Getting Started with the MapleSim Connector

Copyright © Maplesoft, a division of Waterloo Maple Inc. 2026

Getting Started with the MapleSim Connector

Copyright

Maplesoft, Maple, and MapleSim are all trademarks of Waterloo Maple Inc.

© Maplesoft, a division of Waterloo Maple Inc. 2009-2026. All rights reserved.
No part of this book may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means — electronic, mechanical, photo-copying, recording, or otherwise. Information in this document is subject to change without notice and does not represent a commitment on the part of the vendor. The software described in this document is furnished under a license agreement and may be used or copied only in accordance with the agreement. It is against the law to copy the software on any medium except as specifically allowed in the agreement.

Macintosh is a trademark of Apple Inc., registered in the U.S. and other countries.

MATLAB and Simulink are registered trademarks of The MathWorks, Inc.

Microsoft, Windows, and Visual Studio are registered trademarks of Microsoft Corporation.

Linux is a registered trademark of Linus Torvalds.

All other trademarks are the property of their respective owners.

This document was produced using Maple and DocBook.

Contents

Introduction iv
1 Getting Started 1
1.1 Getting Help 1
1.2 Using the Simulink® Component Block Generation App 1
Subsystem Preparation 1
Subsystem Selection 1
Inputs, Outputs, and Parameter Configuration 1
Code Export Options 3
Export 6
View Code 6
1.3 Viewing MapleSim Connector Examples 7
1.4 Example: RLC Circuit Model 7
1.5 Enabling Extended Visualization Data 9
1.6 Preparing a Model for Export 10
Converting the Model to a Subsystem 10
Defining Subsystem Inputs and Outputs 11
Define and Assign Subsystem Parameters 14
Exporting Your Model Using the Simulink® Component Block Generation App 16
Implement the S-Function Block in Simulink® 16
2 Creating and Exporting Mathematical Models in Maple 18
2.1 Creating and Exporting a DynamicSystems Object Programmatically 18
2.2 Example: DC Motor 19
Index 22

Introduction

The MapleSim™ Connector provides all of the tools you need to prepare and export your dynamic systems models to Simulink® as S-function blocks. You can create a model in MapleSim, simplify it in Maple™ by using an extensive range of analytical tools, and then generate an S-function block that you can incorporate into your Simulink® toolchain.

You can also use these tools for exporting mathematical models that you have created from first principles in Maple as S-functions.

Furthermore, various options allow you to use the C code generation feature in Maple to create code libraries of your MapleSim models for implementation in other applications.

Features of this toolbox include:

- Maple templates, which provide an intuitive user interface for optimizing your MapleSim model, and then generate an S-function in Simulink®.
- A range of examples illustrating how to prepare and export your models.
- A direct interface between Maple and Simulink® allows you to generate and test an S-function block as you develop the model.
- Commands for developing S-functions of mathematical models from first principles in the Maple environment and examples to illustrate how to do it.
- Access to commands in the **MapleSimConnector** and **DynamicSystems** packages in Maple for developing automated applications to generate S-functions.

Scope of Model Support

MapleSim is a very comprehensive modeling tool where it is possible to create models that could go beyond the scope of this MapleSim Connector release. In general, the MapleSim Connector supports systems of any complexity, including systems of DAEs of any index, in any mix of domains.

Requirements

Requires MATLAB® and Simulink®. For details on supported platforms, visit the Maplesoft System Requirements website at **http://www.maplesoft.com/products/system_requirements.aspx**.

For installation instructions and system requirements, see the **Install.html** file on the product disc.

Adding External Libraries to Your Search Path

You can export a model that uses an external library as part of the model to an S-function block. In order to do this, you **first** need to add the directory that contains the external library file (that is, the .dll or .so file) to your search path. This involves appending the external library directory to either your PATH environment variable (for Windows®) or your LD_LIBRARY_PATH environment variable (for Linux® and Macintosh®).

To add an external library directory to your search path

1. Determine the location of the external library directory.

   Note: This is the directory that contains the .dll file (Windows) or the .so file (Linux or Macintosh) that is used in your model.

2. Add the library directory found in step 1 to the appropriate environment variable for your operating system.
   - For Windows, add the library directory to your PATH environment variable.
   - For Linux and Macintosh, add the library directory to your LD_LIBRARY_PATH environment variable.

Consult the help for your operating system for instructions on how to edit these environment variables.

3. Restart your computer.

1 Getting Started

1.1 Getting Help

Refer to the MapleSimConnector help page in the Maple help system.

1.2 Using the Simulink® Component Block Generation App

The MapleSim Connector provides a **Simulink® Component Block Generation** app in the form of a Maple worksheet for manipulating and exporting MapleSim subsystems. This app contains pre-built embedded components that allow you to generate S-function or C code from a MapleSim subsystem, export the subsystem as a Simulink® block, and save the source code.

Using this app, you can define inputs and outputs for the system, set the level of code optimization, choose the format of the resulting S-function, and generate the source code, library code, block script, or Simulink® block. You can use any Maple commands to perform task analysis, assign model equations to a variable, group inputs and outputs to a single vector and define additional input and output ports for variables.

Note: Code generation can handle all systems modeled in MapleSim, including hybrid systems with defined signal input (RealInput) and signal output (RealOutput) ports.

The S-Function Block Generation consists of the following steps:

- Subsystem Preparation
- Subsystem Selection
- Port and Parameter Management
- S-Function Options
- Generate S-Function
- View S-Function

Subsystem Preparation

Convert your model or part of your model into a subsystem. This identifies the set of modeling components that you want to export as a block component. Since Simulink® only supports data signals, properties on acausal connectors such as mechanical flanges and electrical pins, must be converted to signals using the appropriate ports.

To connect a subsystem to modeling components outside of its boundary, you add subsystem ports. A subsystem port is an extension of a component port in your subsystem. The resulting signals can then be directed as inputs and outputs for MapleSim™ Connector App.

Note: For connectors you must use signal components, since acausal connectors can not be converted to a signal.

By creating a subsystem you not only improve the visual layout of a system in **Model Workspace** but also prepare the model for export. The following examples in this section show you how to group all of the components into a subsystem.

Subsystem Selection

You can select which subsystems from your model you want to export to a Simulink® block. After a subsystem is selected, click **Load Selected Subsystem**. All defined input and output ports are loaded.

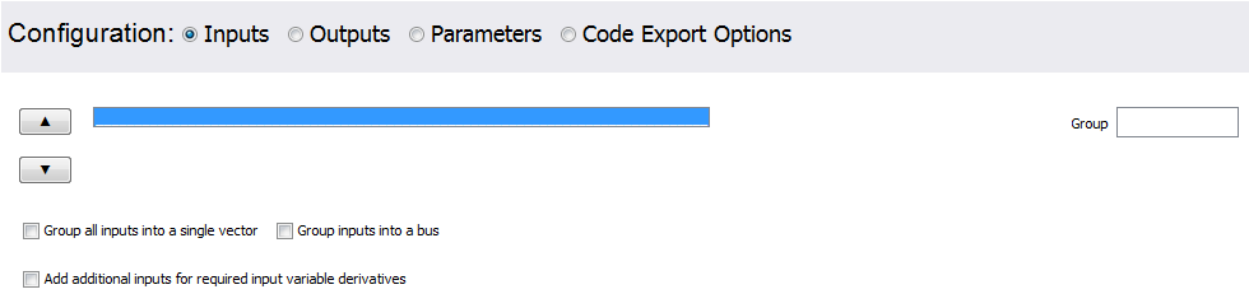
Inputs, Outputs, and Parameter Configuration

The **Configuration** section lets you customize, define and assign parameter values to specific ports. Subsystem components to which you assign the parameter inherit a parameter value defined at the subsystem level. After the subsystem

is loaded you can group individual input and output variable elements into a vector array, and add additional input and output ports for customized parameter values. Input ports can include variable derivatives, and output ports can include subsystem state variables.

Note: If the parameters are not marked for export they will be numerically substituted.

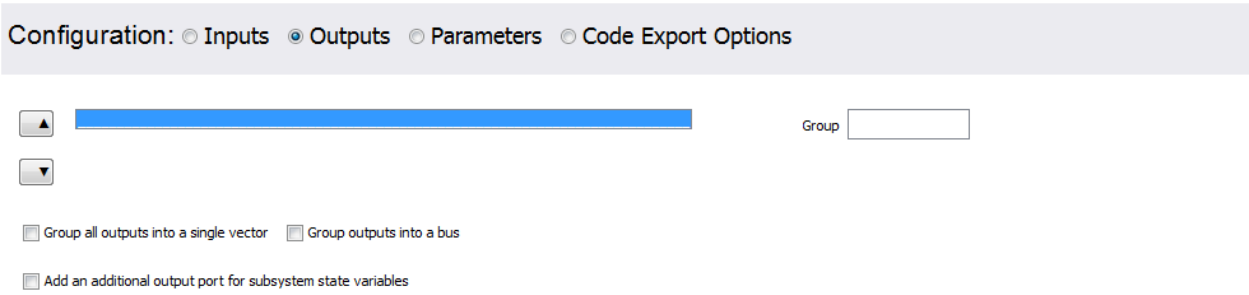
The following selections specify the input ports, output ports, and states for generating Simulink® blocks.



Select **Group all inputs into a single vector** to create a single 'vector' input port for all of the input signals instead of individual ports. The order of the inputs are the same as given in the S-function mask window.

Select **Group inputs into a bus** to have the S-function accept a bus signal as input.

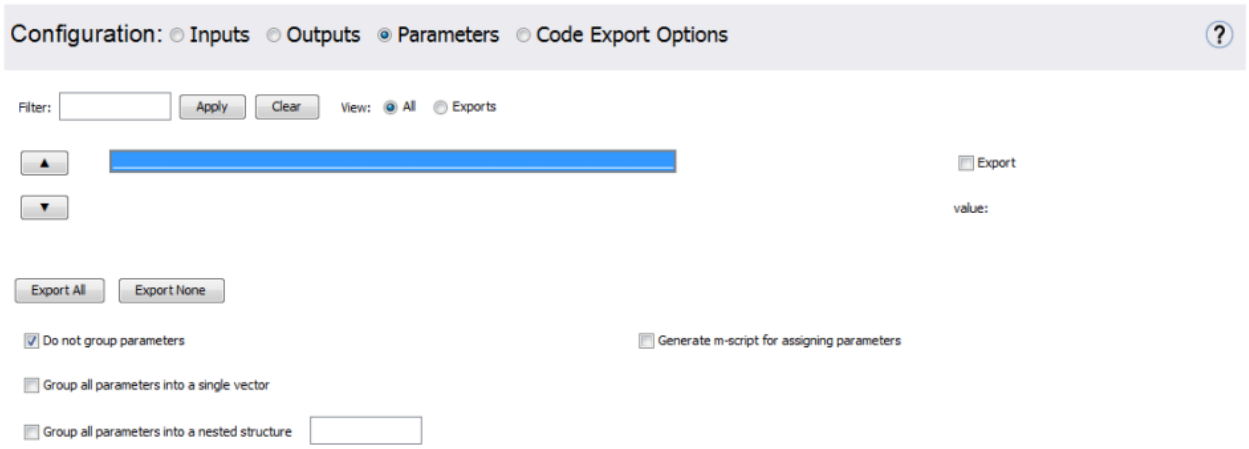
Select **Add additional inputs for required input variable derivatives** to specify calculated derivative values instead of numerical approximations.



Select **Group all outputs into a single vector** to define outputs as an S-Function 'mask'.

Select **Group outputs into a bus** to have the S-function return a bus signal output.

Select **Add an additional output port for subsystem state variables** to add extra output ports for the state variables.



Select **Group all parameters into a single vector** to create a single parameter 'vector' for all of the parameters in the S-function. If not selected, the S-function mask will contain one parameter input box for each of the S-function parameters.

Select **Generate m-script for assigning parameters** to generate an initialization m-file with the system parameters.

Select **Group all parameters into a nested structure** to specify the parameters as a nested MATLAB structure. Use the textbox to define the name of the top-level parameter structure. Selecting this option will automatically disable the **Generate m-script for assigning parameters** option.

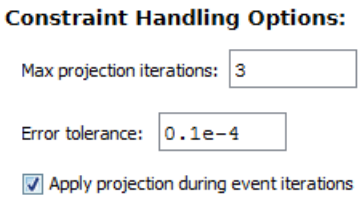
Select **Export** to have selected parameters exported.

Code Export Options

The **Code Export Options** settings in the **Configuration** section specify the advanced options for the code generation process.

Constraint Handling Options

The **Constraint Handling Options** area specifies whether the constraints are satisfied in a DAE system by using constraint projection in the generated Simulink® block. Use this option to improve the accuracy of a DAE system that has constraints. If the constraint is not satisfied, the system result may deviate from the actual solution and could lead to an increase in error at an exponential rate.



Set the **Max projection iterations** to specify the maximum number of times that a projection is permitted to iterate to obtain a more accurate solution.

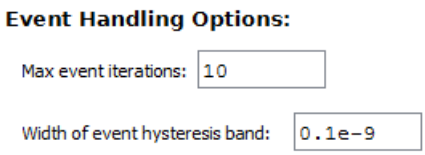
Set the **Error tolerance** to specify the desirable error tolerance to achieve after the projection.

Select **Apply projection during event iterations** to interpolate iterations to obtain a more accurate solution.

Constraint projection is performed using the **constraint projection** routine in the External Model Interface as described on The MathWorks™ web site to control the drift in the result of the DAE system.

Event Handling Options

The **Event Handling Options** area specifies whether the events are satisfied in a DAE system by using event projection in the generated Simulink® block. Use this option to improve the accuracy of a DAE system with events. If the constraint is not satisfied, the system result may deviate from the actual solution and could lead to an increase in error at an exponential rate.

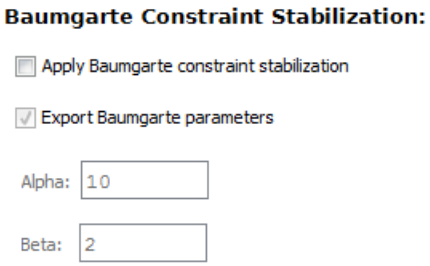


Set the **Max event iterations** to specify the maximum number of times that a projection is permitted to iterate to obtain a more accurate solution.

Set the **Width of event hysteresis band** to specify the desirable error tolerance to achieve after the projection.

Baumgarte Constraint Stabilization

The Baumgarte constraint stabilization method stabilizes the position constraint equations, by combining the position, velocity, and acceleration constraints into a single expression. By integrating the linear equation in terms of the acceleration, the Baumgarte parameters, alpha and beta, act to stabilize the constraints at the position level.



Apply Baumgarte constraint stabilization: Apply the Baumgarte constraint stabilization.

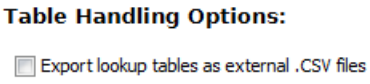
Export Baumgarte parameters: Add **Alpha** and **Beta** as parameters in the generated code.

Alpha: Set the derivative gain for Baumgarte constraint stabilization.

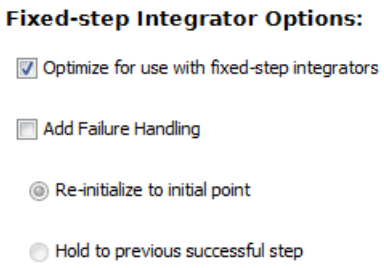
Beta: Set the proportional gain for Baumgarte constraint stabilization.

Table Handling Options

Select **Export lookup tables as external .CSV files** to generate a comma-separated values (CSV) file with the values for your lookup tables. You can then change the values in your lookup table without having to recompile your code. The updated values are read at run-time.



Fixed-Step Integrator Options



Select **Optimize for use with fixed-step integrators** to optimize events handling and relax the inconsistencies detection tolerance.
Select **Add Failure Handling** to reinitialize the model on failure, and then select one of the following re-initialization options:

- **Re-initialize to initial point:** Reinitialize the model to the initial values when a failure happens.
- **Hold to previous successful step:** Reinitialize the model to the values at the end of the last successful step.

Note: Failure handling is only available for Simulink® fixed-step integration.

Discretization

Select **Export as a discrete model (no continuous states)** to apply discretization to your model. When selected, you can select a solver type from one of the following options:

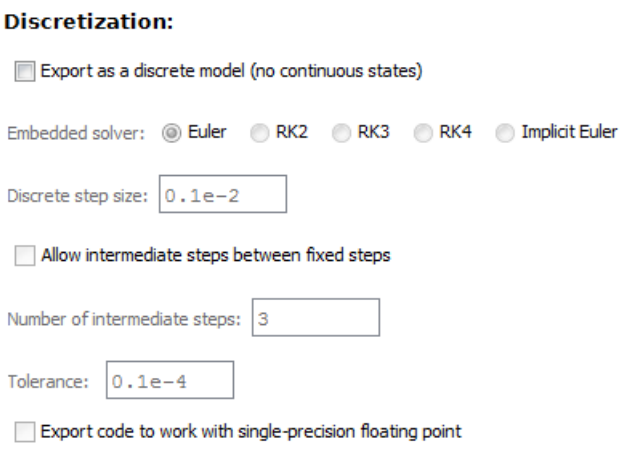
- **Euler**: *forward Euler* method
- **RK2**: *second-order Runge-Kutta* method
- **RK3**: *third-order Runge-Kutta* method
- **RK4**: *fourth-order Runge-Kutta* method
- **Implicit Euler**: *implicit Euler* method

In this section, you can also set the **Discrete step size** (in seconds) for the discretization.

Select **Allow intermediate steps between fixed steps** if you want to trigger events inside a fixed step (instead of at the end of the fixed step). Your solver must be either *Euler* or *Implicit Euler*. After selecting this option, enter the following settings:

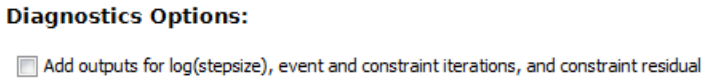
- **Number of intermediate steps:** The maximum number of intermediate steps you want to take inside the fixed step.
- **Tolerance:** The proportion of the current step size that the solver is allowed to overshoot so as to catch more than one close-firing event. Generally, enter a smaller tolerance for models that are fairly linear between steps and a larger tolerance for solutions that are far from linear.
- **Export code to work with single-precision floating point:** Select this option to have internal calculations done in single-precision rather than double-precision. Further, some of the criteria for MapleSim solvers are adjusted to work with the lower precision.

Note: You will need to configure Simulink to properly run a single-precision S-Function.

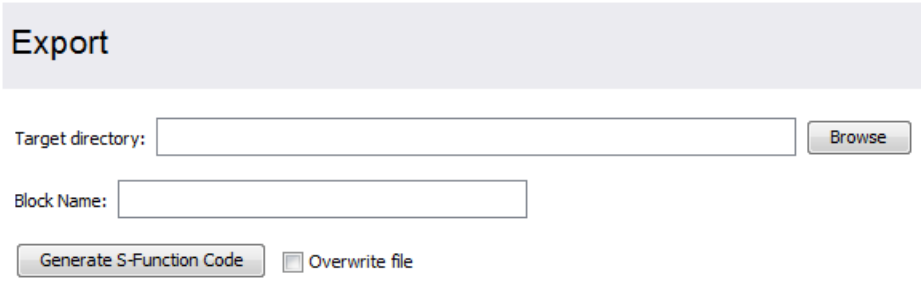


Diagnostic Options

Select **Add outputs for log(stepsize), event and constraint iterations, and constraint residual** to provide run-time diagnostics in the generated Simulink® block. This adds four outputs to the bottom of the Simulink® block: **LogStep-Size(t)**, **EventIterations(t)**, **ConstraintIterations(t)**, and **ConstraintResidual(t)**.



Export



Provide a name, specify the location for the generated file and click **Generate S-Function Code**.

Note: If your model contains an external library, then you must add the directory that contains the external library to your search path. See *Adding External Libraries to Your Search Path (page iv)* for instructions on how to do this.

View Code

After you generate the S-Function code and run the block a MATLAB® command window opens and the block with any of the following specified parameters is generated in Simulink®:

- Block Generation Script
- C Code
- Parameter Script

1.3 Viewing MapleSim Connector Examples

Toolbox examples are available in MapleSim.

To view an example:

1. From the **Help** menu, select the **Examples > MapleSim Connector Examples** menu, and then click the entry for the model that you want to view.

Note: Some models include additional documents, such as templates that display model equations or define custom components.

2. In the **Attached Files** tab (), expand **Documents**. You can open any of these documents by right-clicking (**Control**-clicking for Mac) its entry in the list and selecting **View**. After you add a template to a model, it will be available from this list.

1.4 Example: RLC Circuit Model

In this example, you will generate a Simulink® block from an RLC circuit model that was created in MapleSim.

To generate an S-function block

1. From the **Help** menu, select **Examples > MapleSim Connector Examples**, and then select the **RLC Parallel Circuit**.

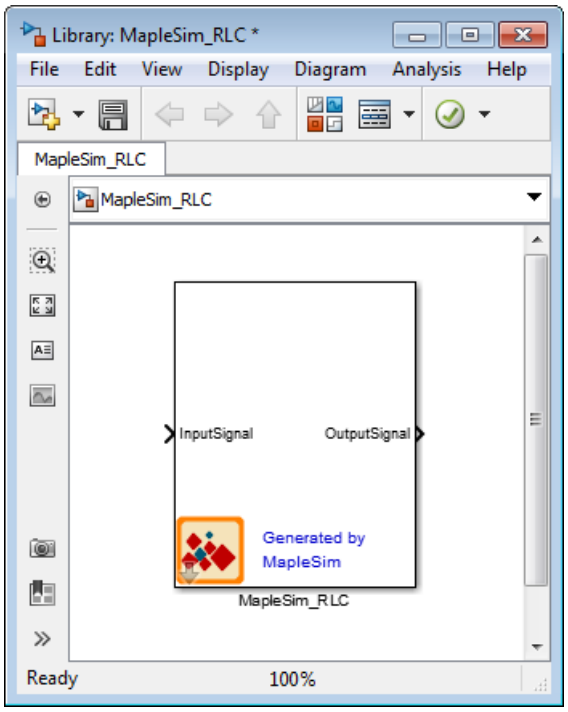
2. Select the **Add Apps or Templates** tab ().
3. Double-click on the **Simulink® Component Block Generation** entry under Simulink in the **Apps** palette. The Analysis window opens with the **Simulink® Component Block Generation** app loaded in the **Apps** tab.
4. In the **Subsystem Selection** section, select **Main > RLC**.
5. Click **Load Selected Subsystem**. All of the fields are populated with information specific to the RLC subsystem.

Note: By default, all parameters in the model are kept as configurable parameters.

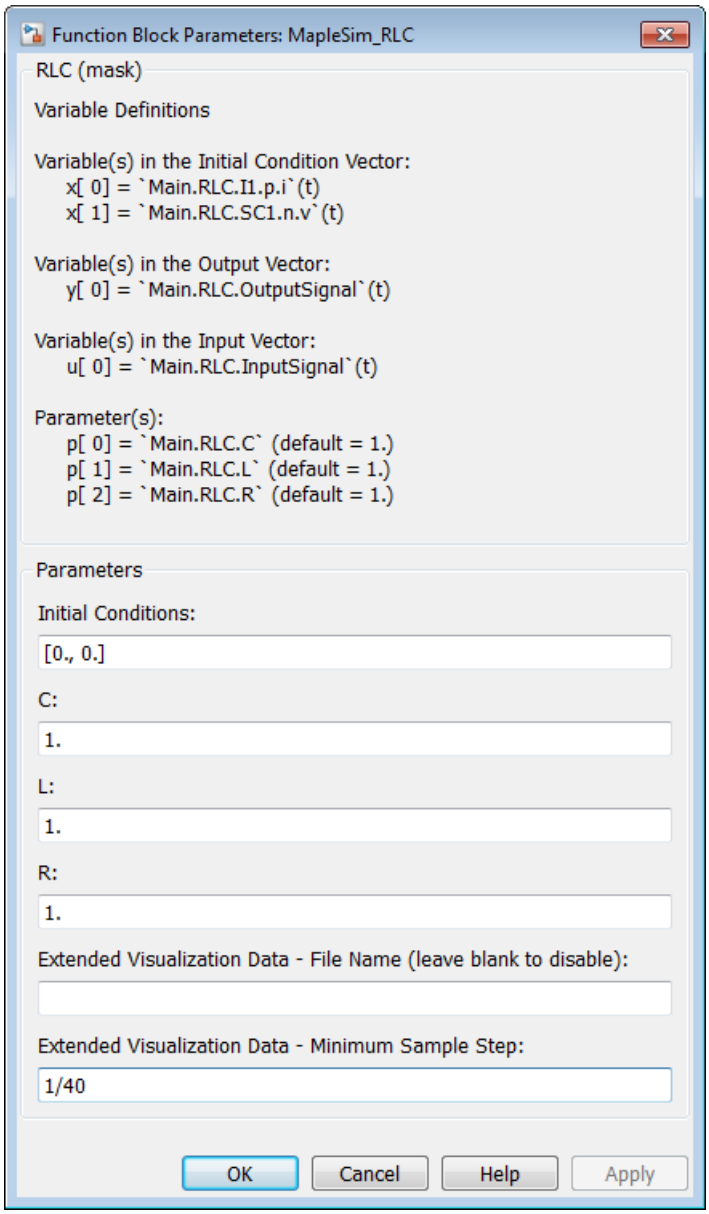
6. In the **Export** section, specify the directory to save the generated files to.
7. Click **Generate S-Function Code** to generate the S-function code and create the block.

Note: Generating a block may require a few minutes.

8. Open the .m file in MATLAB® and run the code to open the block in Simulink®.



Double-click on the block to open the **Parameter Mask** dialog box that contains the symbolic parameters from the original model (see the following figure).



You can set parameters for the block from this mask. For 2016.2 and later versions of the MapleSim Connector, the two *Extended Visualization Data* parameters are used to create data files that can be imported back into MapleSim. This lets you see view the results from executing the block in MapleSim (see the following section, **Enabling Extended Visualization Data**, for details).

This block can now be connected with any compatible Simulink® blocks.

1.5 Enabling Extended Visualization Data

For 2016.2 and later versions of the MapleSim Connector, there are two **Extended Visualization Data** parameters in the **Parameter Mask** dialog box for the generated block. These extended visualization parameters are used to generate simulation results (that is, plots and 3-D animations) that can be imported back into MapleSim. To generate extended visualization data, enter the following parameters in the **Parameter Mask** dialog.

- **Extended Visualization Data - File Name:** Enter a file name with a *.bin* file extension for the extended visualization data file.The extended visualization data file is necessary in order to import the Simulink® simulation results into MapleSim. Leave this field blank if you do not want to generate extended visualization data.
- **Extended Visualization Data - Minimum Sample Step:** Enter the minimum sample step, in seconds, for the visualization data in the *.bin* file. The default value is 1/40 of a second. This sample step is used for both plots and 3-D animations (for 3-D models). For example, a sample step of 1/40 of a second generates plot points every 1/40 of a second and gives you a frame rate of 40 frames per second in your 3-D animation. Smaller values for the minimum step size give higher resolution plots and animations, but may result in larger *.bin* files. Set this to 0 to store all the visualization data generated by Simulink®.

After you execute your block in Simulink®, the extended visualization data file is created in the same directory that your Simulink® model is in. See **MapleSim > Using MapleSim > Simulating a Model > Importing FMU or Simulink(R) S-function Results** in the MapleSim help system for information on how to import these results into MapleSim.

Note: For best results, set the solver type and error tolerances settings in Simulink® to match the settings you have in the original MapleSim model.

1.6 Preparing a Model for Export

In this example, you will perform the steps required to prepare a slider-crank mechanism model and export it as an S-function block:

1. Convert the slider-crank mechanism model to a subsystem.

2. Define subsystem inputs and outputs.

3. Define and assign subsystem parameters.

4. Export the model using the Simulink® Component Block Generation app.

5. Implement the S-function block in Simulink®.

Note: The following tutorial will take you through these steps in detail. Before starting this tutorial, you must set up MATLAB® and the mex compiler. For more information, see the MapleSimConnector,setup help page for more information.

To open the slider-crank mechanism example

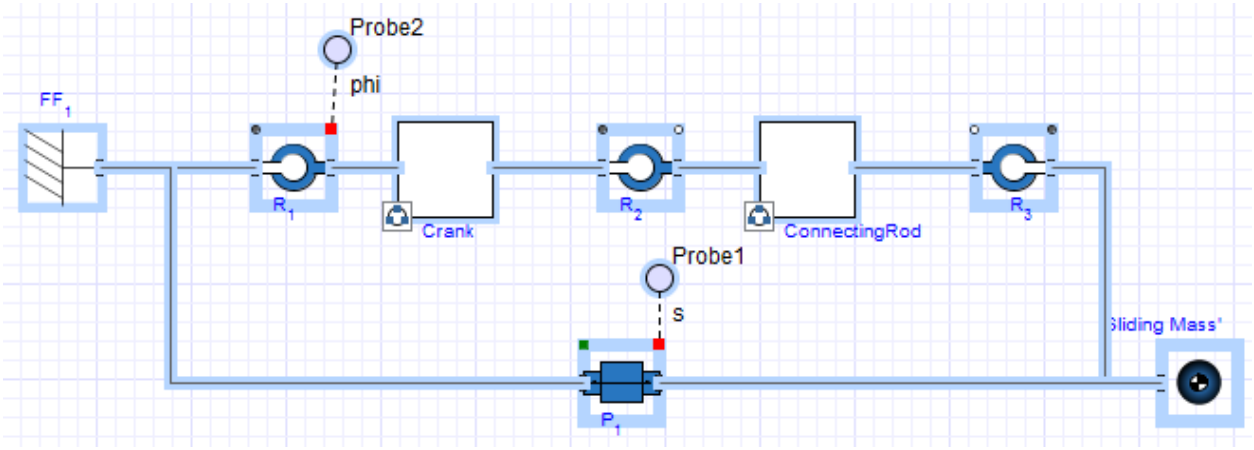
1. In MapleSim, click the **Help** menu item.
2. Select **Examples > User's Guide Examples > Chapter 6**, and then select **Planar Slider-Crank Mechanism**.

Converting the Model to a Subsystem

By converting your entire model or part of your model into a subsystem, you identify which parts of the model that you want to export. In this example, you will prepare the system for export by grouping all of the components into a subsystem.

To convert the model to a subsystem:

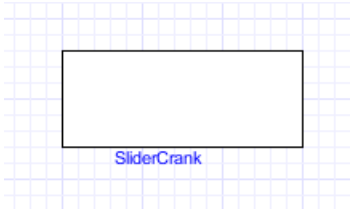
1. Draw a box around all of the components in the model by dragging your mouse over them.



2. From the **Edit** menu, select **Create Subsystem**. The **Create Subsystem** dialog box appears.

3. Enter **SliderCrank** as the subsystem name.

4. Click **OK**. A **SliderCrank** subsystem block appears in the **Model Workspace**.



Defining Subsystem Inputs and Outputs

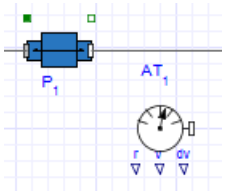
MapleSim uses a topological representation to connect interrelated components without having to consider how signals flow between them, whereas traditional signal-flow modeling tools require explicitly defined system inputs and outputs. Since Simulink® only supports data signals, properties on acausal ports, such as mechanical flanges and electrical pins, must be converted to signals using the appropriate components. The resulting signals are directed as inputs and outputs for the subsystem in MapleSim and for the S-function block.

Note: Currently, code generation is limited to subsystems with defined signal input (*RealInput*) and signal output (*RealOutput*) ports.

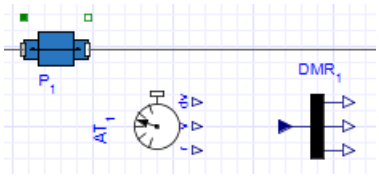
In this example, you will convert the displacements of the slider and the joint between the crank and connecting rod to output signals. The input signal needs to be converted to a torque that is applied to the revolute joint that represents the crank shaft.

To convert the system signals:

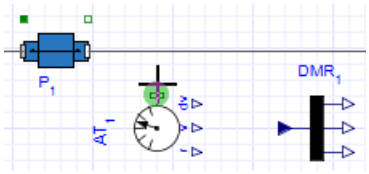
1. Double-click the subsystem block to view its contents. In this view, the broken line indicates the subsystem boundary. You can edit the connection lines and components within the boundary, and add subsystem ports to connect the subsystem to other components.
2. Delete the probes that are attached to the model.
3. In the **Library Components** tab ( ) on the left side of the MapleSim window, expand the **Multibody** palette and then expand the **Sensors** submenu.
4. Drag the **Absolute Translation** component to the **Model Workspace** and place it below the **Prismatic Joint** component.



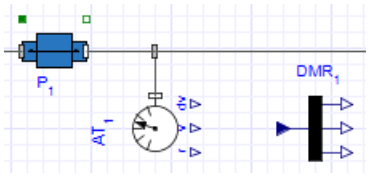
5. Right-click (**Control**-click for Mac) the **Absolute Translation** component and select **Rotate Counterclockwise**.
6. From the **Signal Blocks** > **Routing** > **Demultiplexers** menu, drag a **Real Demultiplexer** component to the **Model Workspace** and place it to the right of the **Absolute Translation** component.



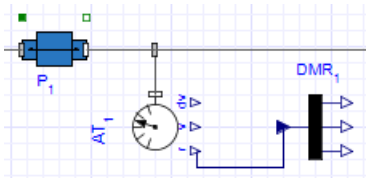
7. To connect the **Absolute Translation** component to the model, click the frame_b connector. The frame is highlighted in green when you hover your pointer over it.



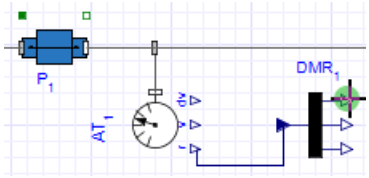
8. Draw a vertical line and click the connection line directly above the component. The sensor is connected to the rest of the diagram.



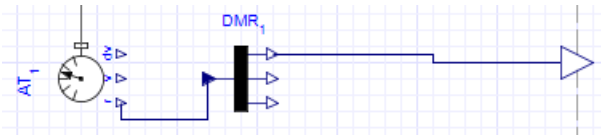
9. In the same way, connect the **r** output port (*TMOutputP*) of the **Absolute Translation** component to the demultiplexer Real input signal (u) port. This is the displacement signal from the sensor in x, y, and z coordinates. Since the slider only moves along the x axis, the first coordinate needs to be an output signal.



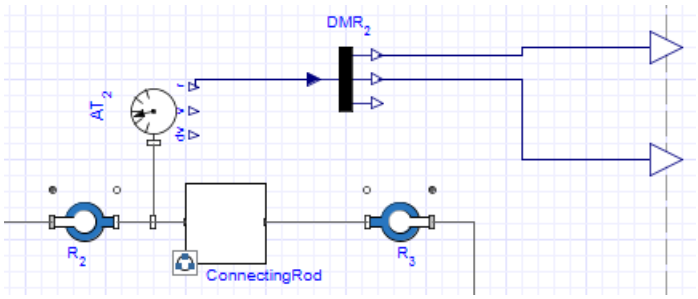
10. Hover your pointer over the first demultiplexer port and click your mouse button once.



11. Drag your pointer to the subsystem boundary and then click the boundary once. A real output port is added to your subsystem.



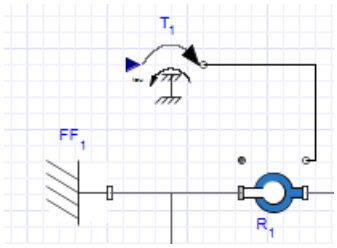
12. Add another **Absolute Translation** component above the **Connecting Rod** subsystem.
13. Right-click (**Control**-click for Mac) the **Absolute Translation** component and select **Flip Vertical**. Right-click the **Absolute Translation** component again and select **Rotate Clockwise**.
14. Add a **Real Demultiplexer** component to the right of the sensor and connect the components as shown below.



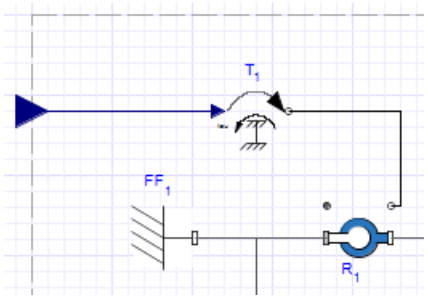
Note: Since the crank is moving in the x-y plane, you only need to output the first two signals.

You will now add a real input port to your subsystem to control the torque on the crank shaft.

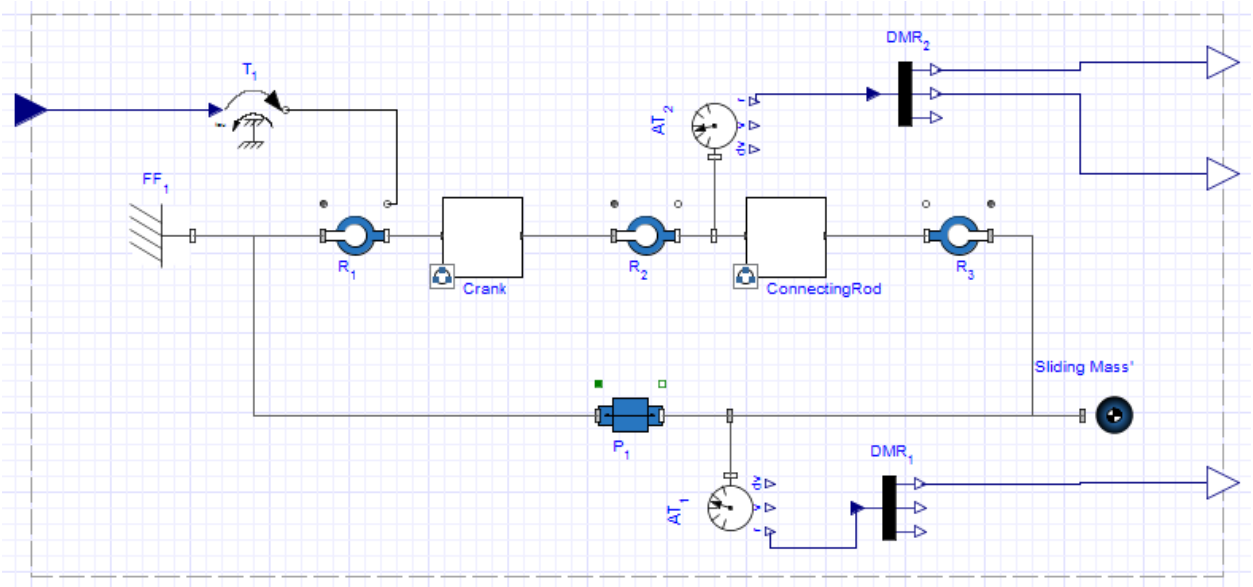
15. From the **1-D Mechanical** > **Rotational** > **Torque Drivers** menu, add a **Torque** component to the **Model Workspace** and place it above the **Fixed Frame** component.
16. Connect the white flange of the **Torque** component to the white flange of the leftmost **Revolute Joint**.



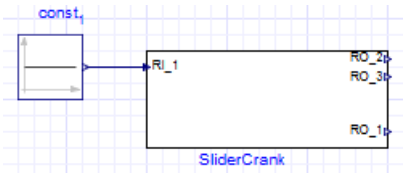
17. Click the input port of the **Torque** component, then drag your pointer to the subsystem boundary and click the boundary once. A real input port is added to your subsystem.



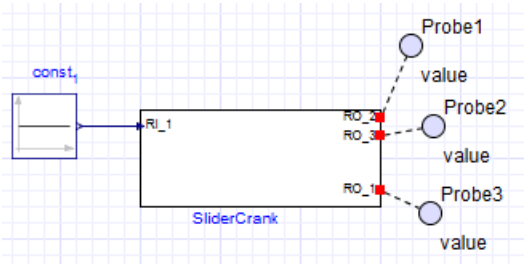
The complete subsystem appears below.



18. Click **Main** ( ) in the **Model Workspace** toolbar browse to the top level of the model.
19. From the **Signal Blocks** > **Sources** > **Real** menu, drag a **Constant** source into the **Model Workspace** and connect its output port to the input port of the **SliderCrank** subsystem as shown below.



20. Click **Attach probe** ( ) in the **Model Workspace** toolbar and then click the top output port of the **SliderCrank** subsystem.
21. Drag the probe to an empty location on the **Model Workspace**, and then click the workspace to position the probe.
22. In the same way, add probes to the other **SliderCrank** output ports as shown below.

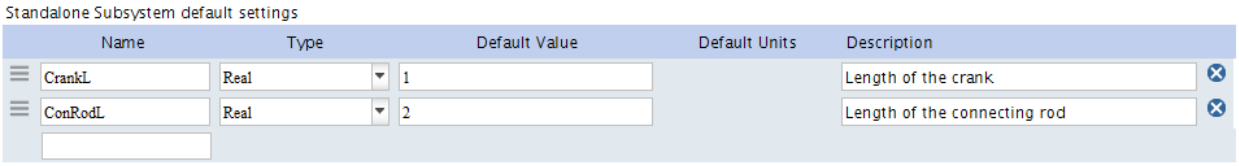


Define and Assign Subsystem Parameters

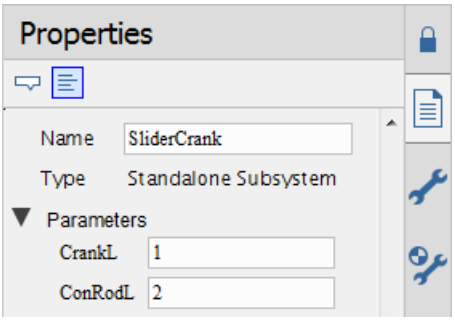
You can define custom parameters that can be used in expressions in your model to edit values more easily. To do so, you define a parameter with a numeric value in the parameter editor. You can then assign that parameter as a variable to the parameters of other components; those individual components will then inherit the numeric value of the parameter defined in the parameter editor. By using this approach, you only need to change the value in the parameter editor to change the parameter values for multiple components.

To edit parameters

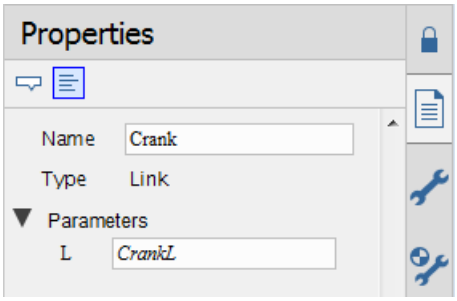
1. Double-click on the **SliderCrank** subsystem, and then click **Parameters** ( ) in the **Model Workspace** toolbar. The parameter editor appears.
2. In the first **Name** field, enter **CrankL** and press **Enter**.
3. Enter **1** in the **Default Value** field of **CrankL**, and then enter **Length of the crank** in the **Description** field.
4. In the second row of the table, enter **ConRodL** in the Name field and press **Enter**.
5. Enter **2** in the **Default Value** field of **ConRodL**, and then enter **Length of the connecting rod** in the **Description** field.



6. Click **Diagram View** ( ) to switch to the diagram view, and then click **Main** ( ).
7. Select the **SliderCrank** subsystem. The parameters are defined in the **Properties** tab ( ).



8. Double-click the **SliderCrank** subsystem, and then select the **Crank** subsystem.
9. In the **Properties** tab ( ), change the length value (**L**) to **CrankL**. The **Crank** subsystem now inherits the numeric value of **CrankL** that you defined.



10. Select the **ConnectingRod** subsystem and change its length value to **ConRodL**.
11. Click **Main** ( ) in the **Model Workspace** toolbar to navigate to the top level of the model.

You will include these parameter values in the model that you export. You are now ready to convert your model to an S-function block.

Exporting Your Model Using the Simulink® Component Block Generation App

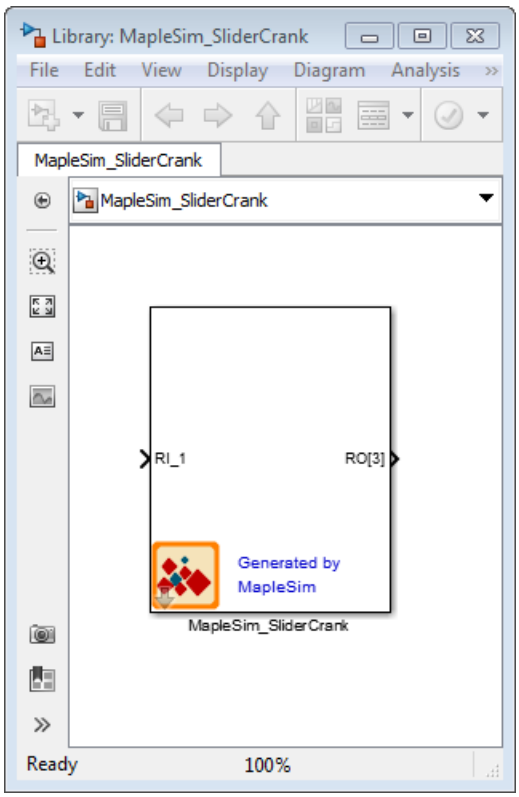
After preparing the model, you can use the **Simulink® Component Block Generation** app to set export options and convert the model to an S-function block.

To export your model:

1. Select the **Add Apps or Templates** tab ( ).
2. Double-click on the **Simulink® Component Block Generation** entry in the **Apps** palette. The **Analysis** window opens with the **Simulink® Component Block Generation** app loaded in the **Apps** tab.
3. In the **Subsystem Selection** section, select the **SliderCrank** subsystem from the drop-down list, and then click **Load Selected Subsystem**. All of the fields are populated with information specific to the subsystem.
4. In the **Configuration** section, select **Parameters**, and then select the **ConRodL** parameter that you defined in the previous section.

Note: The **Export** option for this parameter is selected by default. Also, by default, all input ports, output ports, and parameters in the model are kept as configurable parameters.

5. Click **Generate S-Function Code** to generate the S-function code and create the block.
6. Open the .m file in MATLAB® and run the code to open the block in Simulink®.



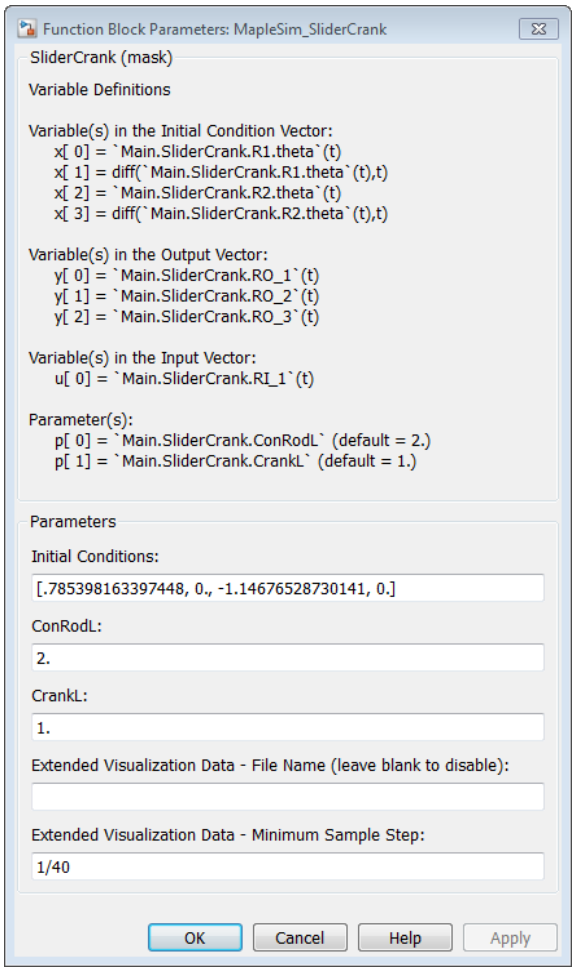
Note: Generating a block may require a few minutes.

Implement the S-Function Block in Simulink®

In Simulink®, you can connect your block to other compatible blocks, specify initial conditions, and edit the component parameter values.

To implement the S-Function block

1. In Simulink®, double-click the block. The **Parameter Mask** dialog box appears. This dialog box displays the **ConRodL** and **CrankL** parameters that you defined in MapleSim as a vector. The text in the dialog describes each parameter in the order they appear in the vector. Initial conditions can also be changed in this dialog box.



2. Click **Help**. This window provides a model description and information about the inputs, outputs, parameters, and initial conditions.
3. All inputs and outputs are implemented as vector signals. To access individual signals in Simulink®, use a **Mux** block for inputs and a **Demux** block for outputs.

Note: The extended visualization data parameters are used to generate a simulation results data file that can be imported into MapleSim with the simulation results generated by Simulink® into MapleSim. See *Enabling Extended Visualization Data (page 9)* for more information on these parameters.

2 Creating and Exporting Mathematical Models in Maple

In Maple, you can use commands from the **DynamicSystems** package to create a system from first principles. Maple contains a data structure called a *system object* that encapsulates the properties of a dynamic system. This data structure contains information, for example, the description of the system, and the description of the inputs. Five different types of systems can be created.

- Differential equation or difference equation
- Transfer function as an expression
- Transfer function as a list of numerator and denominator coefficients
- State-space
- Zero, pole, gain

You can create a **DynamicSystems** object in a new worksheet and use commands from the **MapleSimConnector** package to generate source code programmatically and save it as a MATLAB® .m file.

2.1 Creating and Exporting a DynamicSystems Object Programmatically

First, load the **DynamicSystems** and **MapleSimConnector** packages in the Maple worksheet.

```
> with(DynamicSystems) :
```

```
> with(MapleSimConnector) :
```

To create a system object from the transfer function $\frac{1}{s^2 + a \cdot s + b}$, use the following command:

```
> sys := TransferFunction(1/(s^2 + a*s + b))
```

$$sys := \begin{bmatrix} \textbf{Transfer Function} \\ \text{continuous} \\ \text{1 output(s); 1 input(s)} \\ \text{inputvariable} = [u1(s)] \\ \text{outputvariable} = [y1(s)] \end{bmatrix} \tag{2.1}$$

To view the details of the system, use the PrintSystem command.

```
> PrintSystem(sys)
```

$$\begin{bmatrix} \textbf{Transfer Function} \\ \text{continuous} \\ \text{1 output(s); 1 input(s)} \\ \text{inputvariable} = [u1(s)] \\ \text{outputvariable} = [y1(s)] \\ \mathrm{tf}_{1,1} = \dfrac{1}{s^2 + a\,s + b} \end{bmatrix} \tag{2.2}$$

The default values for the input names ($u1$) and output names ($y1$) have been used. Alternatively, during creation of the system, different input and output names can be specified.

To define parameters values, use the following command:

```
> par := [a = 1, b = 1]
```

$$par := [a = 1, b = 1] \tag{2.3}$$

Finally, use the SBlock command to generate the source code and the SaveCode command to save the code as a .c file and MATLAB® .m file.

```
> script := SBlock(sys, sys:-inputvariable, sys:-outputvariable,
      "MyTransferFunction", parameters = par) :
```

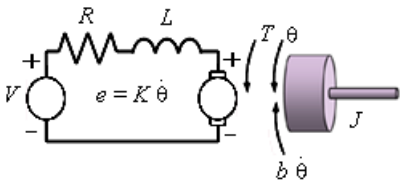
```
> SaveCode("MyTransferFunction", extension = "c", script[1],
      interactive = true) :
```

```
> SaveCode("MyTransferFunction", extension = "m", script[2],
      interactive = true) :
```

2.2 Example: DC Motor

Consider the classic example of the simplified DC motor. Using the built-in functionality of the **DynamicSystems** package in Maple, you can define the system model, and then visualize and simulate it before saving the code.

This example demonstrates how to define, analyze, and export a system programmatically.



To define, visualize and simulate a DC motor:

1. In a new Maple worksheet, define the system model.

Differential Equation Model:

```
> eq1 := L*(d/dt i(t)) + R*i(t) = v(t) - K*(d/dt θ(t)):
```

```
> eq2 := J*(d^2/dt^2 θ(t)) + b*(d/dt θ(t)) + Ks*θ(t) = K*i(t):
```

Transfer Function Model:

```
> sys_de := DynamicSystems:-DiffEquation( [eq1, eq2], [v(t)],
      [θ(t), i(t)] ):
  sys_tf := DynamicSystems:-TransferFunction(sys_de):
  sys_tf:-tf[1, 1]; sys_tf:-tf[2, 1];
```

$$\frac{K}{J\,L\,s^3 + (b\,L + J\,R)\,s^2 + (Ks\,L + K^2 + b\,R)\,s + Ks\,R}$$

$$\frac{J\,s^2 + b\,s + Ks}{J\,L\,s^3 + (b\,L + J\,R)\,s^2 + (Ks\,L + K^2 + b\,R)\,s + Ks\,R} \tag{2.4}$$

In place of the above commands, you could use the PrintSystem command to display each part of the model.

2. Specify the parameters in the model.

Description	(Initial) Value	Units
Input Variables		
Applied voltage	$v = 0$	V
Output Variables		
Motor shaft angular position	$\theta = 0$	rad
Motor current	$i = 0$	A
Parameters		
Moment of inertia of the motor	$J = 0.1$	$kg \cdot m^2$
Damping of the mechanical system	$b = 0.1$	$N \cdot m \cdot s$
Electromotive force constant	$K = 0.1$	$\frac{N \cdot m}{A}$
Motor coil resistance	$R = 1$	Ω
Motor coil inductance	$L = 0.5$	H
External Spring Load Constant	$Ks = 0$	$N \cdot m$

```
> params := [J = 0.1, b = 0.1, K = 0.1, R = 1, L = 0.5, Ks = 0]:
```

```
> ics := [i(0) = 0, θ(0) = 0, D(θ)(0) = 0]:
```

3. Generate and save the source code as a .c file and MATLAB® .m file.

```
> (cSFcn, MBlock) := MapleSimConnector:-SBlock(sys_tf,
      sys_tf:-inputvariable, sys_tf:-outputvariable,
      "MyTransferFunction", parameters = params,
      initialconditions = ics):
```

```
> MapleSimConnector:-SaveCode("MyTransferFunction",
      cSFcn, extension = "c", interactive = true) :

> MapleSimConnector:-SaveCode("MyTransferFunction",
      MBlock, extension = "m", interactive = true) :
```

With the basic tools shown in this guide, you are now ready to use the MapleSim Connector to solve many system design problems. For more information about the commands used in this guide, enter DynamicSystems and MapleSimConnector in the Maple help system.

Index

A

Apps
  Simulink® Block Generation, 1, 16

C

Code Export Options, 3

D

DynamicSystems object, 18
  Creating and Exporting Programmatically, 18
  Transfer function, 18

E

Export S-Function, 6
Exporting, iv
Extended Visualization Data, 9
External Libraries, iv

G

Generate
  External Libraries, 6

I

Inputs and outputs, 11
Inputs/Outputs and Parameter Configuration, 1

M

MapleSim Connector Examples, 7
Mathematical model, 18
Models using external libraries, iv

S

S-Function
  Export, 6
  View Code, 6
Simulink®, 16
Subsystem
  Creating, 10
  Preparation, 1
  Selection, 1
Subsystem parameters, 14
System object, 18

V

View Code, 6