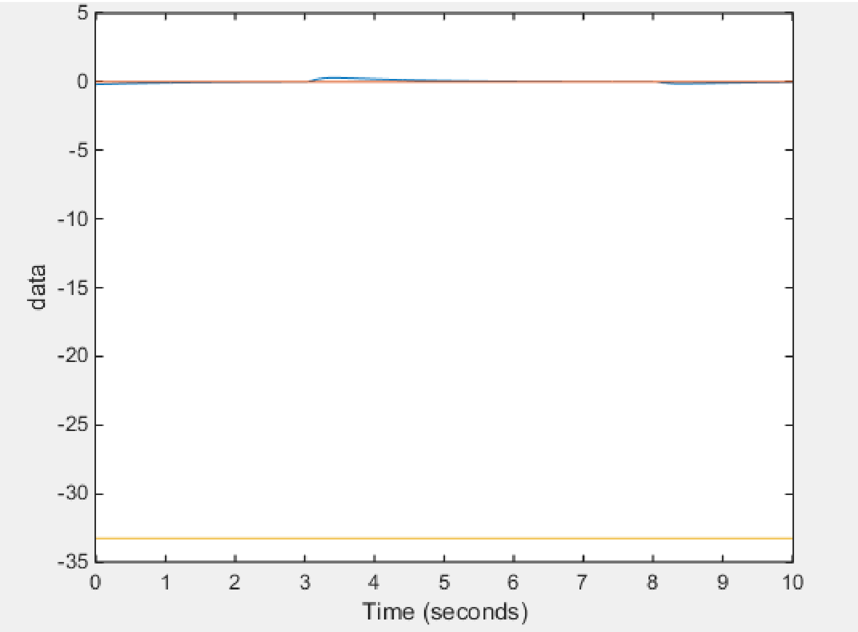
3 Deselect **Z Revolute Primitive (Rz)** > **Sensing** > **Actuator Torque**.

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 rigid 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 Internal Forces in Double-Pendulum Link” on page 4-97.

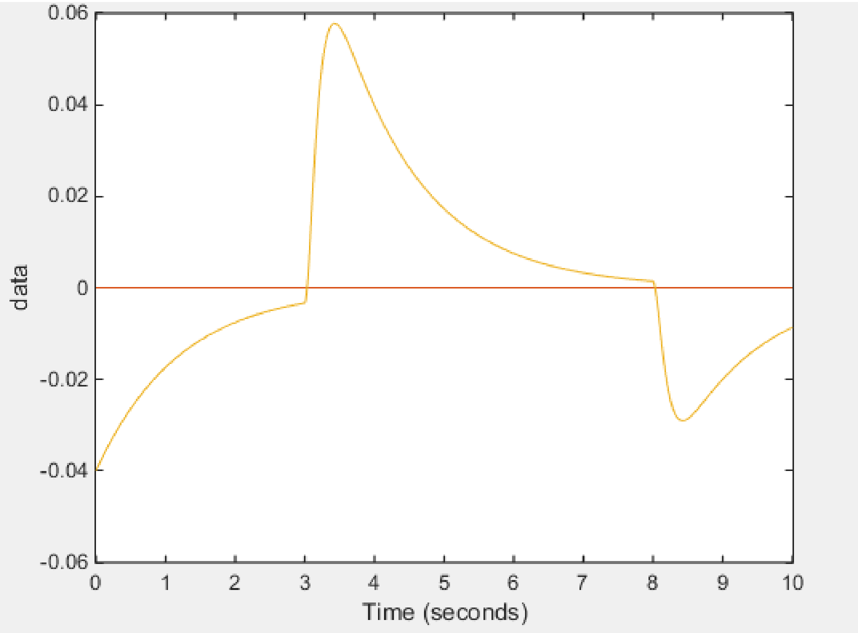
## Sense Total Forces

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.

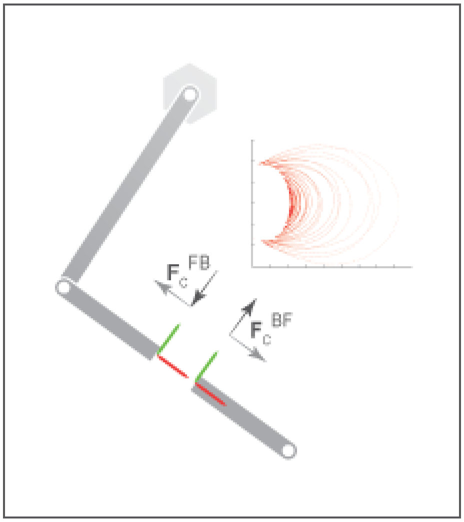
# Sense Internal Forces in Double-Pendulum Link

**In this section...**

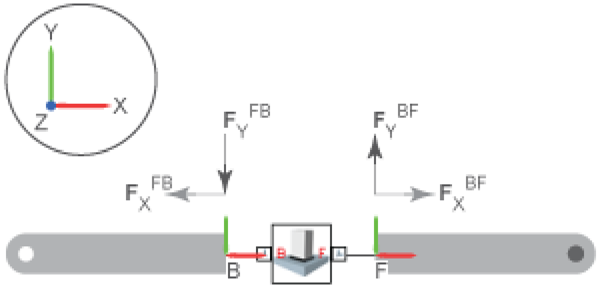
## Model Overview

SimMechanics 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 rigid 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 Double Pendulum” on page 3-14.



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.



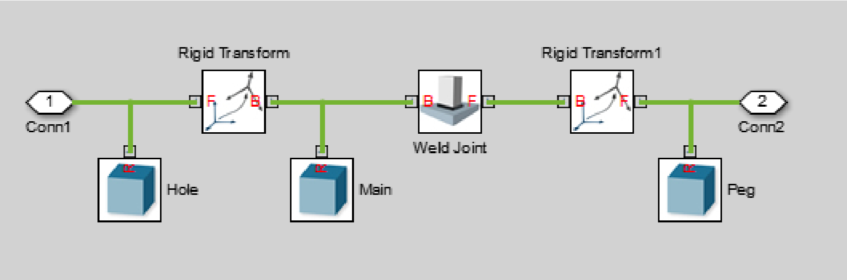
The Weld Joint block enables you to sense the constraint force that the follower frame exerts on the base frame or, alternatively, the constraint force that the base frame exerts on the follower frame. The two forces have the same magnitude but, as shown in the binary link schematic, opposite directions. 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

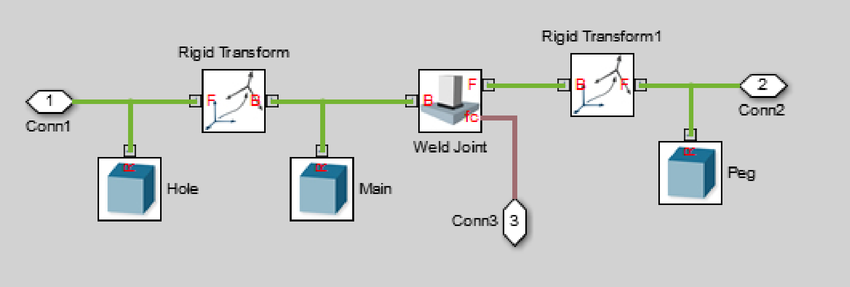
1. At the MATLAB command prompt, enter `smdoc_double_pendulum`. A double-pendulum model opens up.
2. Click the Look Inside Mask arrow in the Binary Link A1 subsystem block.
3. From the **SimMechanics** > **Second Generation** > **Joints** library, drag a `Weld Joint` block.
4. Connect the Weld Joint block as shown in the figure. This block enables you to sense the constraint forces that hold the rigid 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

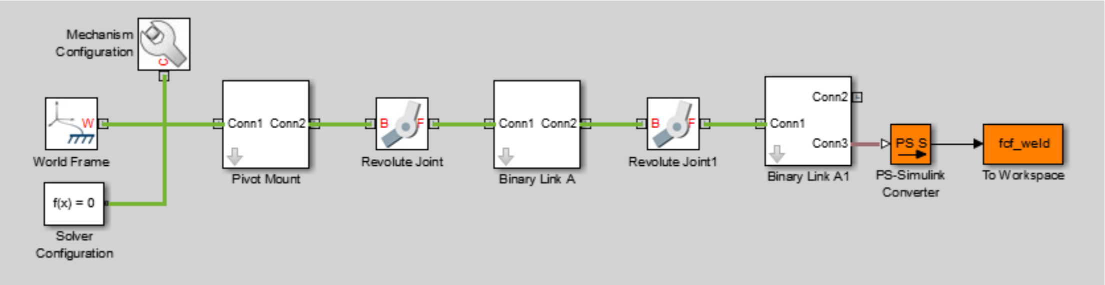
1. In the Weld Joint block dialog box, select **Constraint Force**. The block exposes a physical signal output port labeled fc.
2. Add a Simscape Output port to the subsystem block diagram. Connect the block as shown in the figure and exit the subsystem view.



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.

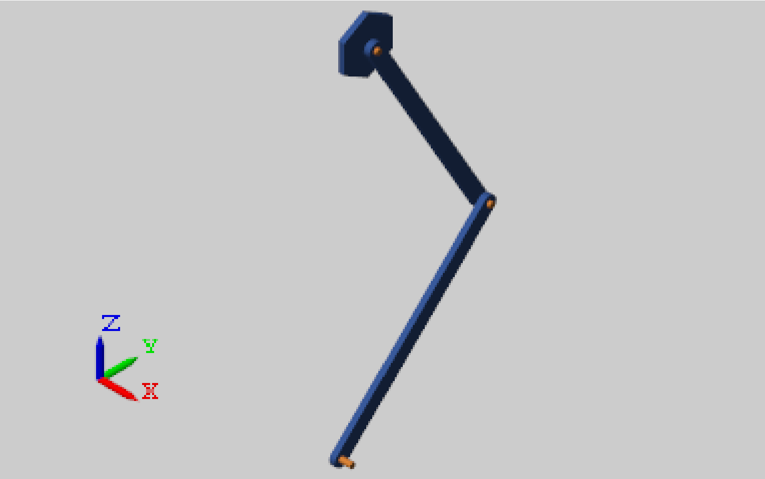
## Simulate Model

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.

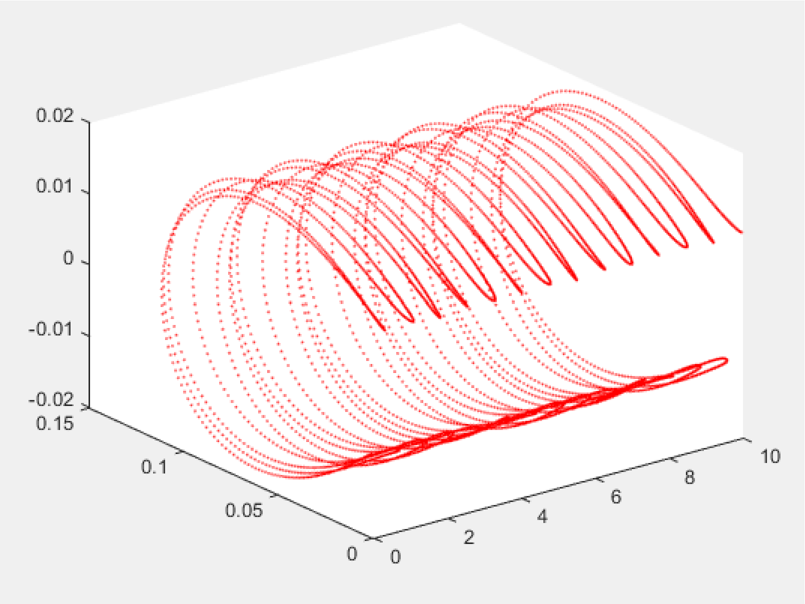


## Plot Constraint Forces

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.

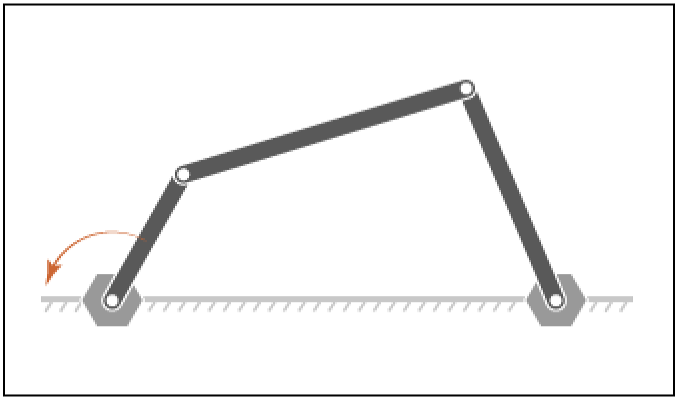

0.02
0.01
0
-0.01
-0.02
0.15
0.1
0.05
0
0
2
4
6
8
10

# Prescribe Joint Motion in Four-Bar Model

| In this section... |
|---|
| |
| |
| |

## 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 Four Bar" on page 3-19.

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 |

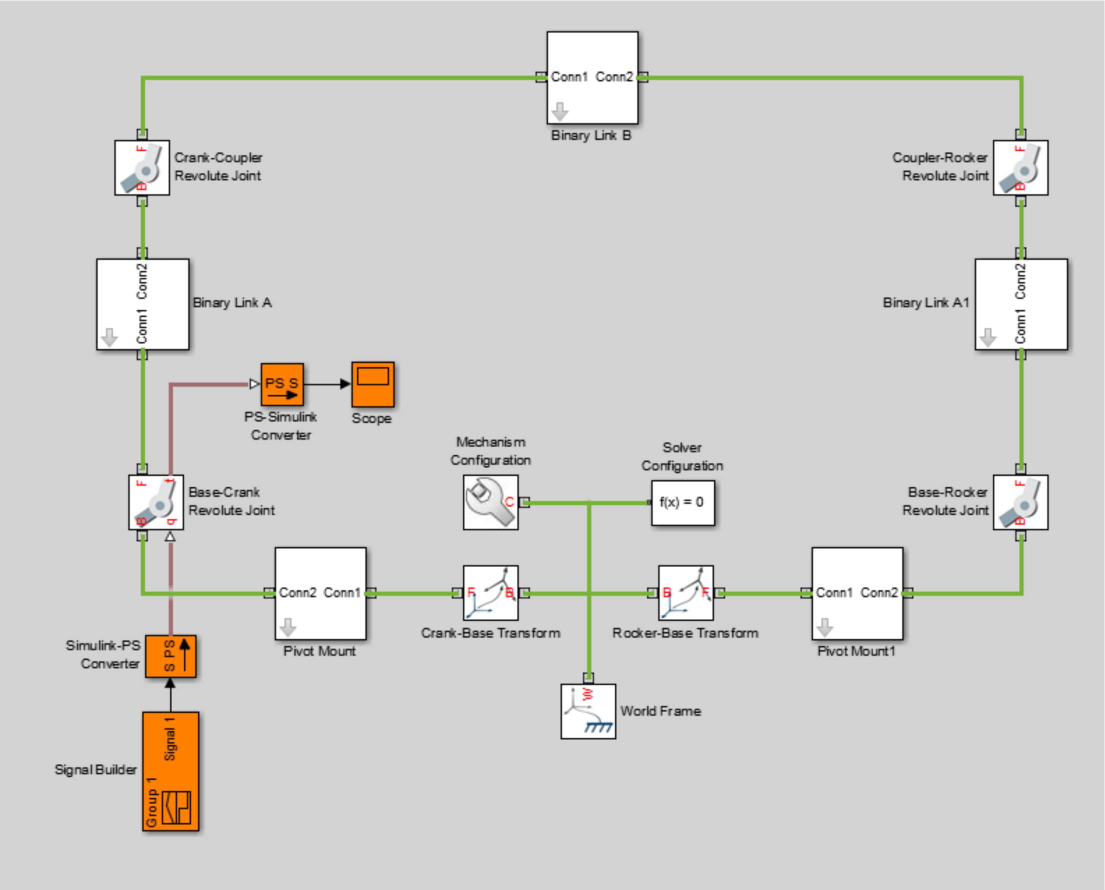
The joint block displays two physical signal ports. Input port q accepts the joint angular position. Output port t provides the joint actuation torque required to achieve that angular position.

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** |

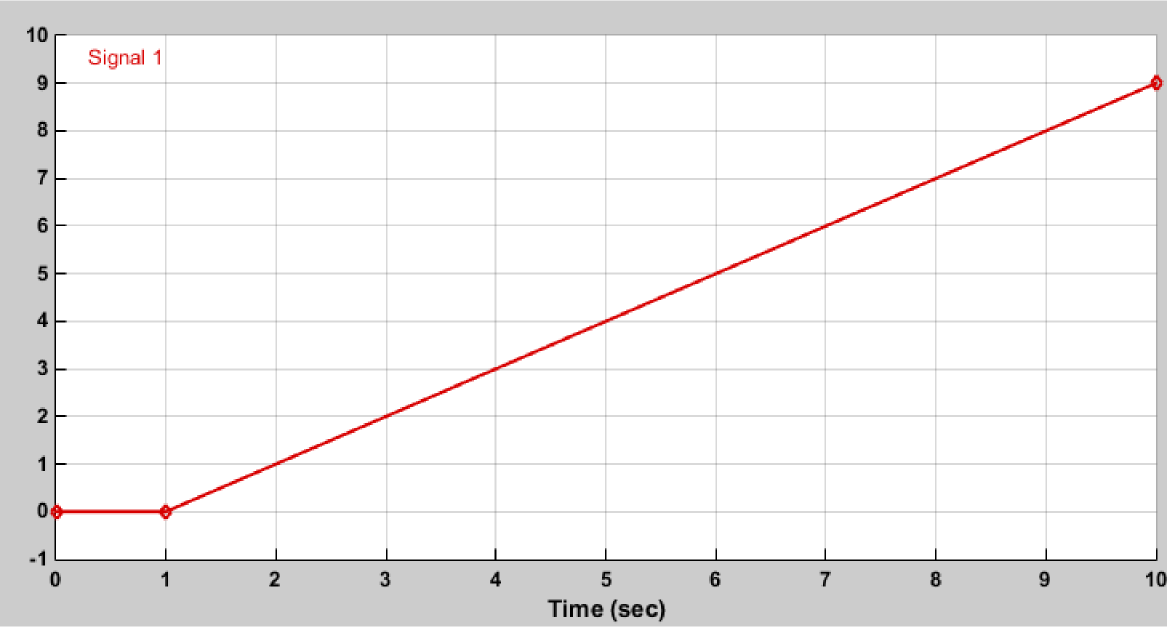
5 Connect the blocks as shown in the figure.



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` |

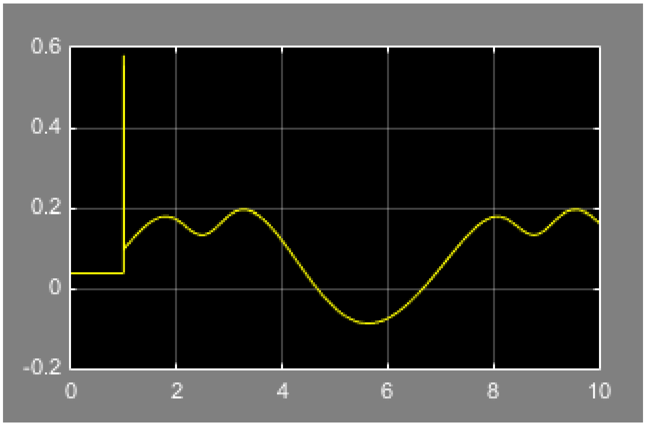
7 In the Signal Builder window, specify the joint angular trajectory as shown in the figure.



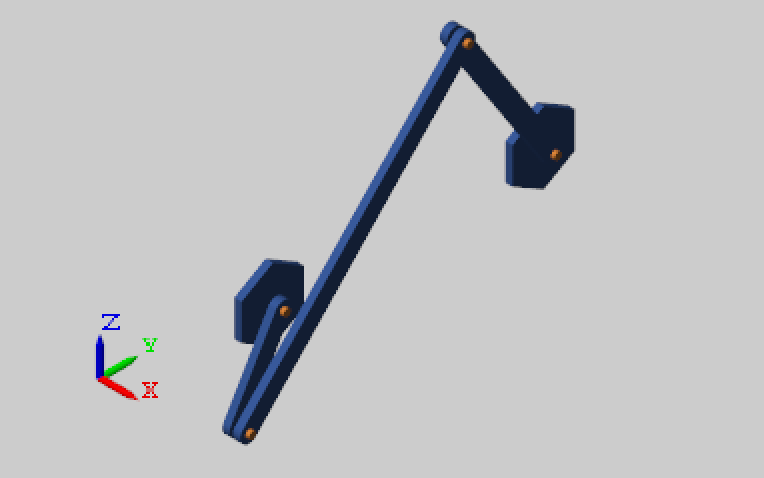
This signal corresponds to a constant angular speed of 1 rad/s from t = 1s onwards.

## 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.



## Related Examples

- “Sense Motion in Double-Pendulum Model” on page 4-65
- “Prescribe Joint Motion in Planar Manipulator Model” on page 4-108
- “Specify Motion Input Derivatives” on page 4-26

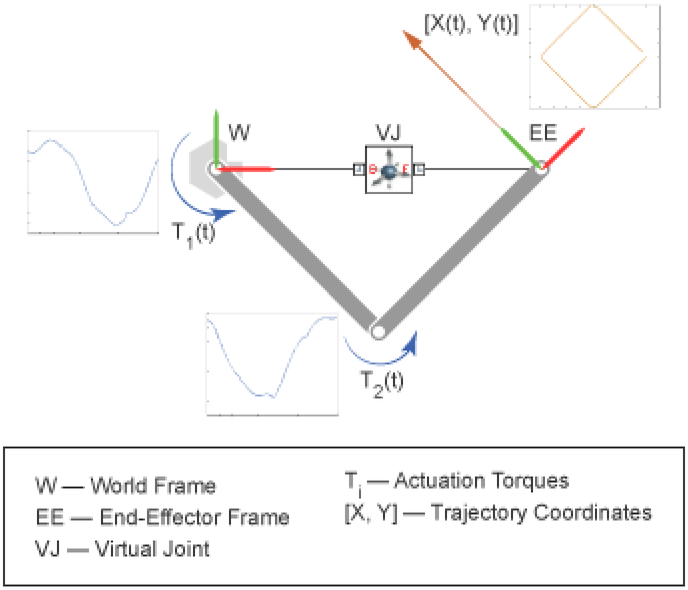
# Prescribe Joint Motion in Planar Manipulator Model

**In this section...**

## 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.

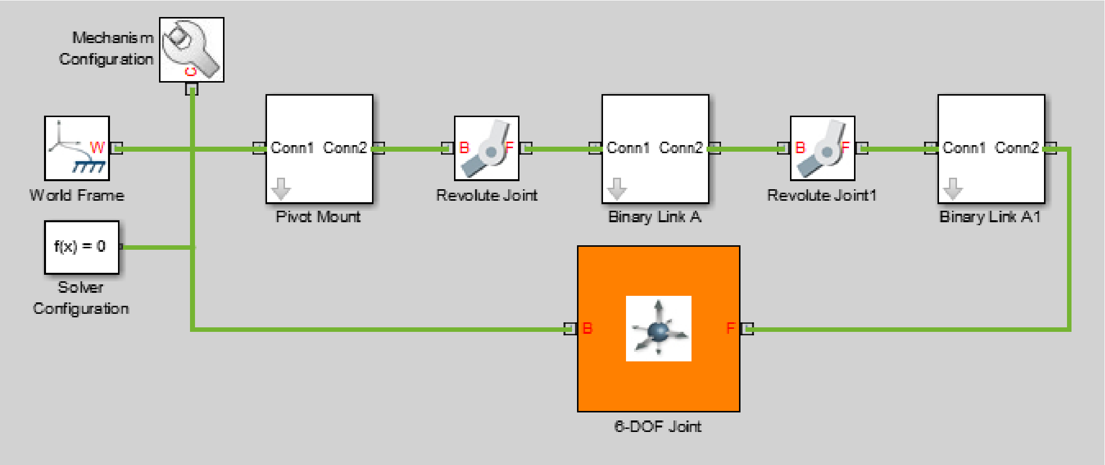
The time-varying coordinates trace a square pattern, achieved by automatically computing and applying actuation torques at the various manipulator joints. During simulation, you can output the automatically computed torques and plot them using Simulink blocks or MATLAB commands, e.g. for analysis purposes.



## 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 Double Pendulum” on page 3-14

2 From the **SimMechanics** > **Second Generation** > **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` |

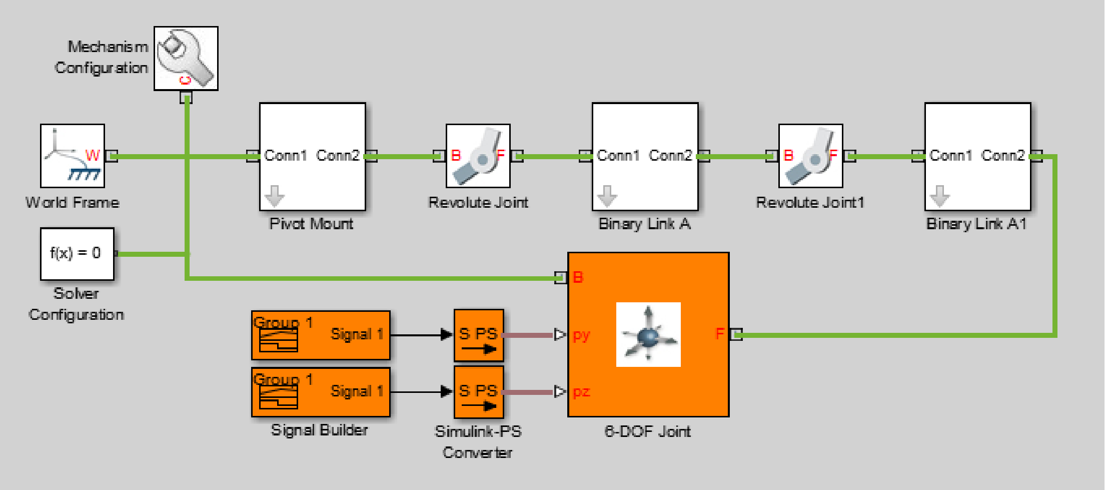
The block exposes two physical signal ports through which you can provide the joint motion inputs.

2 Drag these blocks into the model.

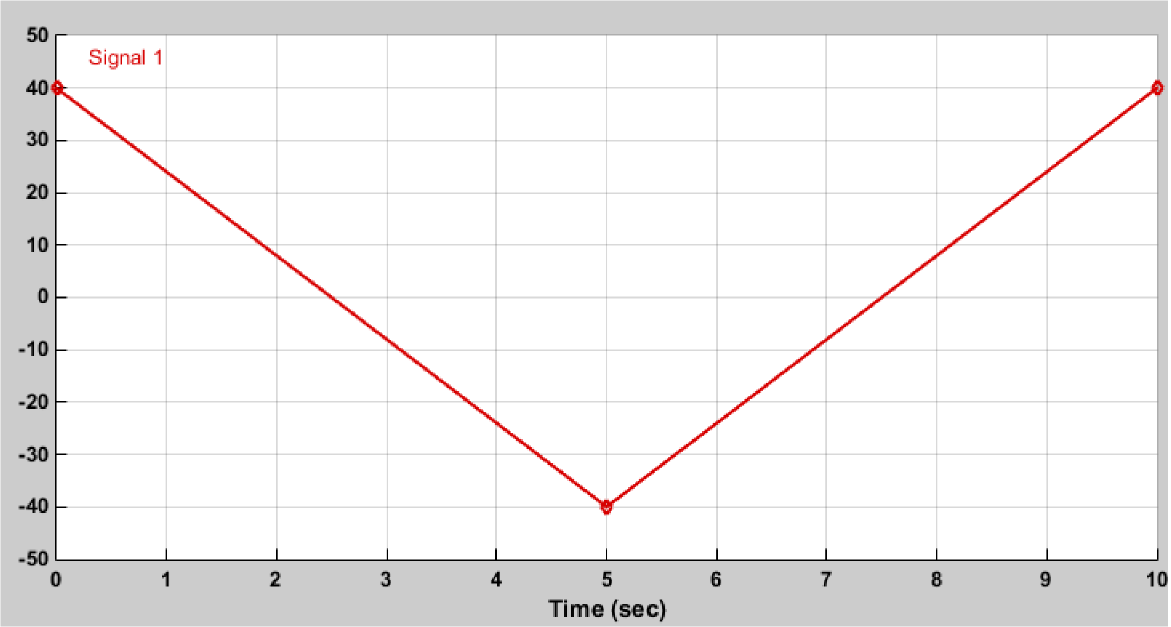
| Library | Block | Quantity |
|---|---|---|
| **Simscape > Utilities** | `Simulink-PS Converter` | 2 |
| **Simulink > Sources** | `Signal Builder` | 2 |

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 SimMechanics blocks.

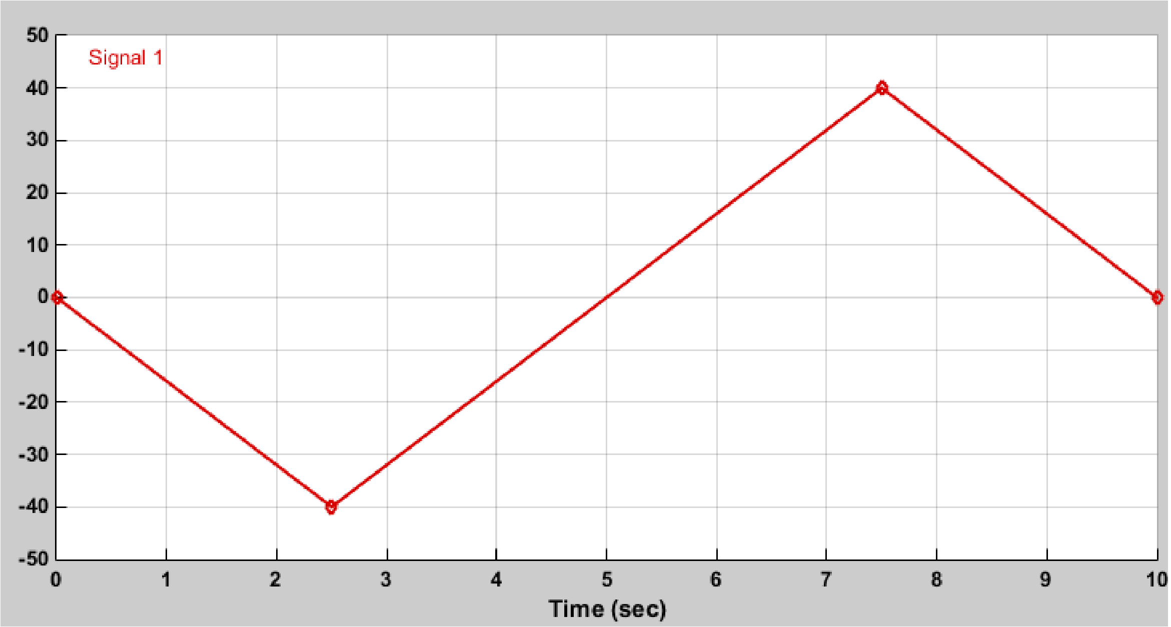
**3** Connect the blocks as shown in the figure.



**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. SimMechanics 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 SimMechanics 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 |

SimMechanics 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.

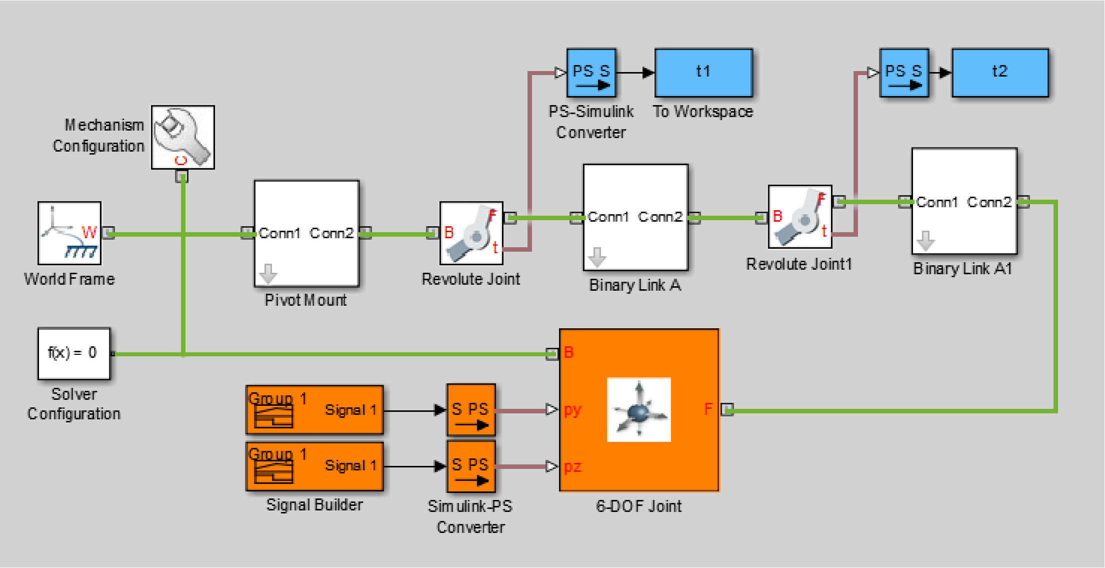
2 Drag these blocks into the model.

| 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`.

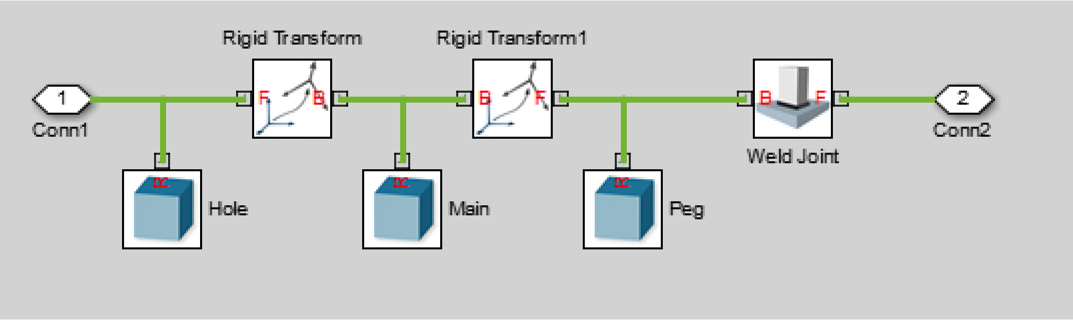
4 Connect the blocks as shown in the figure.



## Simulate Model

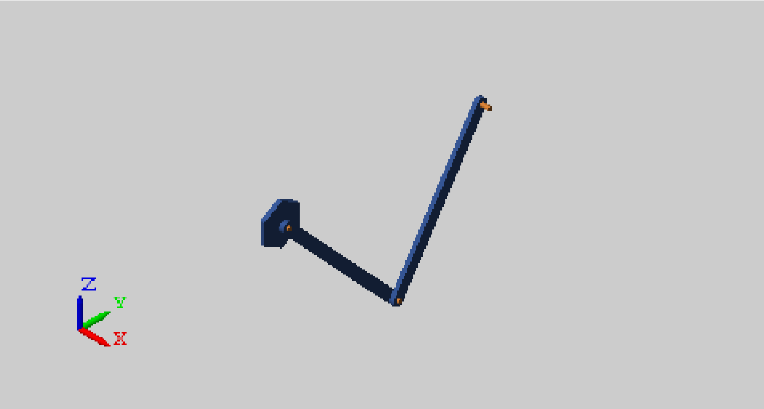
Attempt to run the simulation. You can do this in the Simulink Editor menu bar, by selecting **Simulation** > **Run**. Simulation fails with an error arising from the closed kinematic loop present in the model. SimMechanics requires this loop to contain at least one joint block without motion inputs or automatically computed actuation forces or torques.

1 From the **SimMechanics** > **Second Generation** > **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.

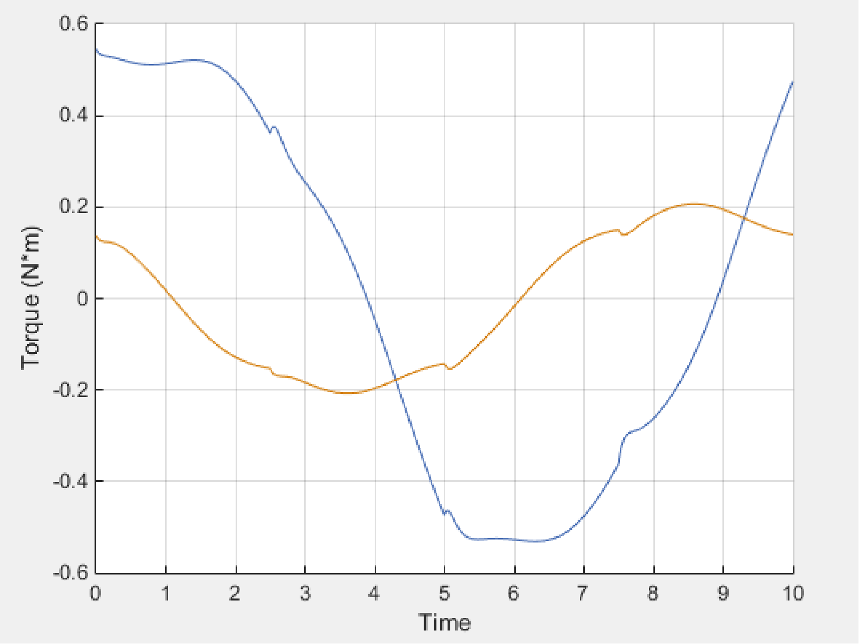


Plot the computed actuation torques acting at the two revolute joints in the linkage. At the MATLAB command line, enter this code:

```
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.



## Related Examples

- “Sense Motion in Double-Pendulum Model” on page 4-65
- “Prescribe Joint Motion in Four-Bar Model” on page 4-103
- “Specify Motion Input Derivatives” on page 4-26

## More About

- “Joint Actuation” on page 4-18
- “Actuating and Sensing Using Physical Signals” on page 4-29

# Simulation and Analysis

5

# Simulation

# Configure Model for Simulation

During simulation, SimMechanics employs a Simulink global solver to determine the configuration of a model as a function of time. You can select the best solver for your application from a list of solvers that Simulink provides. Simulation parameters include the numerical step used to progress through the simulation and the solver tolerance values. Adjust the parameters to optimize speed and accuracy of the simulation.

For solver selection and parameter specification, see:

- "Solvers" in the Simulink documentation.
- "Setting Up Solvers for Physical Models" in the Simscape documentation.

## Specify Solver Settings

To select a global solver for your model:

1. On the Simulink menu bar, click **Simulation** > **Model Configuration Parameters**.
2. On the Tree View pane, select **Solver**.
3. In **Solver Options**, click **Type** and select `Variable-step` or `Fixed-step`.

   **Note:** For best performance, select `Variable-step`. For model deployment, select `Fixed-step`.

4. Click **Solver** and select the appropriate solver for your application. The default solver is `ODE45 (Dormand-Prince)`.

To modify the global solver parameters for your model:

1. In the **Solver options** pane of the **Model Configuration Parameters** window, enter the desired values for step size and tolerance parameters.

Reducing the values of the step size and tolerance parameters enhances simulation accuracy, but decreases simulation speed. Adjust the parameters to obtain an optimal trade-off between simulation speed and accuracy.

## Related Examples

- "Configure Model for Rapid Accelerator Mode" on page 8-6

- “Find and Fix Simulation Issues” on page 5-4

# Find and Fix Simulation Issues

**In this section...**

Under certain conditions, a model that you simulate can behave in unexpected ways. Some issues that you can encounter while simulating a SimMechanics model include:

- Models with For Each Subsystem blocks have limited visualization
- Models with Model blocks have no visualization
- Simscape local solvers do not work for SimMechanics

## Models with For Each Subsystem Blocks Have Limited Visualization

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 which the For Each Subsystem block provides. The visualization limitation does not affect model simulation—SimMechanics simulates the model for all instances of the block.

## Models with Model Blocks Have No Visualization

Models with `Model` blocks (known as referenced models) simulate with no visualization. During model simulation, SimMechanics issues a warning at the MATLAB command line. The Mechanics Explorer visualization utility does not open.

## Simscape Local Solvers Do Not Work with SimMechanics

SimMechanics 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 SimMechanics portion of a model. SimMechanics blocks continue to use the Simulink global solver that you select in **Model Configuration Parameters** for your model.

**Note:** SimMechanics requires the Simulink global solver to be *continuous*. If the global solver is discrete, SimMechanics issues an error and the model does not simulate. This requirement applies to both fixed- and variable-step solvers.

## Related Examples

- "Configure Model for Simulation" on page 5-2
- "Configure Model for Rapid Accelerator Mode" on page 8-6

# Visualization and Animation

# Model Visualization

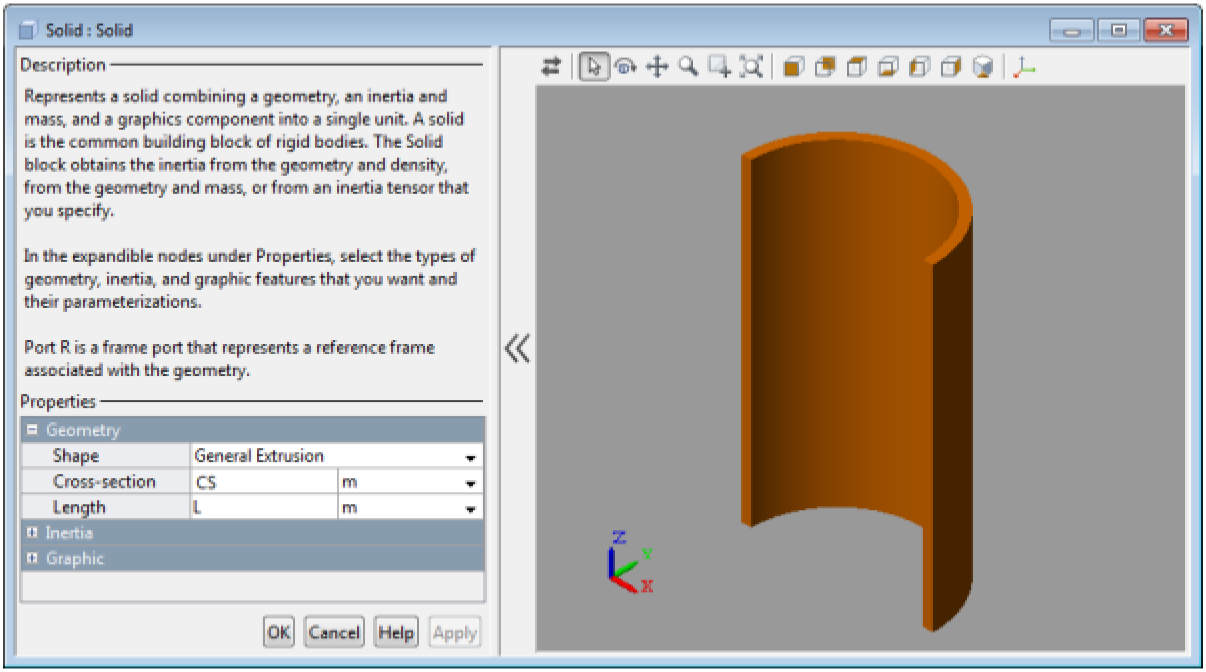
**In this section...**

## About Visualization

Visualization is the graphical rendering of bodies and multibody assemblies from a model. You can use it as a modeling aid, e.g., to visually check bodies during modeling and multibody connections during assembly or as a qualitative analysis tool, e.g., to analyze motion during simulation.

## Visualizing Individual Solids

To visualize a solid while modeling, you use the `Solid` block. This block provides a visualization pane that you can use to view the solid from different perspectives. You can select a standard viewpoint or rotate, pan, and zoom your solid. The figure shows the Solid block visualization pane.



Solid : Solid

Description

Represents a solid combining a geometry, an inertia and mass, and a graphics component into a single unit. A solid is the common building block of rigid bodies. The Solid block obtains the inertia from the geometry and density, from the geometry and mass, or from an inertia tensor that you specify.

In the expandible nodes under Properties, select the types of geometry, inertia, and graphic features that you want and their parameterizations.

Port R is a frame port that represents a reference frame associated with the geometry.

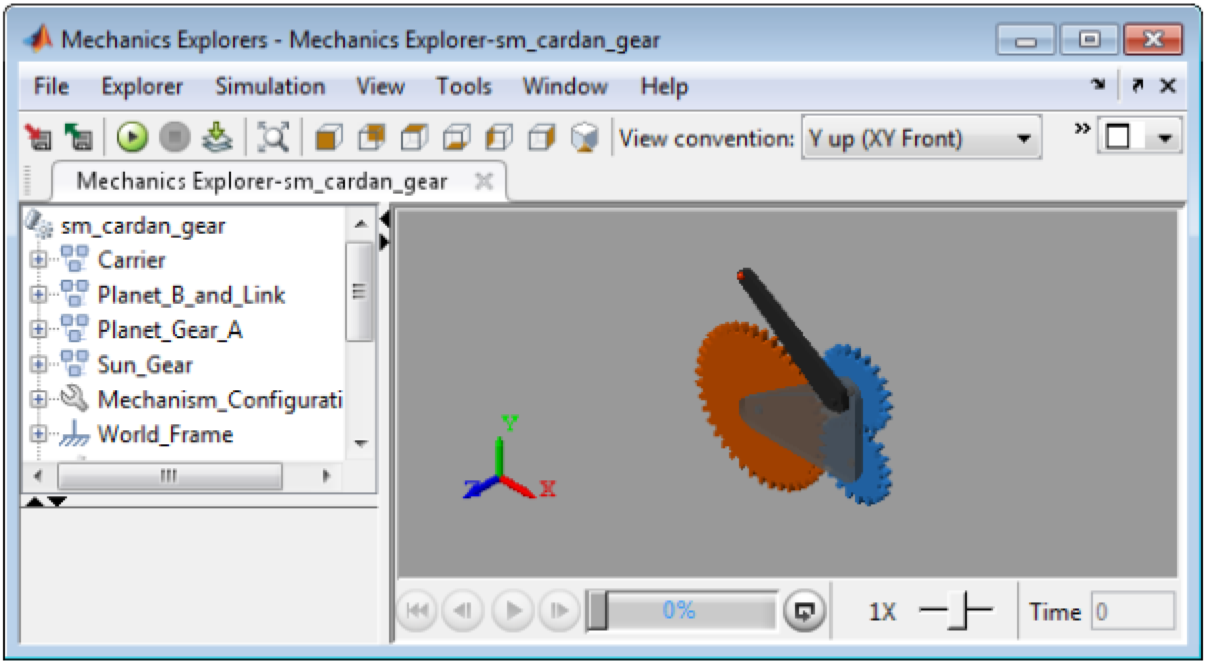
Properties

| | | |
|---|---|---|
| ⊟ Geometry | | |
| Shape | General Extrusion | |
| Cross-section | CS | m |
| Length | L | m |
| ⊞ Inertia | | |
| ⊞ Graphic | | |

You can view a solid before connecting the Solid block to a valid physical network. However, you must first specify the solid shape and color. If parameterizing these properties in terms of MATLAB variables, you must also initialize these variables—e.g., in the model workspace or in a subsystem mask.

## Visualizing Bodies and Assemblies

To visualize a compound rigid body—one containing several solids—or a multibody assembly, you use Mechanics Explorer. You can view a model from various perspectives by selecting a standard view or by rotating, panning, and zooming. The figure shows Mechanics Explorer.



Mechanics Explorer enables you to:

- View a model in its initial configuration. You must update the block diagram, e.g., by selecting **Simulation** > **Update diagram**.
- View a model animation during simulation. You must run simulation, e.g., by selecting **Simulation** > **Run**.

To visualize a model, the block diagram must contain a topologically valid physical network.

# Open Mechanics Explorer

Mechanics Explorer opens automatically when you update or simulate a model with SimMechanics blocks. For each model that you update or simulate, Mechanics Explorer opens a new tab with its own tree view, property view, and visualization panes. You can modify the model view in one tab without affecting the remaining open tabs.

If Mechanics Explorer does not automatically open on model update or simulation, you may have disabled model visualization. For more information, see “Turn Model Visualization Off and On” on page 6-35.

# Modify Model View

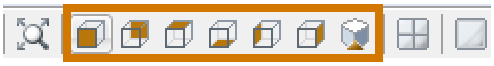
## Model Visualization

Multibody models lend themselves to 3-D visualization, a qualitative means of analysis that you can use to examine rigid body geometries, mechanical connections, and trajectories in three-dimensional space. In SimMechanics, 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.

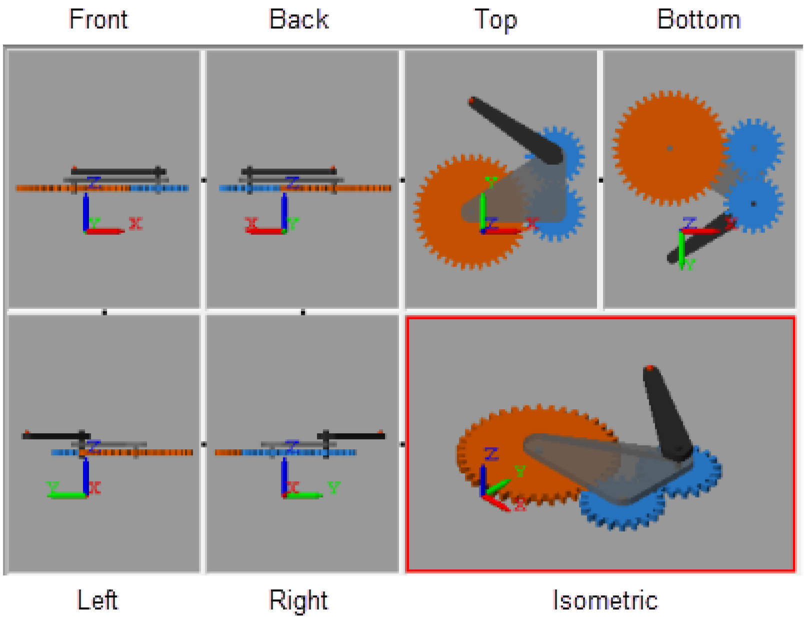
## Select a Standard View

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.

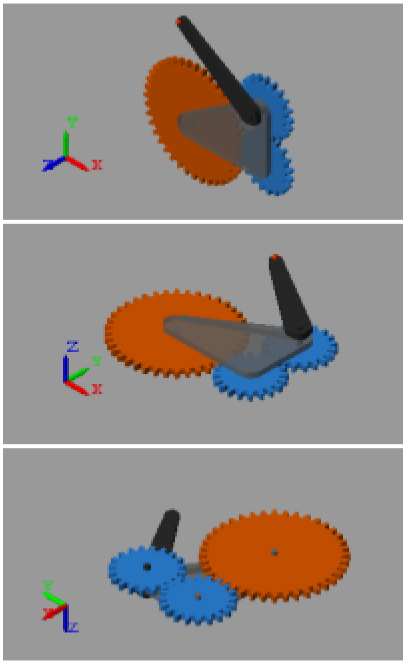


## Set View Convention

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:

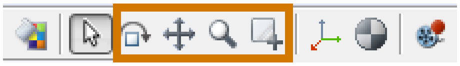
1 In the Mechanics Explorer tool strip, set **View convention** to one of the three options.

2 Select a standard view button.

The new view convention takes effect the moment you select a standard view.

## Rotate, Pan, and Zoom View

To view your model from an arbitrary point of view or at varying zoom levels, use the Rotate, Pan, and Zoom buttons. You can find these buttons in the Mechanics Explorer tool strip.

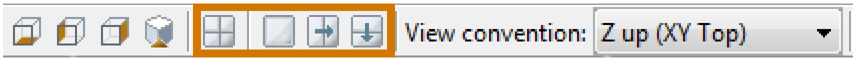


You can also use keyboard-and-mouse shortcuts. The table summarizes these shortcuts.

| Button | Shortcut |
|---|---|
| Rotate | **1** Click and hold the mouse scroll wheel.<br>**2** Move the mouse in the direction you want to rotate the model. |
| Pan | **1** Press and hold **Shift**.<br>**2** Click and hold the mouse scroll wheel.<br>**3** Move the mouse in the direction you want to pan the model. |
| Zoom | **1** Press and hold **Ctrl**.<br>**2** Click and hold the mouse scroll wheel.<br>**3** Move the mouse up to zoom in, down to zoom out. |

## Split Model View

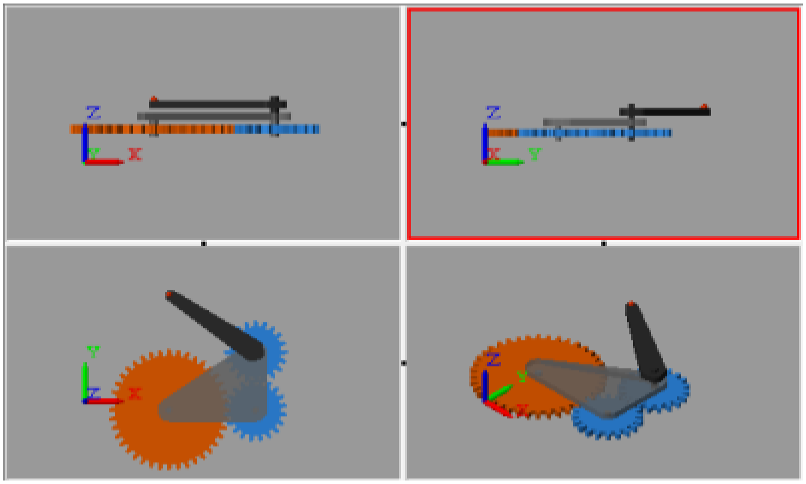
You can view your model from different perspectives, for example, to examine its motion in different planes. So that you can compare different model views, Mechanics Explorer enables you to split the visualization pane into tiles, each with its own view. To split the screen, you use the Mechanics Explorer toolstrip buttons shown in the figure.


View convention: Z up (XY Top)

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.

# Filtering Model Visualization

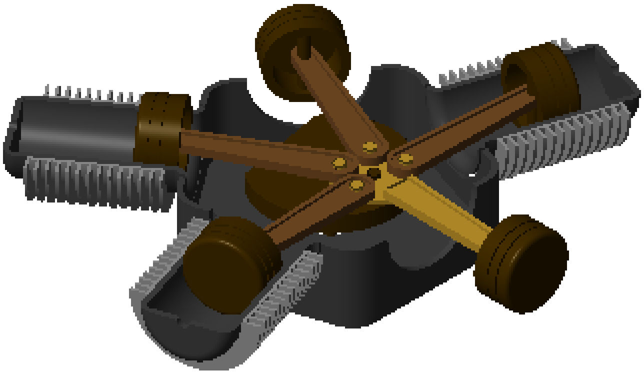
**In this section...**

## 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.

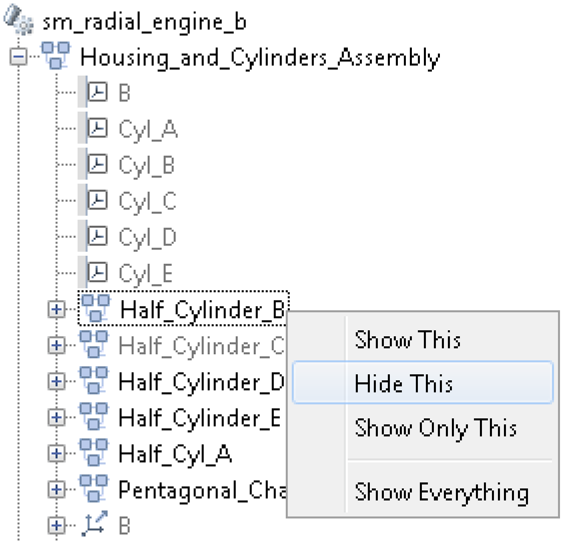
Visualization filtering is a Mechanics Explorer feature that lets you selectively show and hide parts of your model. By showing only those parts that you want to see, you can more easily discern any components placed within or behind other components—such as an engine piston traveling inside a cylinder 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 “Filter Radial Engine Visualization” on page 6-17.



## 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:

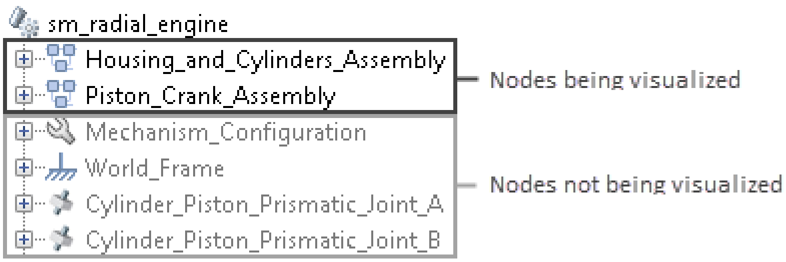
- `Show This` — Enable visualization for the selected component. This option has no effect if the component is already visible.
- `Hide This` — Disable visualization for the selected component. This option has no effect if the component is already hidden.
- `Show Only This` — Enable visualization for 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, rigid 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.


sm_radial_engine
Housing_and_Cylinders_Assembly
Piston_Crank_Assembly
Mechanism_Configuration
World_Frame
Cylinder_Piston_Prismatic_Joint_A
Cylinder_Piston_Prismatic_Joint_B

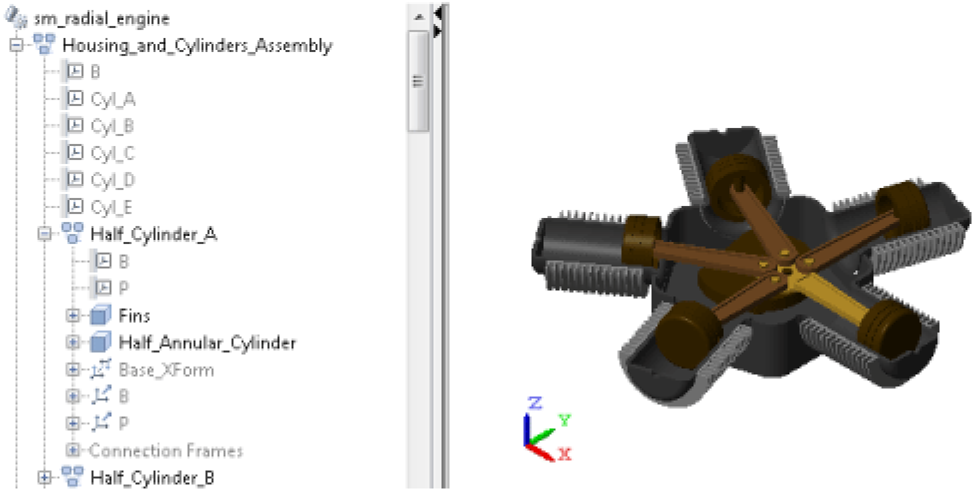
Nodes being visualized

Nodes not being visualized

## Model Hierarchy and Tree Nodes

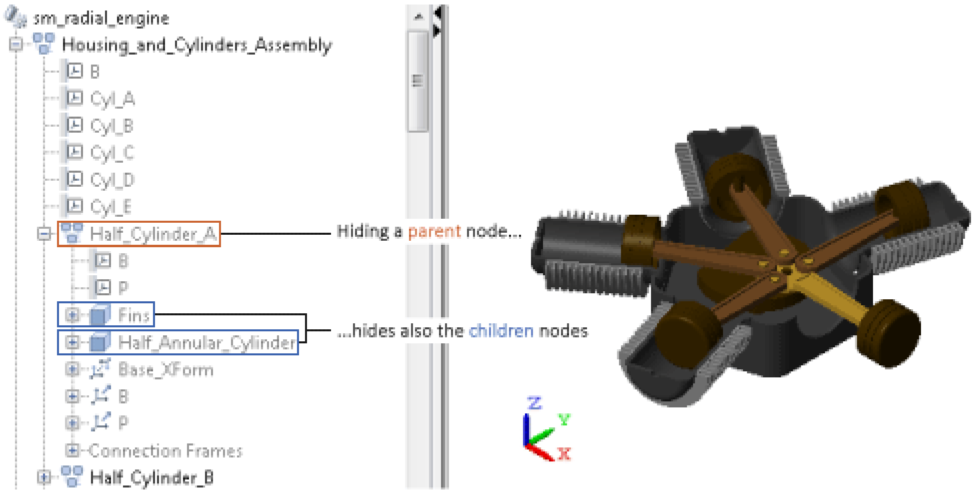
Multibody models are hierarchical in nature. They often contain multibody subsystems comprising rigid-body subsystems, each with one or more solids. The tree-view pane of Mechanics Explorer represents such a model structure through nodes arranged hierarchically. A node is a parent node if it contains 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).



Hiding a parent node...

...hides also the children nodes

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:

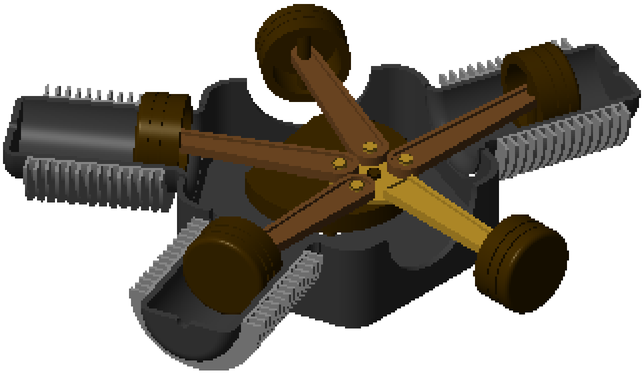
- Lowering the opacity of obstructive components—those obscuring other parts of the model—for example, making the cylinder encasing an engine piston transparent.
- Modeling obstructive components only in part—for example, treating engine cylinders as half-cylinders to preserve piston visibility during 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.

# Filter Radial Engine Visualization

**In this section...**

## 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 SimMechanics model, for example, to observe a model component that might otherwise remain obstructed during simulation. For more information, see "Filtering Model Visualization" on page 6-11.

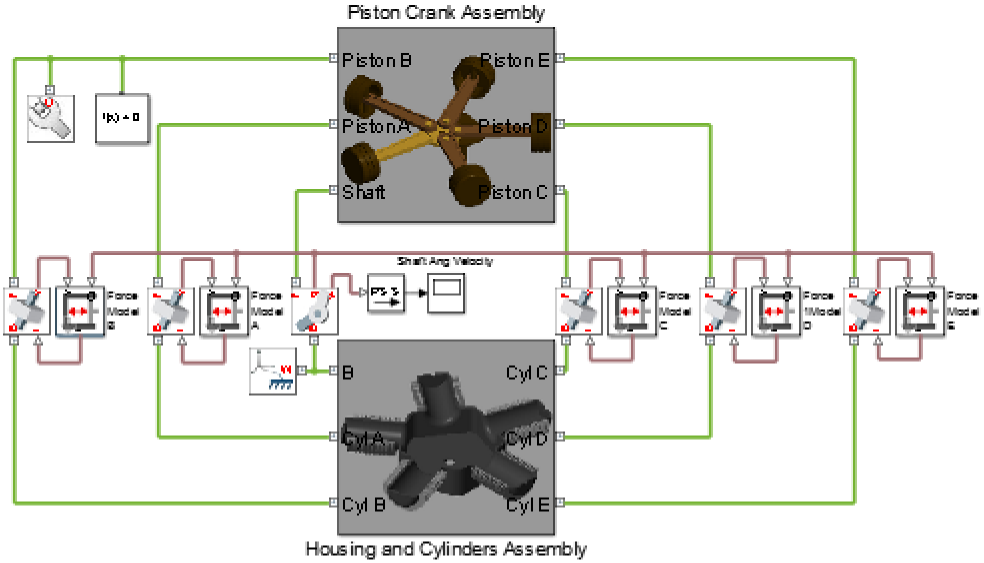


**Radial Engine Visualization with Two Cylinders Hidden**

## Open Example Model

In this tutorial, you filter the visualization of the SimMechanics radial engine featured example. To open this model, at the MATLAB command prompt, enter `sm_radial_engine`.

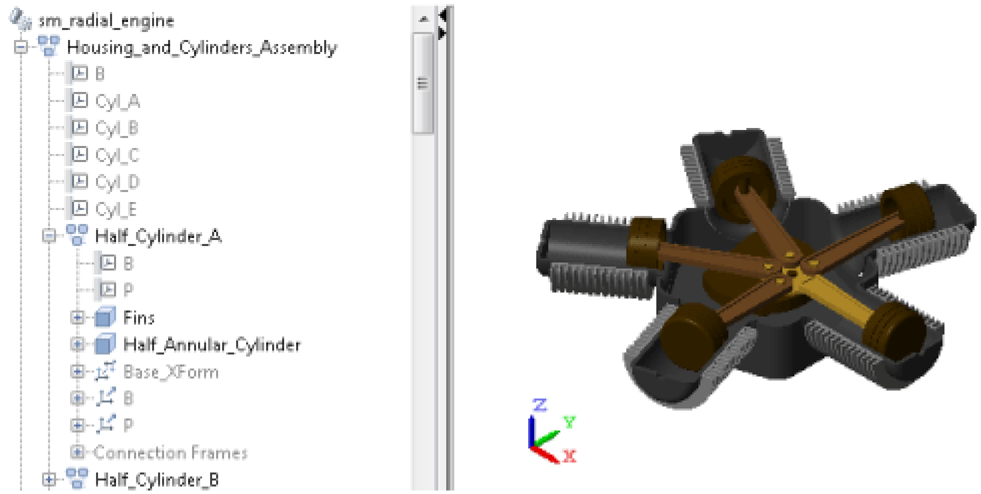
The model contains two top-level subsystems—the housing subsystem, named Housing_and_Cylinders_Assembly, and the piston subsystem, named Piston_Crank_Assembly. The housing subsystem contains five half cylinders. The piston subsystem contains five pistons that travel inside the half cylinders.



**Radial Engine Block Diagram**

## Update Example Model

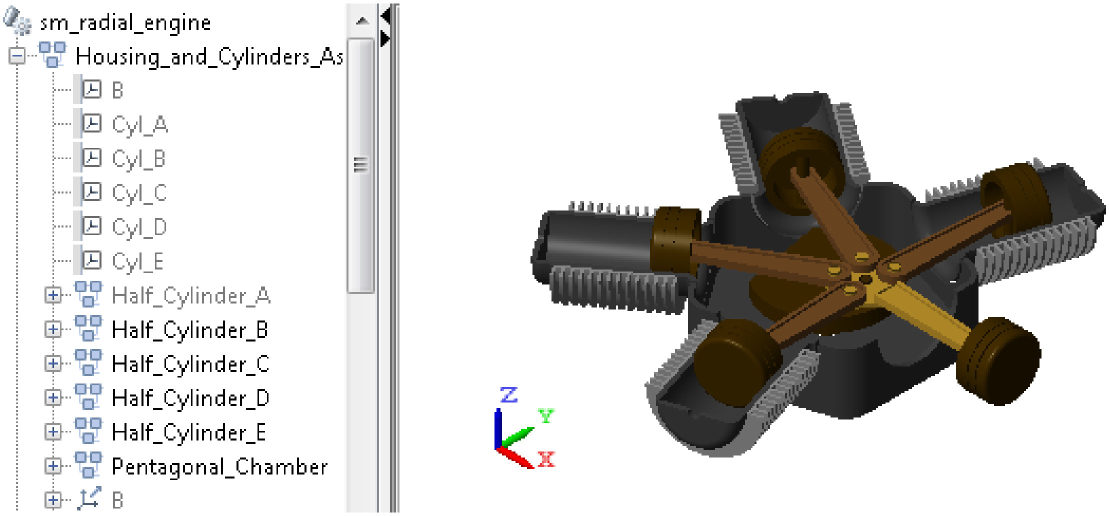
To open Mechanics Explorer, the SimMechanics visualization utility you must first update the example model. To do this, in the Simulink menu bar, select **Simulation** > **Update Diagram** (Windows shortcut **Ctrl + D**). Note the tree-view pane on the left side of Mechanics Explorer. You access the visualization filtering menu by right-clicking a node on this pane.



**Radial Engine Model Visualization**

## Hide Half-Cylinder Subsystem

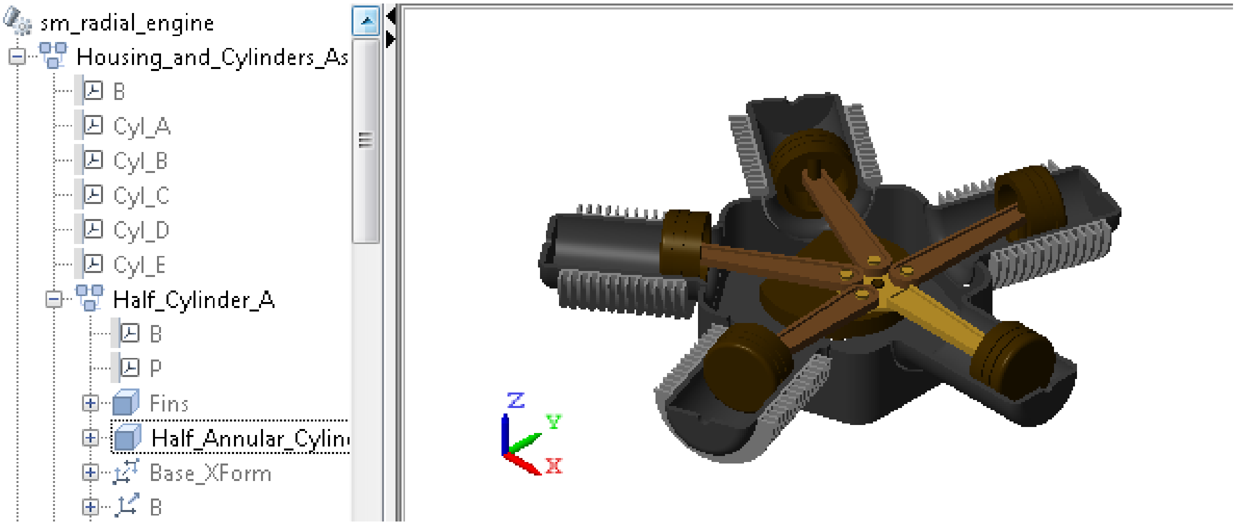
In the tree-view pane, expand the Housing_and_Cylinders_Assembly node. Right-click the Half_Cylinder_A node and select `Hide This`. Mechanics Explorer hides the half-cylinder subsystem and the solids it contains, corresponding to the nodes Fins and Half_Annular_Cylinder. The hidden-node names are grayed 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

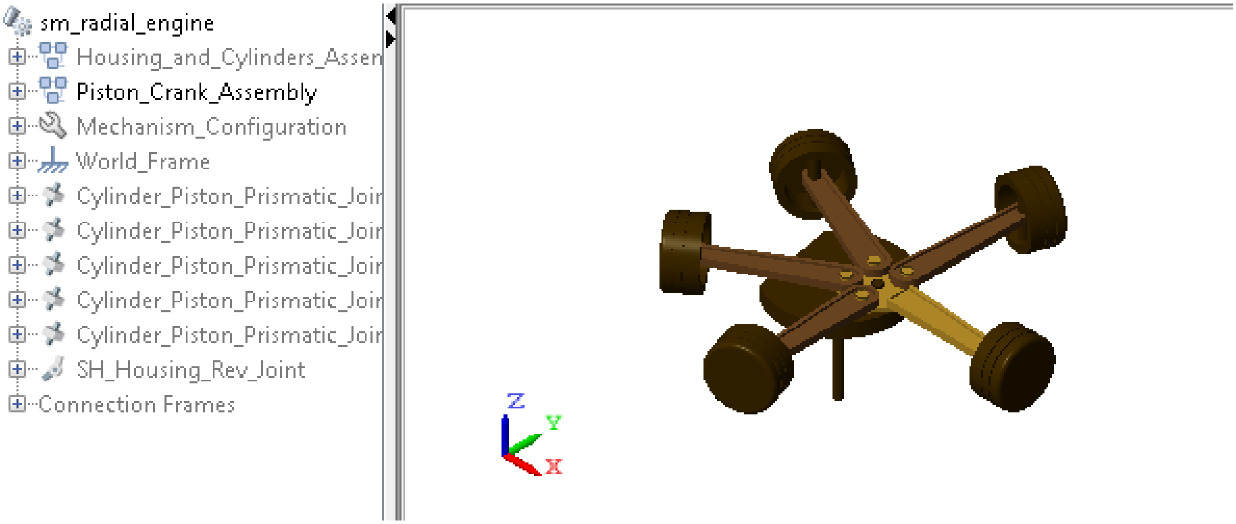
In the tree-view pane, expand the Half_Cylinder_A node. Then, right-click the Half_Annular_Cylinder node and select `Show This`. The half-cylinder solid is now visible, but the remainder of its parent of its parent subsystem—in this case, just the Fins solid—remains hidden. The newly visible half-cylinder 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

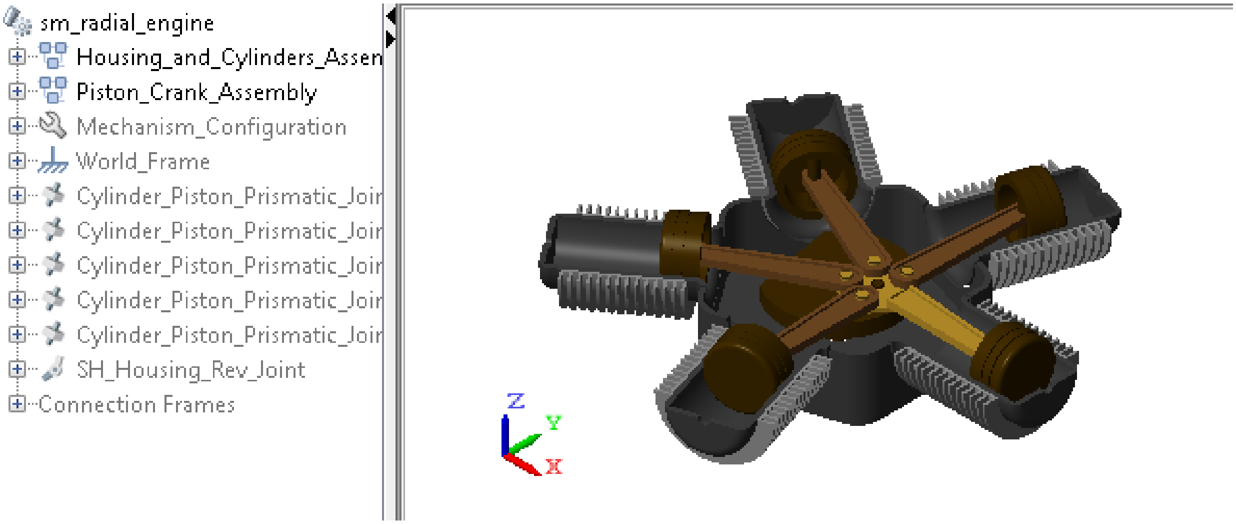
In the tree-view pane, collapse the Housing_and_Cylinders_Assembly node. Then, right-click the Piston_Crank_Assembly node and select `Show Only This`. Mechanics Explorer shows the selected node and hides the remainder of the model. In the tree-view pane, the name of the selected node is the only that is not grayed out. The figure shows the resulting model visualization.


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

**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 Frames

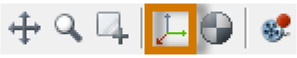
**In this section...**

## Frame Overview

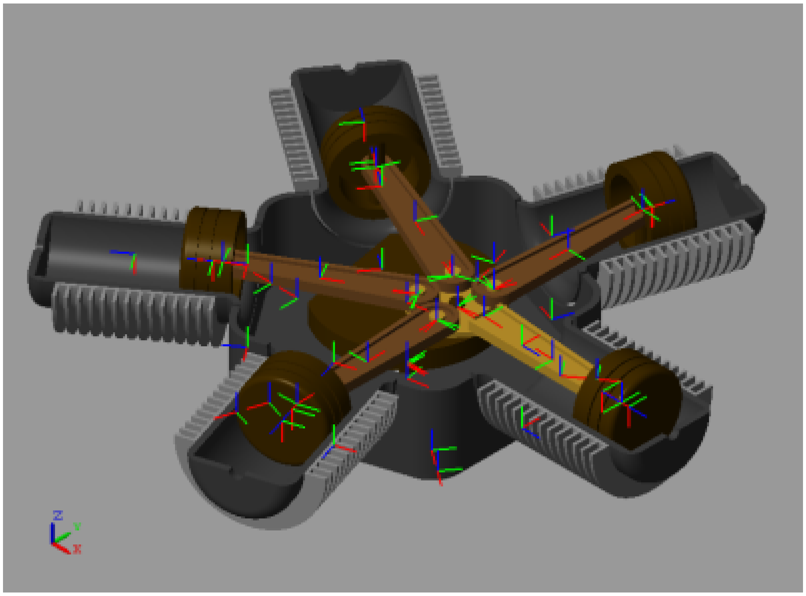
SimMechanics models are based on frames, abstract axis triads that contain all the position and orientation data in a model. These constructs enable you to connect solids into rigid bodies, assemble rigid bodies into mechanisms, and prescribe and sense forces, torques, and motion. Given their importance, 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 Frames** icon in the Mechanics Explorer tool strip, shown in the following figure.



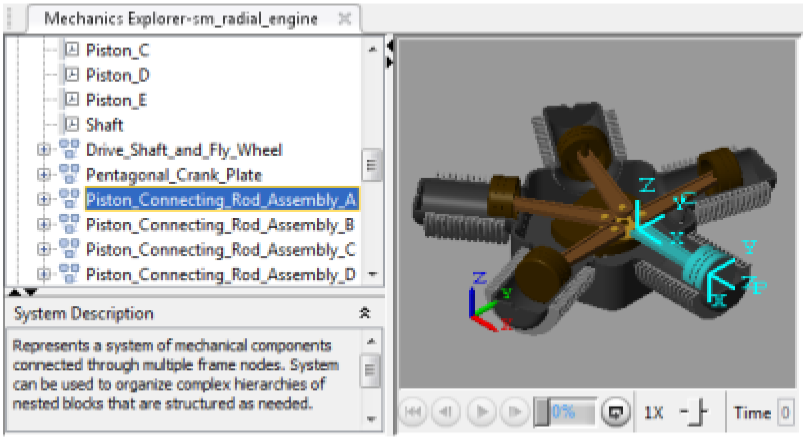
Alternatively, you can select **View** > **Toggle Frames** in the menu bar. Mechanics Explorer shows all the frames in your model, suiting this approach well for models with small numbers of frames. The figure shows a radial engine model with frame visibility toggled on.



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 Individual 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.



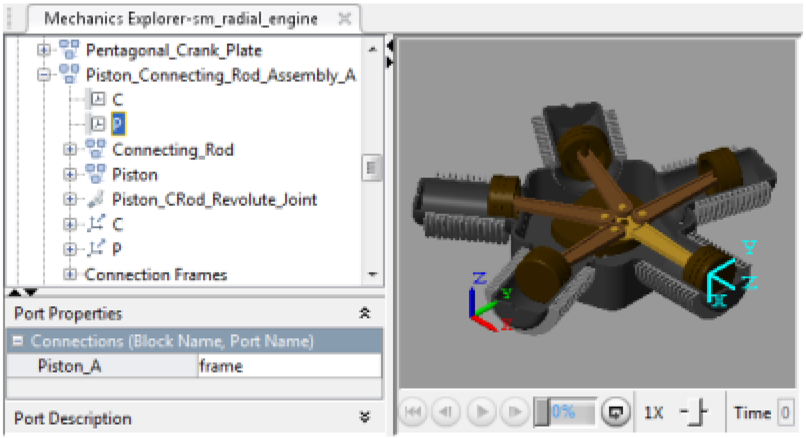
Mechanics Explorer-sm_radial_engine

Piston_C
Piston_D
Piston_E
Shaft
Drive_Shaft_and_Fly_Wheel
Pentagonal_Crank_Plate
Piston_Connecting_Rod_Assembly_A
Piston_Connecting_Rod_Assembly_B
Piston_Connecting_Rod_Assembly_C
Piston_Connecting_Rod_Assembly_D

System Description

Represents a system of mechanical components connected through multiple frame nodes. System can be used to organize complex hierarchies of nested blocks that are structured as needed.

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.



Mechanics Explorer-sm_radial_engine

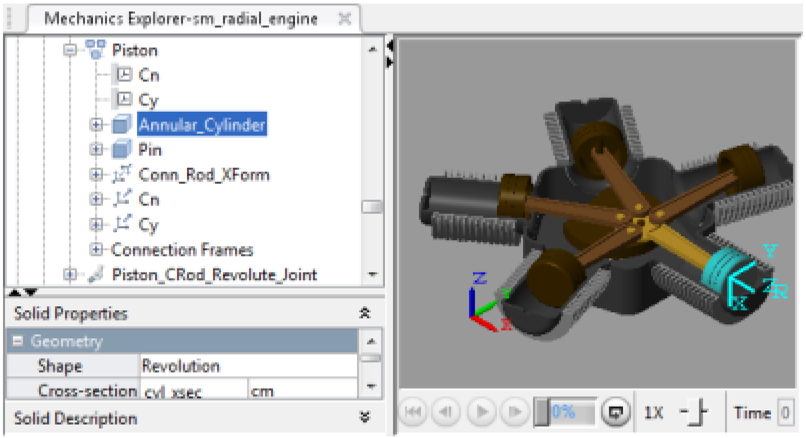
Pentagonal_Crank_Plate
Piston_Connecting_Rod_Assembly_A
C
P
Connecting_Rod
Piston
Piston_CRod_Revolute_Joint
C
P
Connection Frames

Port Properties

| Connections (Block Name, Port Name) | |
|---|---|
| Piston_A | frame |

Port Description

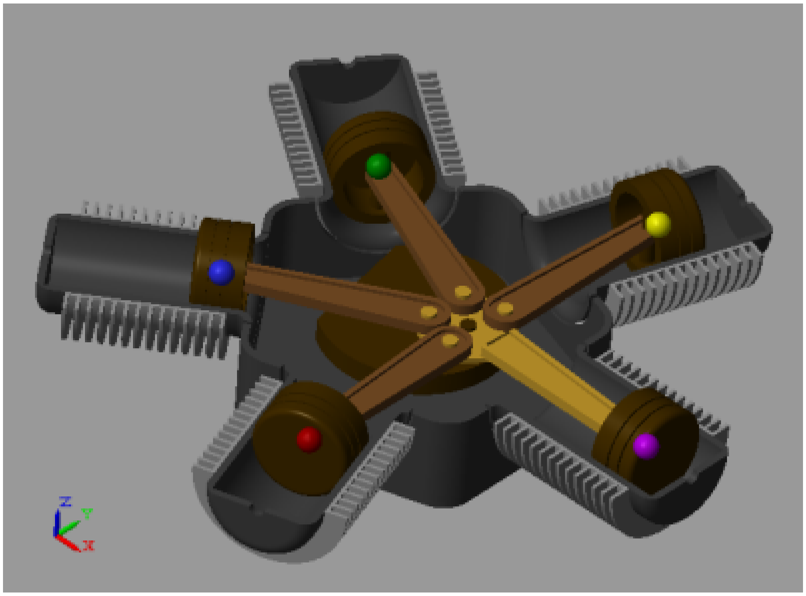
Finally, you can select individual solids directly in the visualization pane, highlighting their reference frames. The figure shows the result of selecting one of the piston solids directly. Mechanics Explorer highlights the solid and its reference frame, while the tree view pane reveals the associated Solid block name. This is the block that you need to change if you want to modify this particular solid.



| | | |
|---|---|---|
| Shape | Revolution | |
| Cross-section | cyl_xsec | cm |

## Visualize Frames with 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.


Z
Y
X

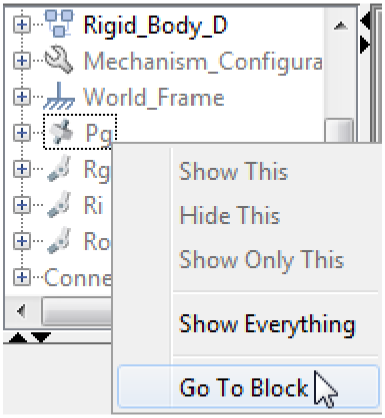
# Go to Block from Mechanics Explorer

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.

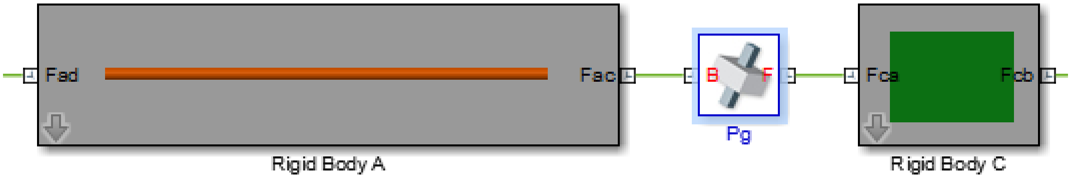
To help you troubleshoot such modeling issues, Mechanics Explorer enables you to go directly to a block associated with a node in the tree view pane. This feature helps you also to iterate on a model that is working properly, for example, if you want to replace a body subsystem with an alternative version.

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**. SimMechanics 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 “Find and Fix Aiming-Mechanism Assembly Errors” on page 3-26.

# Configure Model for Video Recording

**In this section...**

## Correspondence Between Animation and Simulation Speeds

Animation videos play at a fixed rate of 30 frames per second. Each video frame, lasting 1/30 s, corresponds to a simulation output time step. To ensure that an animation video plays at normal speed—i.e., that one second of video playback time corresponds to one second of simulation time—you must ensure that each output time step too lasts 1/30 s. You can do this by adjusting the configuration parameters for your model. The exact parameters to adjust depend on your choice of solver: variable-step or fixed step.

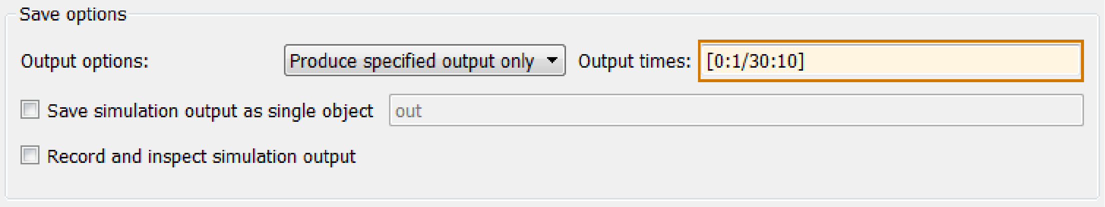
## Configure Model with Variable-Step Solver

Variable-step solvers generate simulation data at time steps of different durations. The different time steps can produce time distortion in your video, causing it to speed up as the time steps contract and to slow down as the time steps expand. To prevent time distortion in your video and ensure that it plays at the correct speed, you must specify the desired output times in the model configuration parameters:

1. In the Simulink Editor menu bar, select **Simulation** > **Model Configuration Parameters**.
2. In the **Configuration Parameters** tree pane, select **Data Import/Export**.
3. In **Output options**, select `Produce specified output only`.
4. In **Output times**, enter `[`$t_i$`:1/f:`$t_f$`]`, where:

   - $t_i$ is the simulation start time. The default value is `0` s.
   - *f* is the animation video frame rate. This rate is fixed at 30 frames per second. The fraction 1/f is the required output time step for a video to play at a normal speed.
   - $t_f$ is the simulation stop time. The default value is `10` s.

The output times vector determines the time steps at which to output simulation data. By specifying time steps in 1/30 s intervals, you ensure that each animation video frame corresponds to a simulation time step of equal size. The result is an animation video that plays at normal speed without distortion.



Save options

Output options: Produce specified output only ▾ Output times: [0:1/30:10]

□ Save simulation output as single object out

□ Record and inspect simulation output

You can also manipulate the output time step to generate videos that play at different speeds. For example, to generate a video that plays at twice the normal speed, multiply the output time step by 2. The output times vector becomes `[`$t_i$`:2/f:`$t_f$`]`. Similarly, to generate a video that plays at half the normal speed, divide the output time step by 2. The output times vector becomes `[`$t_i$`:1/(2*f):`$t_f$`]`.

## Configure Model with Fixed-Step Solver

Video time distortion cannot occur when you use a fixed-step solver. However, the animation video can still play back at an unrealistic speed. To ensure that one second of playback time corresponds to one second of simulation time using a fixed-step solver, you must adjust the solver step size directly:

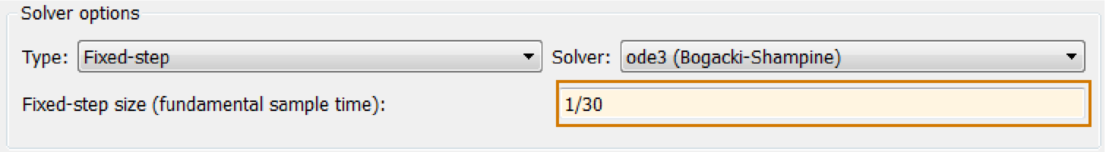
1 In the Simulink Editor menu bar, select **Simulation** > **Model Configuration Parameters**.

2 In the **Configuration Parameters** tree pane, select **Solver**.

3 Under **Solver options**, set **Type** to `Fixed-step`.

4 Set **Solver** to an appropriate fixed-step solver for your model.

5 In **Fixed-step size (fundamental sample time)**, enter `1/30`.



Solver options

Type: Fixed-step ▾ Solver: ode3 (Bogacki-Shampine) ▾

Fixed-step size (fundamental sample time): 1/30

Changing the fixed-step size from `1/30` causes the recorded animation to play back at a different speed. For example, doubling the step size to `2/30` causes the recorded animation to play back at twice the normal speed. Similarly, changing the step size to `1/60` causes the recorded animation to play back at half the normal speed.

**Note:** When using a fixed-step solver, you cannot specify the simulation output times. You must change the solver time step directly.

Model dynamics should take precedence over video playback considerations. Select a solver and step size based on the dynamics of your model. Then, if possible, adjust the time step to control the video playback speed.

# Record Animation Video

You can record a video of a model animation in Mechanics Explorer. The video enables you to document and share in part your simulation results without having to rerun the simulation. Anyone with a basic media player with AVI support can then watch the animation without having SimMechanics installed.

If you have not already done so, “Configure Model for Video Recording” on page 6-31 before proceeding. This step ensures that your video plays at the correct speed and without any time distortion. To record the video:

1. Update or simulate your model. If Mechanics Explorer does not open, see “Turn Model Visualization Off and On” on page 6-35.
2. In the Mechanics Explorer menu bar, select **Tools** > **Create Video**.
3. When prompted, save the video file in a convenient folder. A new window opens with a display of the model and the recording progress.
4. When the message `Video file "filename.avi" has been successfully created` appears on your screen, press **OK**. `filename` is the path and name of your video file.

# Turn Model Visualization Off and On

You may want to suppress model visualization, e.g., when iterating simulation a large number of times with a MATLAB script. You can prevent Mechanics Explorer from opening on model update or simulation by changing the model configuration parameters:

1. In the SimMechanics menu bar, select **Simulation** > **Model Configuration Parameters**.
2. Expand the **SimMechanics 2G** node and select **Explorer**.
3. Clear the **Open Mechanics Explorer on model update or simulation** check box.

To enable model visualization once again, select the **Open Mechanics Explorer on model update or simulation** check box.

# Find and Fix Visualization Issues

| In this section... |
|---|
| "Mechanics Explorer Not Opening" on page 6-36 |
| "Model Showing Sideways in Mechanics Explorer" on page 6-36 |
| "Parts Not Showing in Mechanics Explorer" on page 6-37 |
| "Colored Parts Showing Gray in Mechanics Explorer" on page 6-39 |

## Mechanics Explorer Not Opening

By default, Mechanics Explorer is set to open the first time you update a model. If a Mechanics Explorer window is already open for your model, the open window updates the model display. Note, however, that updating a model does not automatically bring the Mechanics Explorer window to the front. If the Mechanics Explorer window is hidden during model update, you must bring that window to the front to see the updated model.

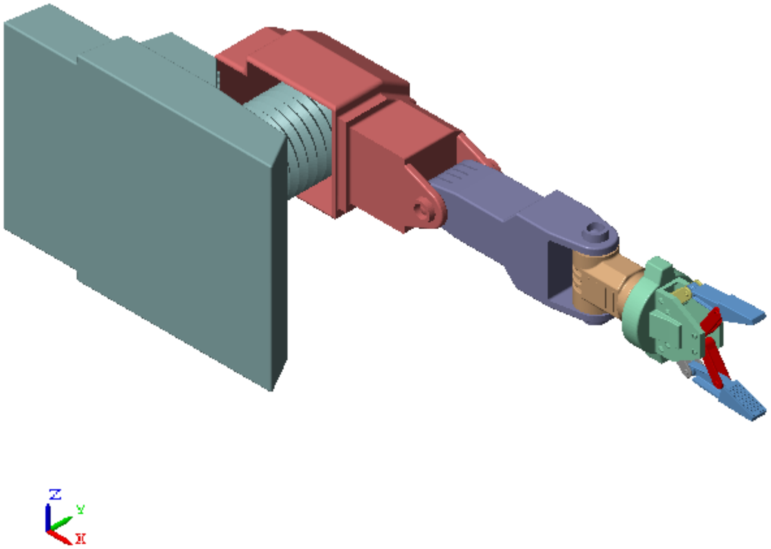
### Set Mechanics Explorer to Open on Model Update

If Mechanics Explorer fails to open during model update, check that Mechanics Explorer is set to open on model update:

1. In the Simulink Editor menu bar, select **Simulation** > **Model Configuration Parameters**.
2. Expand the **SimMechanics 2G** node.
3. Click **Explorer**.
4. Verify that **Open Mechanics Explorer on model update or simulation** is selected.

## Model Showing Sideways in Mechanics Explorer

By default, Mechanics Explorer displays a model with the Z axis of the World frame pointing up. Using this convention, the default gravity vector `[0 0 -9.81] m/s^2` points down, a direction that is practical for most applications. However, this convention differs from that which CAD platforms commonly use, Y axis up, causing Mechanics Explorer to display some models sideways. If this happens, you can manually change the view convention to that used in the original CAD assembly. The figure shows the default Mechanics Explorer display of an imported robot arm model.



### Change View Convention

To change the view convention of a model:

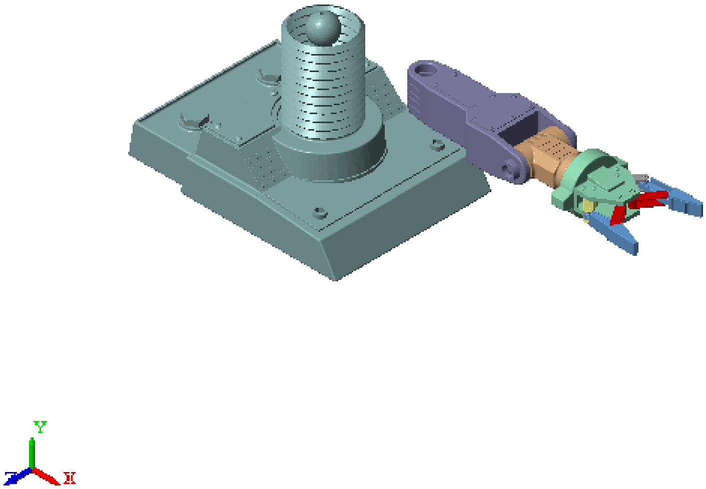
1. In the Mechanics Explorer toolbar, click the **View Convention** drop-down menu.
2. Select **Y up (ZX Top)**.
3. Refresh the Mechanics Explorer display by selecting a view point from the Mechanics Explorer tool bar.

Mechanics Explorer displays the model using the new view convention.

## Parts Not Showing in Mechanics Explorer

During CAD import, SimMechanics uses a set of geometry files in STEP or STL format to generate the 3-D surface geometry of each CAD part. If SimMechanics cannot load the geometry file for a part, that part appears invisible in Mechanics Explorer. This issue does not affect model update or simulation.

The figure shows the Mechanics Explorer display of an imported model containing an invalid geometry file.



### Correct Visualization Issue

If a part of an imported model appears invisible in Mechanics Explorer:

1. In Mechanics Explorer, identify the name of each invisible part.
2. In the block diagram, open the dialog boxes of the associated Solid blocks.
3. In the **Geometry** section, check that the name and location of the geometry files are correct.

   If either is incorrect, enter the correct information and update the model. Check that Mechanics Explorer displays the invisible part. If not, check if the geometry files are valid.

### Geometry File Issues

To visualize a CAD assembly that you import, SimMechanics relies on a set of geometry files that specify the 3-D surface geometry of the CAD parts. Each geometry file specifies the surface geometry of one CAD part as a set of 2-D triangles. To do this, the geometry files contain:

- [X Y Z] coordinates of the triangle vertices
- [X Y Z] components of the normal vectors for the triangles.

If a geometry file specifies a normal vector with zero length, SimMechanics issues a warning. The geometry file fails to load.

## Colored Parts Showing Gray in Mechanics Explorer

To import the color of a CAD part into SimMechanics, the color must be specified at the part level. Colors specified at the feature level, such as individually colored surfaces, or assembly level do not carry over into SimMechanics. This can cause parts to appear gray —the default solid color in SimMechanics—during model visualization. To correct this issue, you must modify the CAD part file directly to ensure that it contains a single color applied to the entire part.

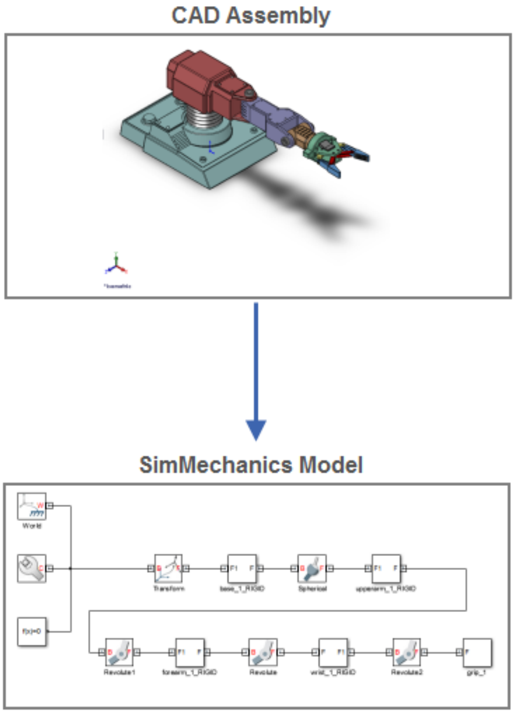
# CAD Import

# About CAD Import

# CAD Translation

**In this section...**

You can translate a CAD assembly into a SimMechanics model for simulation and analysis. This process is called CAD translation. By translating a CAD assembly into a SimMechanics model, you leverage the strengths of your CAD platform with the strengths of SimMechanics software. You can modify any model that you translate—for example, adding actuators and sensors—to fit the needs of your application. CAD translation is especially useful for control system design.

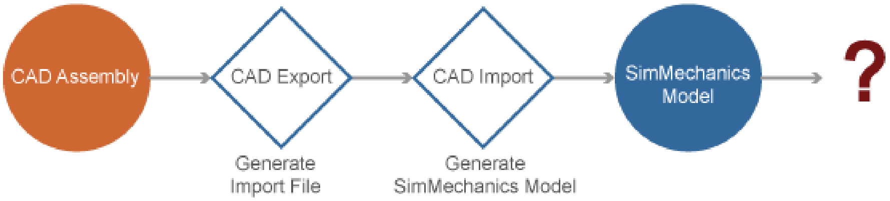


## CAD Translation Steps

CAD translation is a two-step process. First, you export a CAD assembly in XML format. Then, you import the XML file into SimMechanics. SimMechanics uses the XML file to automatically generate a model that replicates the original CAD assembly. If the CAD assembly contains only supported constraints, CAD import requires no additional work on your part. Once SimMechanics generates your model, you are ready to simulate and analyze that model. The table summarizes the two CAD translation steps.

| Translation Step | Description |
|---|---|
| CAD Export | Generate XML import file from CAD assembly |
| CAD Import | Generate SimMechanics model from import files |

You must export a CAD assembly before you import it into SimMechanics. The schematic shows the CAD translation step sequence. A CAD assembly is the starting point of CAD translation. Exporting that assembly in XML format and importing the resulting XML file into SimMechanics produces an equivalent SimMechanics model.



## Software Requirements

The table provides the software requirements for CAD translation. The requirements depend on the CAD translation step—export or import. For example, a CAD platform is a requirement only for CAD export.

| Software | Notes | CAD Export | CAD Import |
|---|---|---|---|
| CAD Platform | | ✓ | |
| MATLAB | Registration as computing server required | ✓ | ✓ |
| SimMechanics | | | ✓ |
| SimMechanics Link | | ✓ | |

The software requirements for CAD translation are optimized for cooperation between CAD and SimMechanics engineers. A CAD engineer can export the CAD assembly without an active SimMechanics installation. Likewise, a SimMechanics engineer can import the CAD assembly without an active CAD platform installation.

## See Also

`smimport`

## Related Examples

- “Install and Register SimMechanics Link Software” on page 7-9
- “Import Robot Arm Model” on page 7-11
- “Import Stewart Platform Model” on page 7-16
- “Find and Fix CAD Import Issues” on page 7-21

## More About

- “CAD Import” on page 7-5

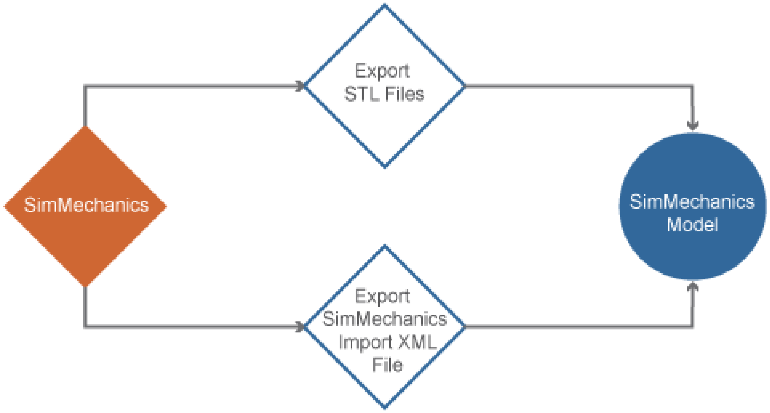
# CAD Import

**In this section...**

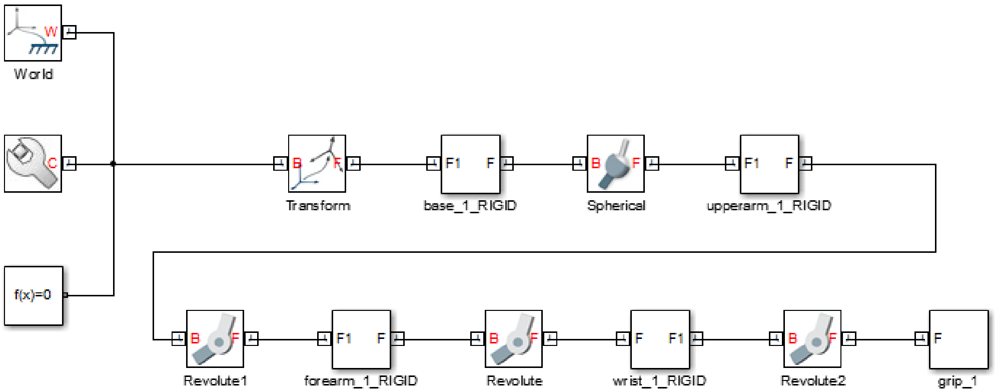
CAD Import is the second and final step of CAD translation. During CAD import, SimMechanics interprets the SimMechanics Import XML file generated during CAD Export. Then, based on the structure and parameters that the XML file provides, SimMechanics automatically generates model that replicates the original CAD assembly.

## Importing a Model

CAD Import does not require access to the original CAD assembly or associated CAD platform. Access to the surface-geometry files is not required for simulation, but it is required for visualization. You can simulate an imported model that contains no geometry files. However, the Mechanics Explorer visualization utility cannot display a representation of a model without the geometry files.



In the model, each CAD part maps into a rigid body subsystem. Each CAD constraint or set of CAD constraints, map into a joint. Block names for SimMechanics subsystems are based on the original CAD parts and subassemblies which the subsystems represent. SimMechanics appends the suffix `RIGID` to the stem of a rigid body name. For example, CAD part `base` translates into rigid body subsystem `base_RIGID`. The following figure shows the imported SimMechanics model of a CAD robot assembly.

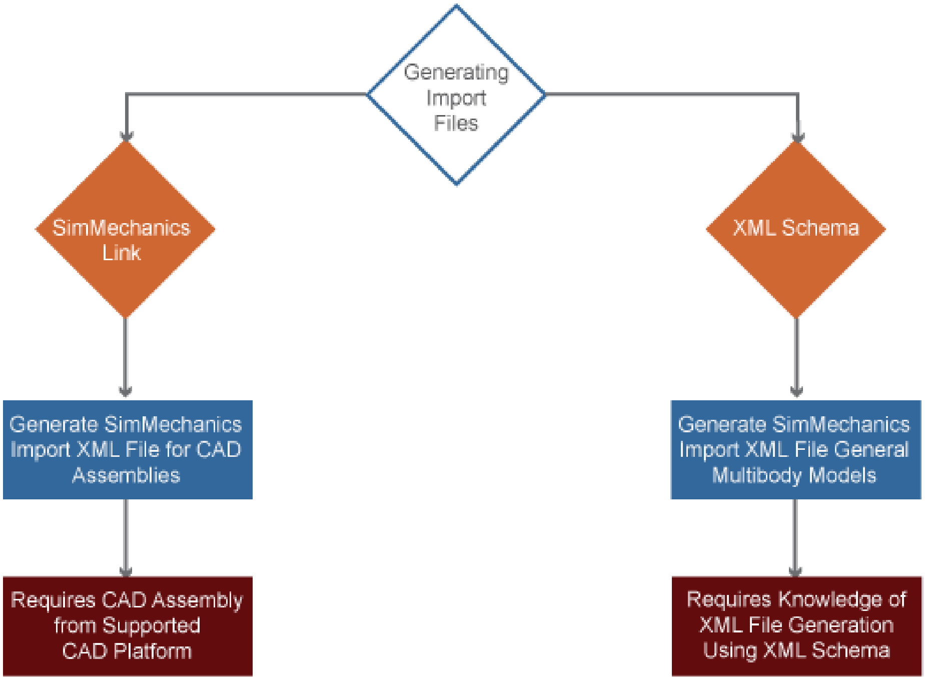


Modify SimMechanics model to fit the needs of your application.

## Generating Import Files

To import a multibody model into SimMechanics, you must first generate the SimMechanics Import XML file. You can generate this file automatically, using the SimMechanics Link utility, or manually, using the XML schema that MathWorks® provides. The method that you use depends on the type of model that you want to import. The table summarizes the two methods and their limitations.

| Import File Generation Method | Limitations |
|---|---|
| SimMechanics Link | Works only for CAD assemblies. CAD assembly must come from one of three supported CAD platforms. |
| XML Schema | Requires knowledge of XML file generation based on XML schema |


Generating Import Files
SimMechanics Link
XML Schema
Generate SimMechanics Import XML File for CAD Assemblies
Generate SimMechanics Import XML File General Multibody Models
Requires CAD Assembly from Supported CAD Platform
Requires Knowledge of XML File Generation Using XML Schema

SimMechanics Link is a free utility that MathWorks provides. Use this utility to generate the SimMechanics Import XML file that you need to import a CAD assembly into SimMechanics. For more information about SimMechanics Link , see "Install and Register SimMechanics Link Software" on page 7-9.

## SimMechanics XML Schema

The XML Schema is a set of files written according to the W3C XML Schema specification. MathWorks provides these files so that you can generate a SimMechanics Import XML file manually or using an external application. Use the XML Schema to generate the SimMechanics Import XML file for a CAD assembly or other multibody model.

The XSD files describe the elements and attributes that a SimMechanics Import XML file can contain and the order in which they must appear. Generating an XML file in accordance with the XML schema ensures that SimMechanics can successfully import it. Once you have generated the XML file, validate it against the schema to ensure SimMechanics can import it without issue.

To access the SimMechanics XML schema, visit the SimMechanics product website. Follow instructions to download the XSD files.

## See Also

`smimport`

## Related Examples

- “Install and Register SimMechanics Link Software” on page 7-9
- “Import Robot Arm Model” on page 7-11
- “Import Stewart Platform Model” on page 7-16
- “Find and Fix CAD Import Issues” on page 7-21

## More About

- “CAD Translation” on page 7-2

# Install and Register SimMechanics Link Software

**In this section...**

## 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 Installation Files

1. Go to the SimMechanics Link download page.
2. Follow the prompts on the download page.
3. Save the zip archive and .m 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 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 Automation Server

Each time you export a CAD assembly, the SimMechanics 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 SimMechanics Link Plug-In

Before you can export an assembly, you must enable the SimMechanics Link plug-in on your CAD application. To do this, see:

- “Enable SimMechanics Link Inventor Plug-In”
- “Enable SimMechanics Link Creo-Pro/E Plug-In”
- “Enable SimMechanics Link SolidWorks Plug-In”

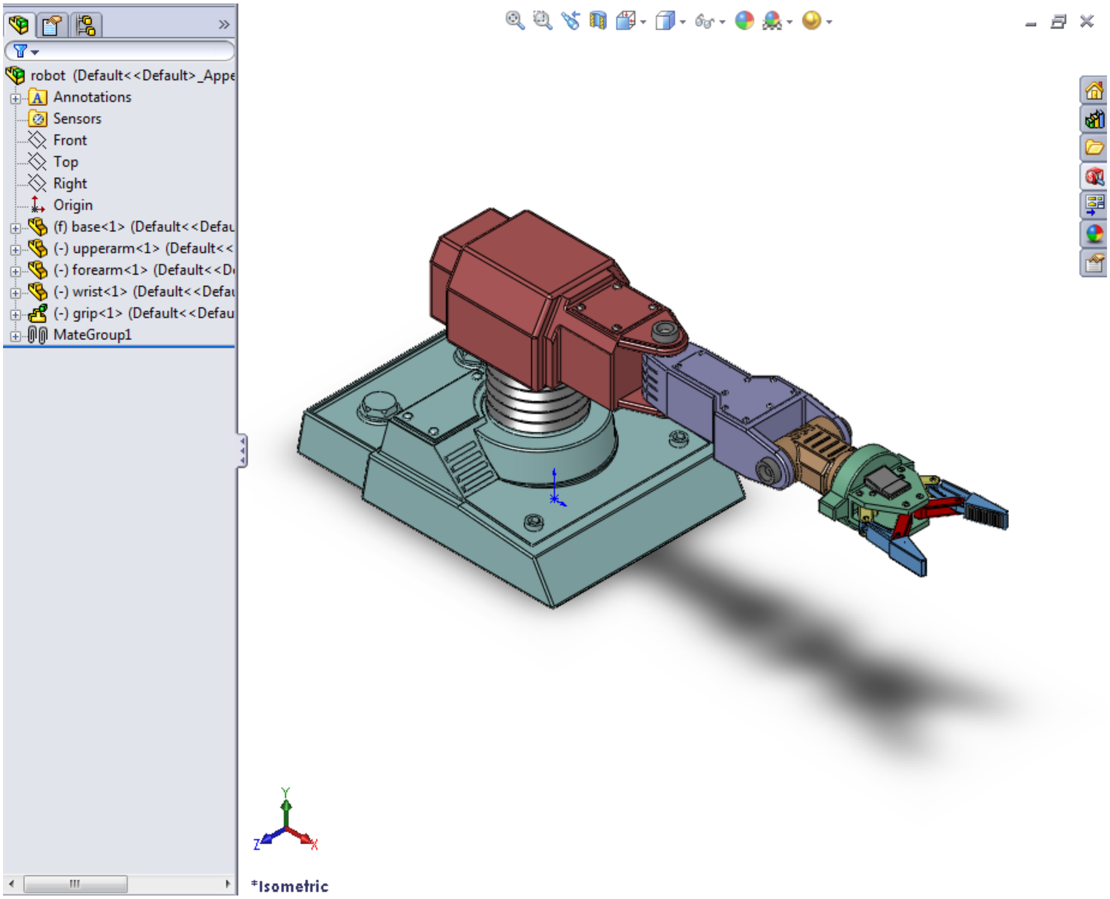
# Import Robot Arm Model

**In this section...**

In this example, you import a CAD assembly with name robot into SimMechanics. SimMechanics provides the `smimport` command so that you can import a CAD assembly. The command is the only SimMechanics tool you need to import a CAD assembly. The CAD import procedure is the same for all CAD platforms.

**Note:** This example uses an XML file and a set of geometry files that are present in your SimMechanics installation. You can export the XML and geometry files directly from a supported CAD platform, but the names of the files may differ from the example.

The following figure shows the original CAD assembly inside the SolidWorks CAD platform.


robot (Default<<Default>_Appe
Annotations
Sensors
Front
Top
Right
Origin
(f) base<1> (Default<<Defau
(-) upperarm<1> (Default<<
(-) forearm<1> (Default<<D
(-) wrist<1> (Default<<Defau
(-) grip<1> (Default<<Defau
MateGroup1
Y
Z
X
*Isometric

## Check Import Files

Before you import the sm_robot CAD assembly, check that the import files exist. The import files include one SimMechanics Import XML file and a set of geometry files that specify the geometry of all CAD parts.

1 At the MATLAB command line, enter the following command to change the current working directory to the subdirectory that contains the robot example files:

```
cd(fullfile(matlabroot,'toolbox','physmod','sm','smdemos',...
'import','robot'))
```

2 At the MATLAB command line, enter `ls` or `dir` to list all files in the `\robot` directory.

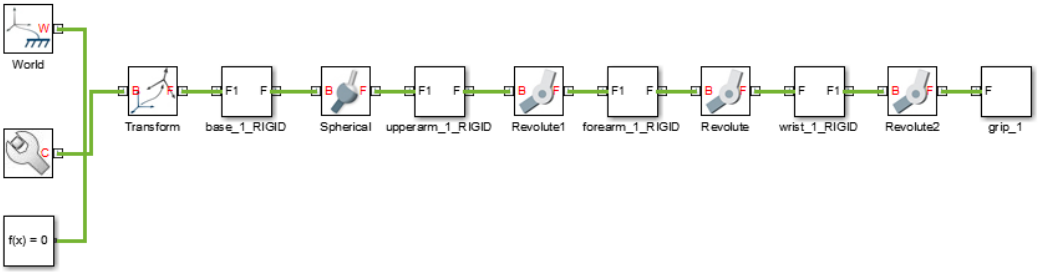
3 Check that the directory contains XML file sm_robot.xml and a set of geometry files.

## Import Robot Assembly

Once you have verified that all required files exist, proceed to import the assembly.

1 At the MATLAB command line, enter `smimport('sm_robot.xml')`.

2 Confirm that SimMechanics opens a new model with name `sm_robot`.



**Note:** SimMechanics automatically generates the new model without extra input on your part. Review the model and check for errors and inconsistencies in the block diagram.

3 In the Simulink Editor window that contains the model, select **File** > **Save As**.

4 In the **Save As** dialog box, enter the desired file name and select a convenient directory in which store the model file.

## Visualize and Simulate Robot Assembly

1 In the Simulink Editor window that contains the robot model, select **Simulation** > **Update Diagram** or press **Ctrl+D**.

**Note:** When you update the diagram, SimMechanics automatically updates the model display in Mechanics Explorer. SimMechanics relies on the set of geometry files to represent the 3-D geometry of each CAD part. If the files are not available, SimMechanics still generates the model, but Mechanics Explorer cannot display the assembly.

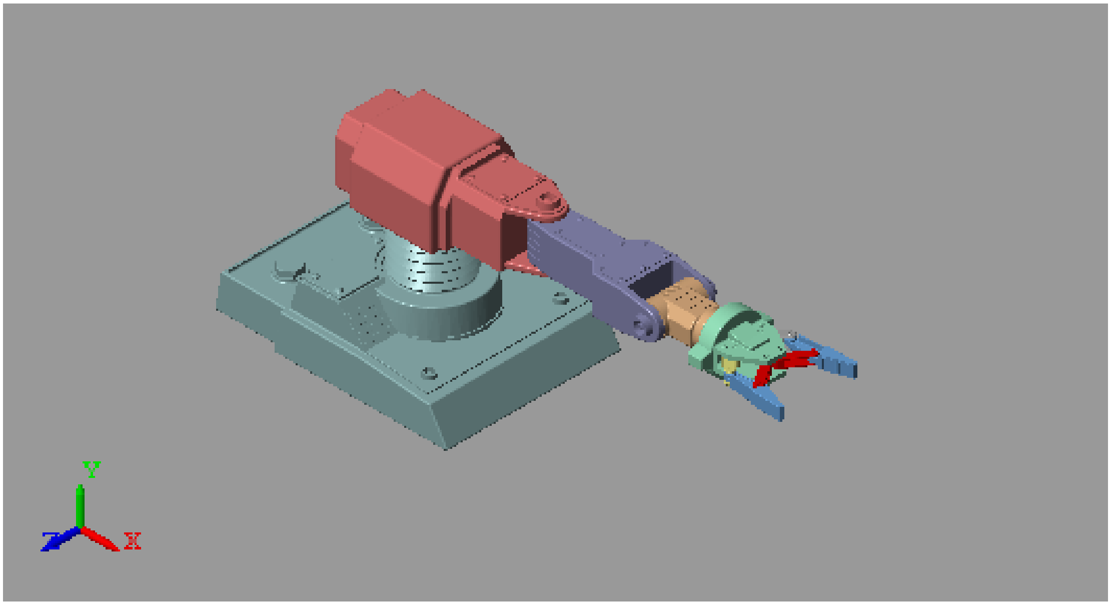
2 In the Mechanics Explorer toolbar, set **View Convention** to `Y up (XY Front)`.

**Note:** Most CAD systems use a `Y up` default view convention. The convention differs from the Mechanics Explorer default setting, `Z up`. Selecting the `Y up` view convention causes Mechanics Explorer to display the assembly with the same orientation used in the CAD platforms.

3 In the toolbar, click the icon for the desired viewpoint.

**Note:** Selecting the `Y up` view convention does not affect the Mechanics Explorer display until you click a view point. You have the choice between seven standard viewpoints: front, back, top, down, left, right, and isometric. Once you select a view point, you can rotate, pan, and zoom to adjust the display of your model.

4 Confirm that a Mechanics Explorer window opens with a static display of the robot assembly.



5 In the Simulink Editor window for the model, select **Simulation** > **Run** or press **Ctrl+T** to simulate the model.

**Tip** The model lacks actuation inputs. When you simulate the model, the robot arm moves strictly due to gravity effects. You can change the gravity specification in the `Mechanism Configuration` block.

You can add actuation inputs to the model. Add a block from the Forces & Torques library to actuate a rigid body. Select an actuation mode in the model joint blocks to actuate a joint.

## See Also

`smimport`

## Related Examples

- “Install and Register SimMechanics Link Software” on page 7-9
- “Import Stewart Platform Model” on page 7-16
- “Find and Fix CAD Import Issues” on page 7-21

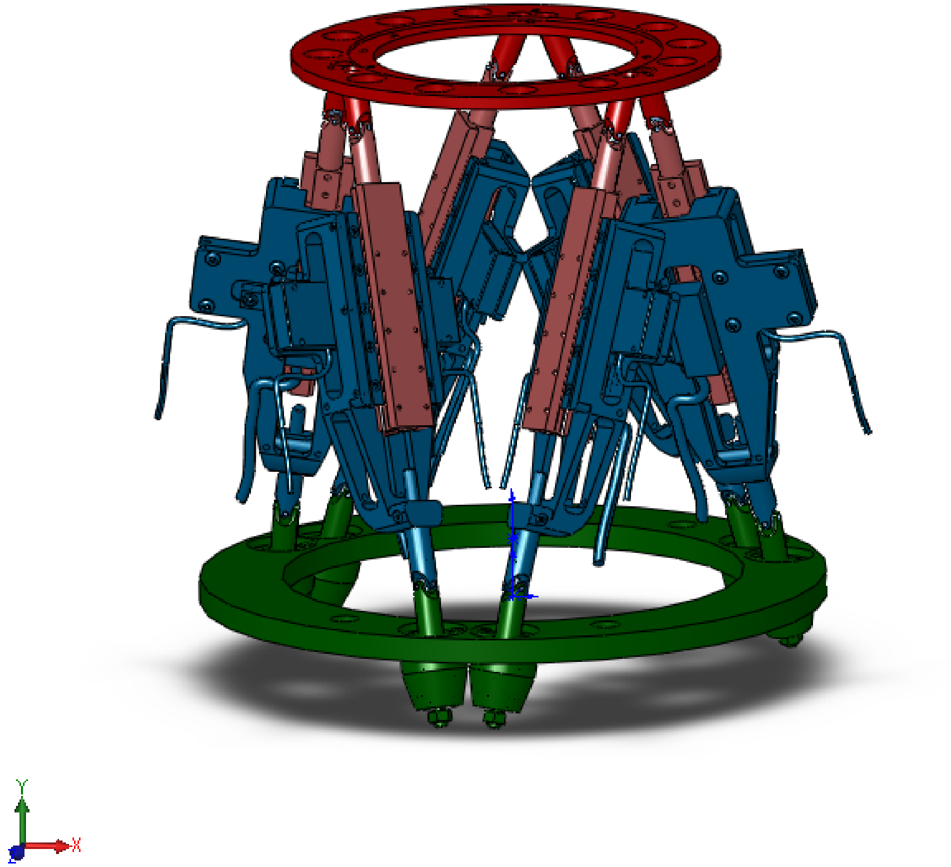
## More About

- “CAD Translation” on page 7-2
- “CAD Import” on page 7-5

# Import Stewart Platform Model

**In this section...**

You can import a CAD assembly into a SimMechanics model. To do this, you use the SimMechanics command `smimport`. In this example, you import the CAD assembly for a Stewart platform. All required files are provided with your SimMechanics installation.



## Check Import Files

To import the CAD assembly, you must have access to the SimMechanics Import XML and geometry files for this assembly. Check that you have these files before proceeding.

1 Navigate to directory

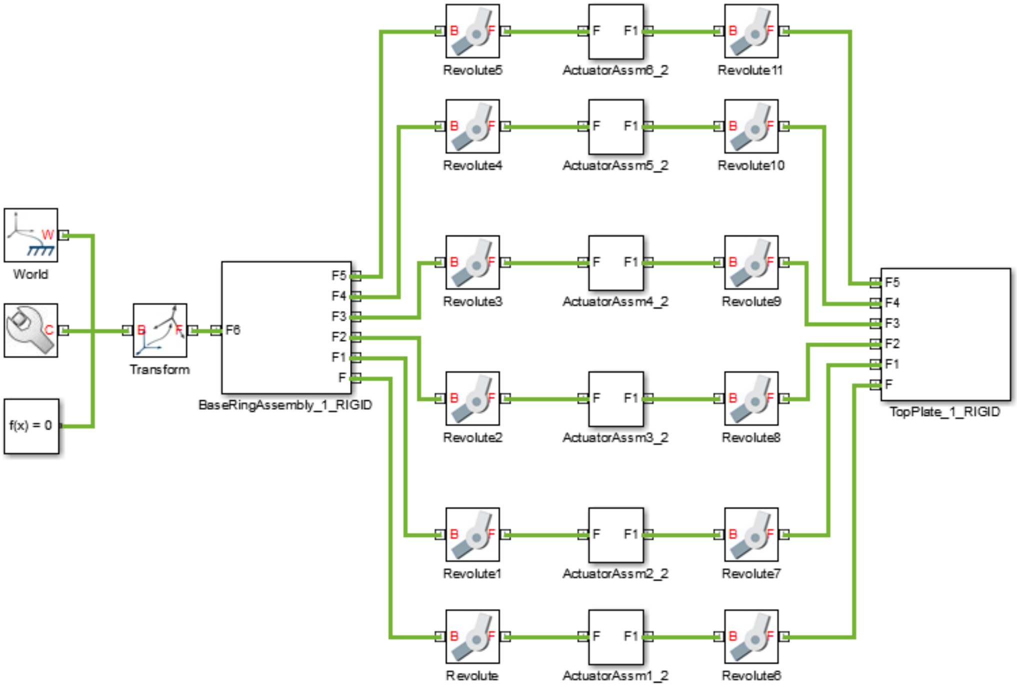
```
<matlabroot>/toolbox/physmod/sm/smdemos/...
...import/stewart_platform
```

2 Check that the following files exist.

| File | Quantity | Description |
|---|---|---|
| SimMechanics Import XML | One | Provides model structure and parameters |
| STEP or STL | Multiple | Provides part geometry |

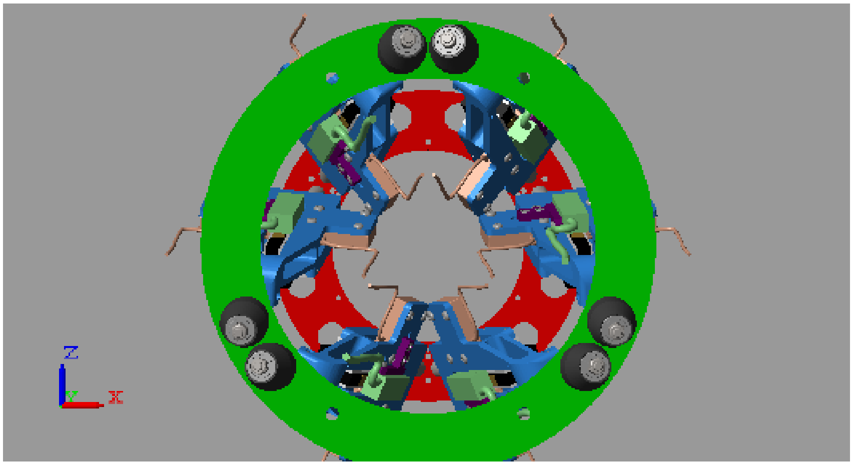
## Import Model

If you have access to the import files, you can import the model. To do this, at the MATLAB command prompt, enter `smimport('stewart_platform.xml')`. SimMechanics automatically generates a Stewart platform model. This model replicates the original CAD assembly.



## Visualize and Simulate Robot Assembly

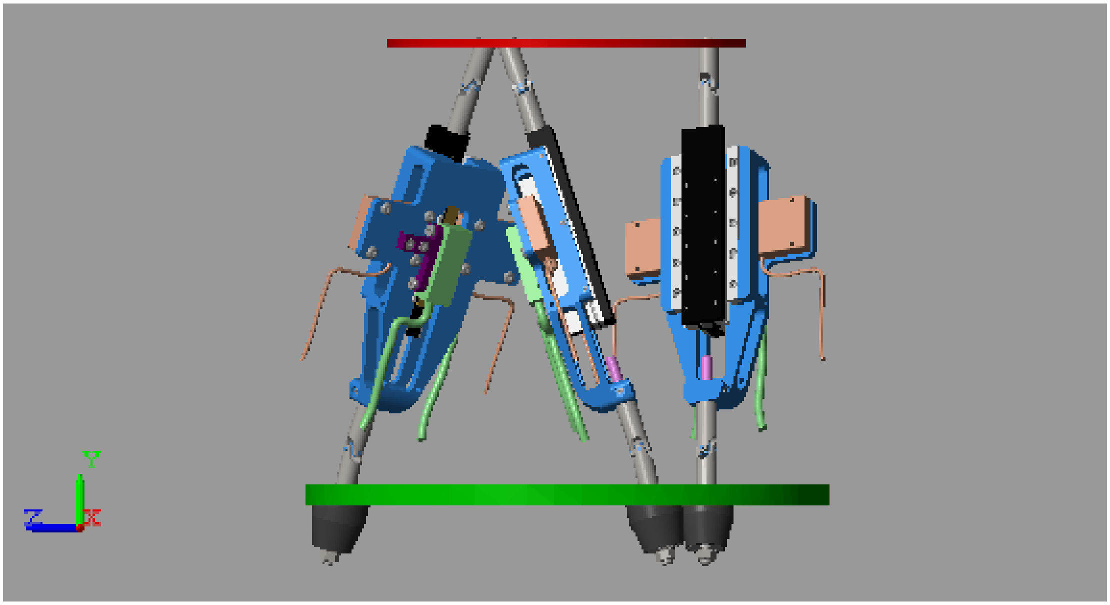
You can now simulate the model that you imported. On the Simulink tool bar, click the **Run** button. Alternatively, press **Ctrl+T**. Mechanics Explorer opens with a dynamic display of your model.



By default, Mechanics Explorer uses a Z axis up view convention. This convention differs from that which most CAD platforms use—Y axis up. The different view conventions cause the Stewart platform to appear sideways in the visualization pane. To fix this issue, change the Mechanics Explorer view convention to Y axis up:

- On the Mechanics Explorer tool bar, in the **View Convention** drop-down list, select **Y Up (XY Front)**.

To refresh the visualization pane using the new view convention, on the Mechanics Explorer tool bar, click any standard view button, e.g., Isometric View.



**Tip** Actuate the stewart_platform model with blocks from the **Forces and Torques** library. Then, simulate the model and analyze its dynamic behavior in Mechanics Explorer.

## See Also

`smimport`

## Related Examples

- "Install and Register SimMechanics Link Software" on page 7-9
- "Import Robot Arm Model" on page 7-11
- "Find and Fix CAD Import Issues" on page 7-21

## More About

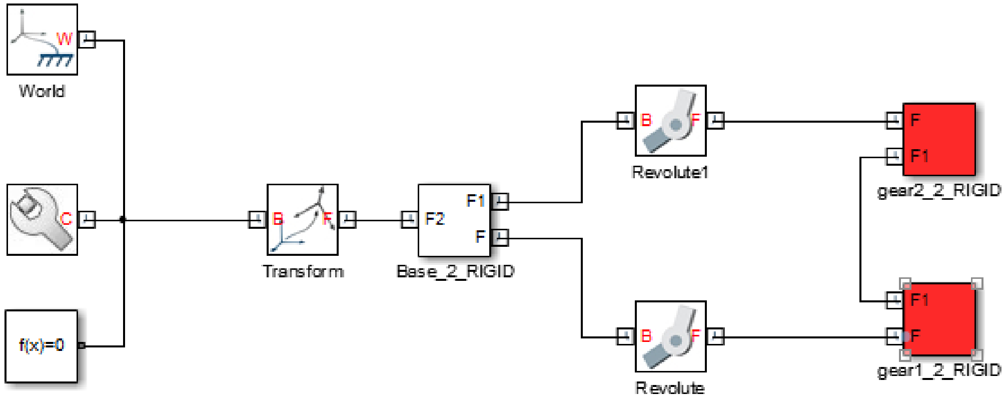
- "CAD Translation" on page 7-2
- "CAD Import" on page 7-5

# Find and Fix CAD Import Issues

## CAD Constraints Replaced with Rigid Connections

SimMechanics supports most, but not all, CAD constraints. If you import a CAD assembly with a CAD constraint that SimMechanics does not support, SimMechanics issues a warning message and automatically replaces that constraint with a rigid connection.

The figure shows the imported model of a CAD assembly that contains an unsupported gear constraint. Because SimMechanics does not support that particular gear constraint, it replaces it with a frame line. The frame line represents a rigid connection.



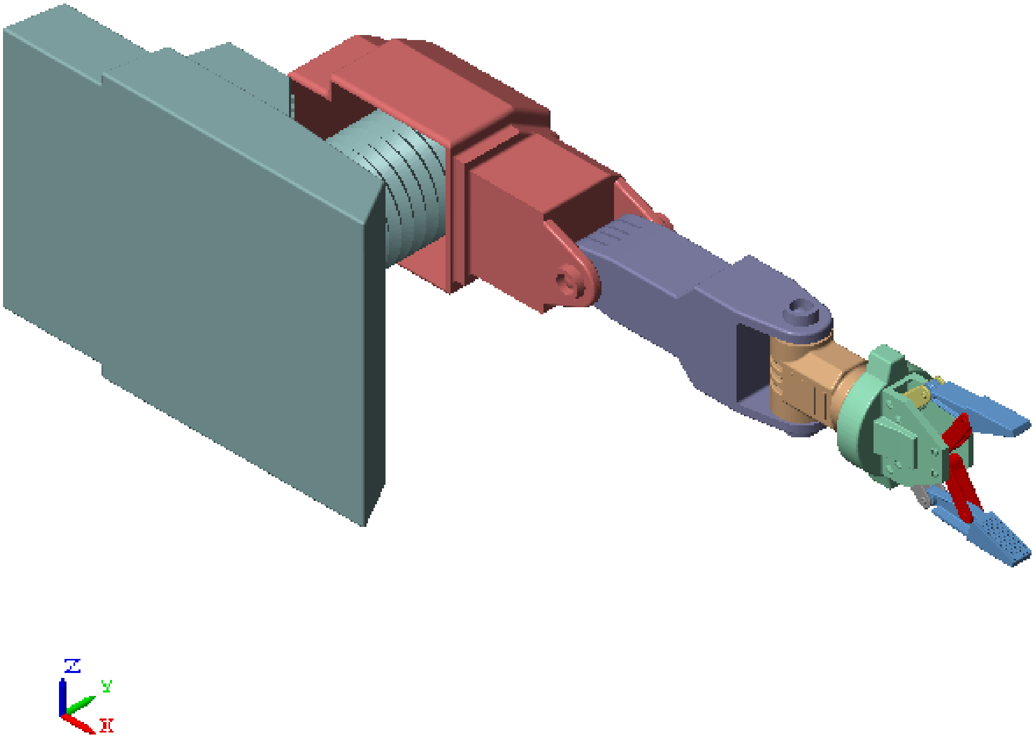
### Identify and Change Automatic Rigid Connections

The warning message identifies the blocks and ports that connect to the unsupported constraint. Use this information to identify the new rigid connection in the model.

Then, determine if any combination of SimMechanics joint, gear, or constraint blocks adequately replaces the unsupported constraint. If so, replace that rigid connection. Run the simulation to check that the model behaves as you expect.

## Model Showing Sideways in Mechanics Explorer

By default, Mechanics Explorer displays a model with the Z axis of the World frame pointing up. Using this convention, the default gravity vector `[0 0 -9.81] m/s^2` points down, a direction that is practical for most applications. However, this convention differs from that which CAD platforms commonly use, Y axis up, causing Mechanics Explorer to display some models sideways. If this happens, you can manually change the view convention to that used in the original CAD assembly. The figure shows the default Mechanics Explorer display of an imported robot arm model.



### Change View Convention

To change the view convention of a model:

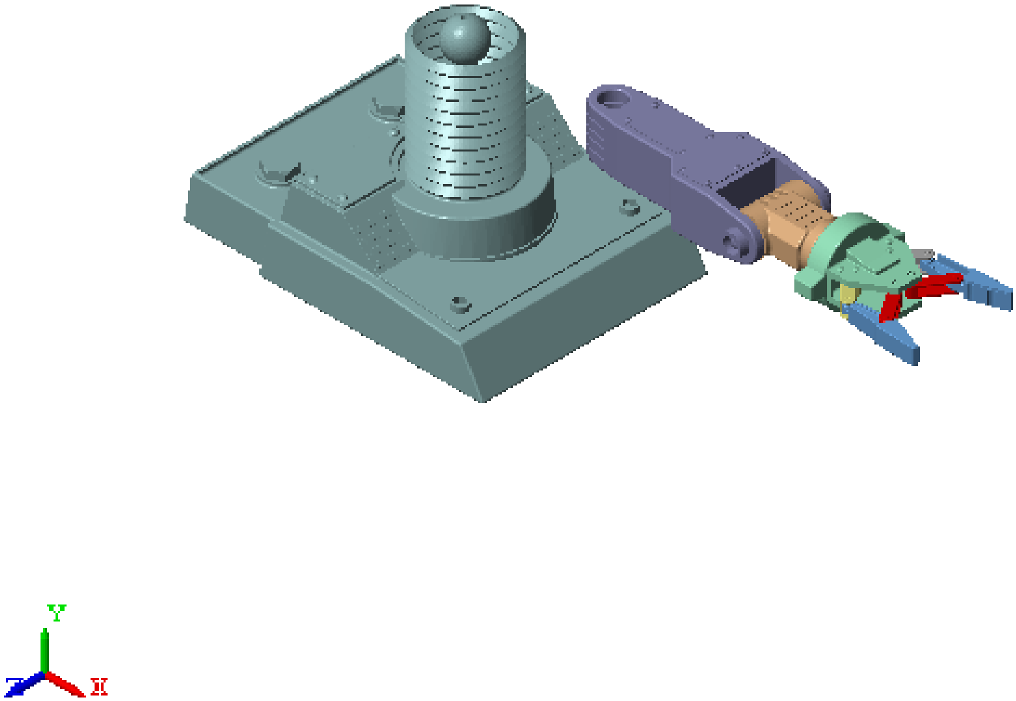
1. In the Mechanics Explorer toolbar, click the **View Convention** drop-down menu.
2. Select **Y up (ZX Top)**.
3. Refresh the Mechanics Explorer display by selecting a view point from the Mechanics Explorer tool bar.

Mechanics Explorer displays the model using the new view convention.

## Parts Not Showing in Mechanics Explorer

During CAD import, SimMechanics uses a set of geometry files in STEP or STL format to generate the 3-D surface geometry of each CAD part. If SimMechanics cannot load the geometry file for a part, that part appears invisible in Mechanics Explorer. This issue does not affect model update or simulation.

The figure shows the Mechanics Explorer display of an imported model containing an invalid geometry file.



### Correct Visualization Issue

If a part of an imported model appears invisible in Mechanics Explorer:

1. In Mechanics Explorer, identify the name of each invisible part.
2. In the block diagram, open the dialog boxes of the associated Solid blocks.
3. In the **Geometry** section, check that the name and location of the geometry files are correct.

   If either is incorrect, enter the correct information and update the model. Check that Mechanics Explorer displays the invisible part. If not, check if the geometry files are valid.

### Geometry File Issues

To visualize a CAD assembly that you import, SimMechanics relies on a set of geometry files that specify the 3-D surface geometry of the CAD parts. Each geometry file specifies the surface geometry of one CAD part as a set of 2-D triangles. To do this, the geometry files contain:

- [X Y Z] coordinates of the triangle vertices
- [X Y Z] components of the normal vectors for the triangles.

If a geometry file specifies a normal vector with zero length, SimMechanics issues a warning. The geometry file fails to load.

## Colored Parts Showing Gray in Mechanics Explorer

To import the color of a CAD part into SimMechanics, the color must be specified at the part level. Colors specified at the feature level, such as individually colored surfaces, or assembly level do not carry over into SimMechanics. This can cause parts to appear gray —the default solid color in SimMechanics—during model visualization. To correct this issue, you must modify the CAD part file directly to ensure that it contains a single color applied to the entire part.

## See Also

`smimport`

## Related Examples

- “Install and Register SimMechanics Link Software” on page 7-9

- “Import Robot Arm Model” on page 7-11
- “Import Stewart Platform Model” on page 7-16

## More About

- “CAD Translation” on page 7-2
- “CAD Import” on page 7-5

# Deployment

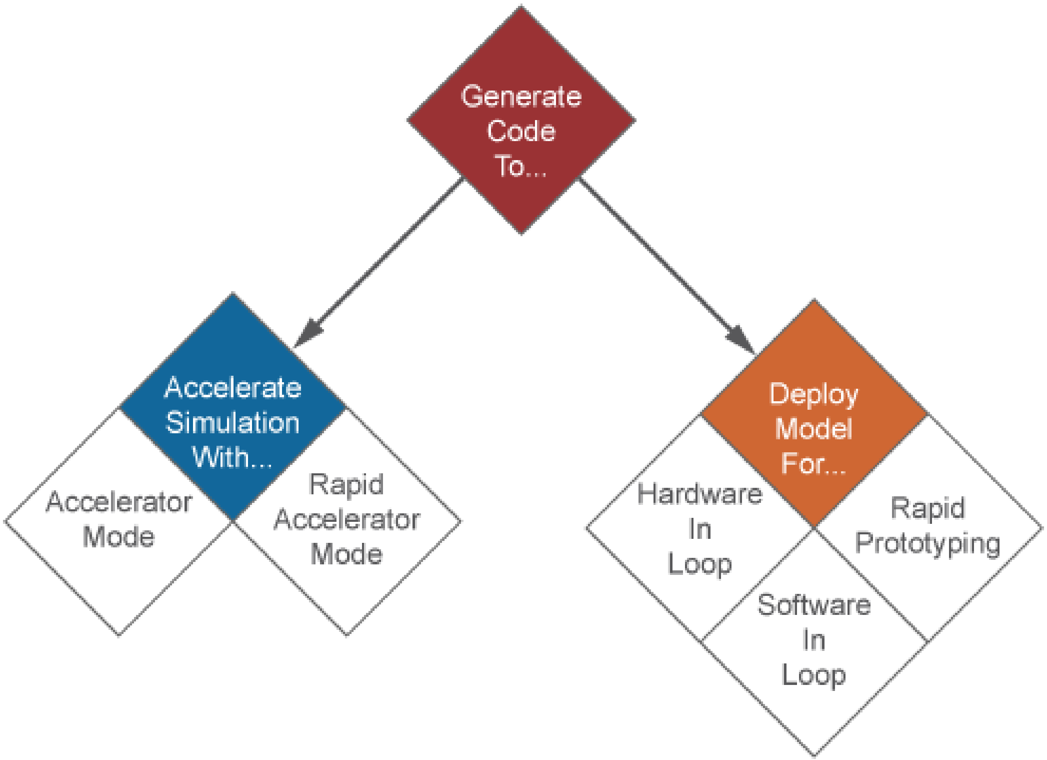
8

# Code Generation

# About Code Generation

| In this section... |
|---|
| "Simulation Accelerator Modes" on page 8-2 |
| "Model Deployment" on page 8-3 |

SimMechanics supports code generation with Simulink Coder™. You can generate C/C++ code from a SimMechanics model to accelerate simulation or to deploy a model.



## Simulation Accelerator Modes

Simulink can generate C/C++ executable code to shorten simulation time. Two simulation modes generate code:

- Accelerator
- Rapid Accelerator

SimMechanics 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 SimMechanics 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 SimMechanics 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:** SimMechanics supports, but does not perform, code generation for model deployment. Code generation for model deployment requires the Simulink Coder product.

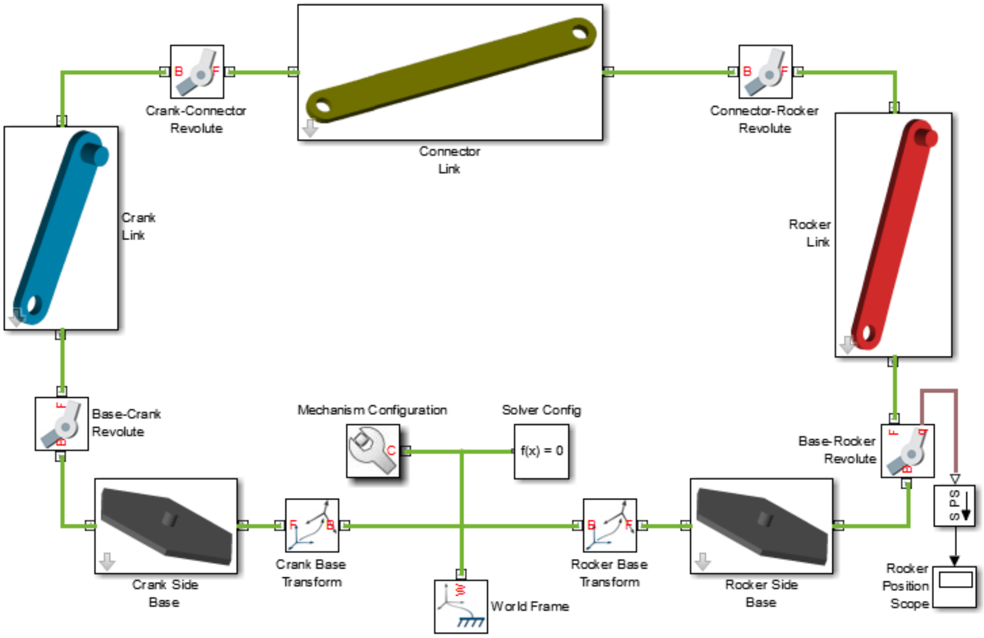
## Related Examples

- "Configure Four-Bar Model for Code Generation" on page 8-4
- "Configure Model for Rapid Accelerator Mode" on page 8-6
- "Find and Fix Code Generation Issues" on page 8-9

# Configure Four-Bar Model for Code Generation

You can generate code from a SimMechanics model for deployment outside the MATLAB environment. This example shows how to configure a four-bar model for code generation using a variable-step solver with the objective of execution efficiency. The example uses the default Simulink solver `ode45 (Dormand-Prince)`.

The four-bar model is present in your SimMechanics installation. To open the model, at the MATLAB command line type `sm_four_bar`. A new Simulink Editor window opens with the block diagram of the four-bar model.



## Configure Model

To configure the model for code generation:

1 In the Simulink Editor window for your model, select **Simulation** > **Model Configuration Parameters**.

2. In the **Model Configuration Parameters** dialog box, select **Code Generation**.
3. In **Target Selection**, enter `rsim.tlc`.

   ---

   **Note:** You must use the `rsim.tlc` target each time you use a variable-step solver. You can change the solver type in the **Solver** section of the **Model Configuration Parameters** window.

   ---
4. In **Code Generation Advisor**, select `Execution Efficiency`.
5. Click **Apply**.
6. To generate C code for your model, click **Build**.

## Related Examples

- “Configure Model for Rapid Accelerator Mode” on page 8-6
- “Find and Fix Code Generation Issues” on page 8-9

## More About

- “About Code Generation” on page 8-2

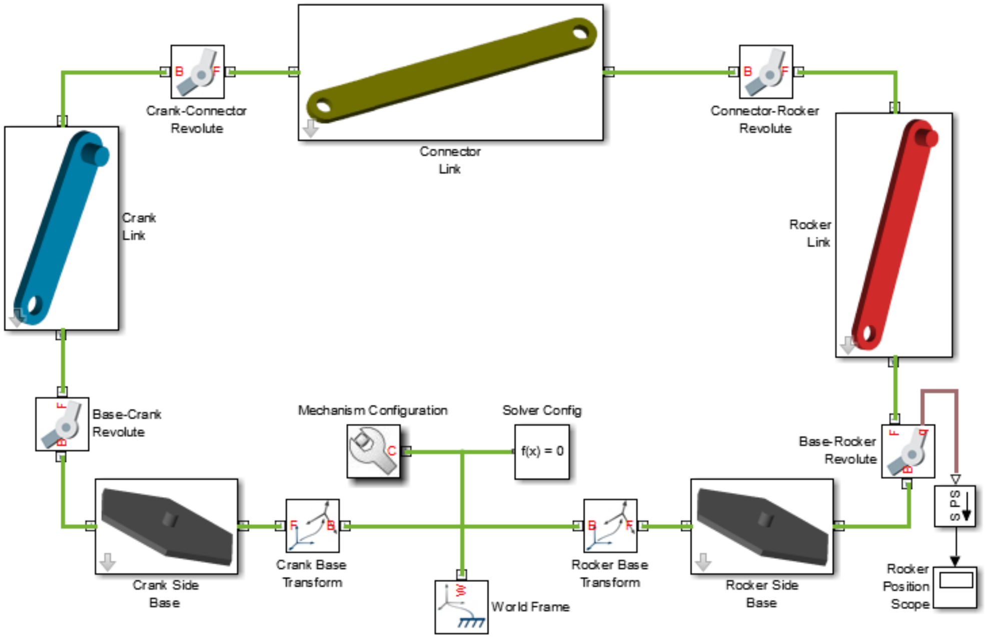
# Configure Model for Rapid Accelerator Mode

**In this section...**

## Model Overview

You can run a SimMechanics model in Accelerator and Rapid Accelerator modes. When you select an accelerator mode, SimMechanics generates executable code that accelerates the model simulation. This example shows how to configure a four-bar model for Rapid Accelerator simulation mode. The simulation uses the default Simulink solver `ode45 (Dormand-Prince)`.

The four-bar model is present in your SimMechanics installation. To open the model, at the MATLAB command line type `sm_four_bar`. A new Simulink Editor window opens with the block diagram of the four-bar model.



## Configure Model

To configure the model for Rapid Acceleration simulation mode, follow these steps:

1. In the Simulink Editor window for your model, select **Simulation**.
2. In the drop-down menu, select **Mode** > **Rapid Accelerator**.
3. Select **Simulation** > **Model Configuration Parameters**.
4. In **Code Generation**, under **System target file**, enter `rsim.tlc`.

   **Note:** You must use the `rsim.tlc` target each time you generate code with a variable-step solver. Both Accelerator and Rapid Accelerator modes generate executable code that requires the `rsim.tlc` target to be used with variable-step solvers.

5. Expand the **SimMechanics 2G** node.
6. Select **Explorer**.
7. Clear the **Open Mechanics Explorer on model update or simulation** check box.

   **Note:** Clearing the **Open Mechanics Explorer on model update or simulation** check box disables visualization with Mechanics Explorer. Disabling visualization prevents SimMechanics from issuing a warning message when you simulate a model in Accelerator or Rapid Accelerator mode.

8. Press **Ctrl+T** to simulate the model.

   **Note:** The Rapid Accelerator mode incurs an initial time cost to generate the executable code. Once the code is generated, the simulation proceeds more rapidly. Rapid Accelerator mode is suggested for large or complex SimMechanics models with long simulation times.

The Rapid Accelerator mode does not support visualization. Mechanics Explorer does not open, and you cannot view a dynamic simulation of the model. All other simulation capabilities remain functional, including graphics and scopes.

## Related Examples

- "Configure Four-Bar Model for Code Generation" on page 8-4

- “Find and Fix Code Generation Issues” on page 8-9

## More About

- “About Code Generation” on page 8-2

# Find and Fix Code Generation Issues

**In this section...**

SimMechanics supports code generation using Simulink Coder. However, certain guidelines and limitations apply. These include:

- Variable step Simulink solver requires `rsim.tlc` target.
- Simulink solver must be continuous.
- SimMechanics does not support visualization in accelerator mode.
- SimMechanics does not support run-time parameters.

**Note:** To generate code for a SimMechanics model, you must have an active Simulink Coder installation.

## Variable step Simulink solver requires `rsim.tlc` target

Code generation is compatible with fixed- and variable-step solvers. If you select a variable-step solver, you must specify system target file `rsim.tlc`. To specify the `rsim.tlc` system target file, follow these steps:

1. In the Simulink Editor window for your model, select **Simulation** > **Model Configuration Parameters**.
2. In the left pane of the **Model Configuration Parameters** dialog box, select **Code Generation**.
3. In **System target file**, enter `rsim.tlc`.
4. Click **Apply**.
5. Click **Build** to generate code for the active model.

## Simulink solver must be continuous

Both fixed- and variable-step solvers can be continuous or discrete. Generating code from a SimMechanics model requires a continuous solver. SimMechanics blocks use continuous time samples, and are incompatible with discrete solvers. If you attempt to generate code with a discrete solver, Simulink Coder issues an error.

If you receive an error stating that SimMechanics does not support a discrete solver, select a continuous Simulink solver. To change the Simulink solver, follow these steps:

1. In the Simulink Editor window for your model, select **Simulation** > **Model Configuration Parameters**.
2. In **Solver**, under **Solver options**, click **Solver**.
3. In the drop-down menu, select any solver with the exception of `discrete (no continuous states)`.

## SimMechanics does not support visualization in accelerator mode

SimMechanics supports Accelerator and Rapid Accelerator simulation modes. Selecting an accelerator mode generates executable code that shortens the time required to run a simulation. However, the simulation produces no visualization output. Mechanics Explorer does not open, and you cannot visualize the model simulation. To restore visualization, select the Normal simulation mode.

If you simulate a model in Accelerator or Rapid Accelerator mode, SimMechanics issues a warning indicating that accelerator modes do not support visualization. To remove the warning, disable visualization with Mechanics Explorer:

1. In the Simulink Editor window for your model, select **Simulation** > **Model Configuration Parameters**.
2. In the **Model Configuration Parameters** window, expand the **SimMechanics 2G** node.
3. Select **Explorer**.
4. Clear the **Open Mechanics Explorer on model update or simulation** check box.

**Note:** Clearing the **Open Mechanics Explorer on model update or simulation** check box disables Mechanics Explorer. When you return to Normal simulation mode, check the box to restore visualization with Mechanics Explorer.

## SimMechanics Does Not Support Run-Time Parameters

Model parameters are fixed during code generation. To change model parameters, edit the parameters in SimMechanics and regenerate code for the model. You can only change model parameters in SimMechanics itself.

## Related Examples

- "Configure Four-Bar Model for Code Generation" on page 8-4
- "Configure Model for Rapid Accelerator Mode" on page 8-6

## More About

- "About Code Generation" on page 8-2