Getting Started with the MapleSim Connector

Getting Started with the MapleSim Connector

Contents

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.

System Requirements

For installation instructions and a complete list of 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 Setting Up the MapleSim Connector

To generate an S-function block and have Maple communicate with MATLAB® you have to establish a connection with MATLAB®.

To set up the MapleSim Connector

1. Start Maple.
2. Enter the following command to establish a connection with MATLAB®.

```
> Matlab[evalM]("simulink")
```

3. A MATLAB® command window opens and the connection is established. If the window does not open, follow the instructions in the Matlab/setup help page in the Maple help system to configure the connection.
4. Next, set up the MATLAB® mex compiler. Go to the MATLAB® command window and enter the following setup command:

```
>> mex -setup
Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n?
```

5. Follow the instructions to choose a local C compiler that supports ANSI (American National Standards Institute) C code. See the MapleSimConnector,setup help page for more information.

You are now ready to use the MapleSim Connector.

1.2 Getting Help

In Maple, enter ?MapleSimConnector at a prompt in a worksheet.

1.3 Using the Simulink® Component Block Generation Template

The MapleSim Connector provides a **Simulink® Component Block Generation** template in the form of a Maple worksheet for manipulating and exporting MapleSim subsystems. This template 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 template, you can define inputs and outputs for the system, set the level of code optimization, chose 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 now handles 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 Template.

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

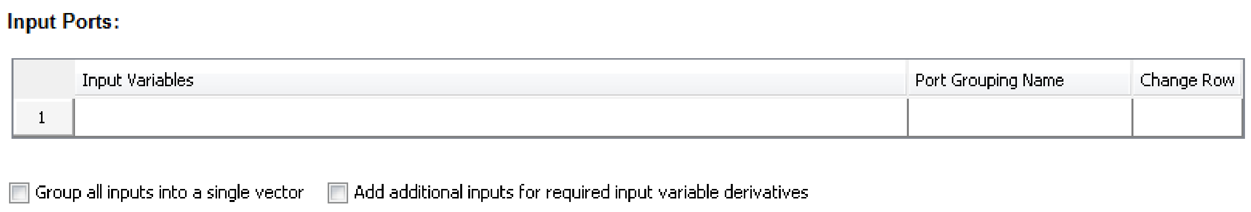
You can select which subsystems from your model you want to export to a Simulink® block. After a subsystem is selected, click **Load Selected Subsystem**. All defined input and output ports are loaded.

Port and Parameter Management

Port and Parameter Management 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.



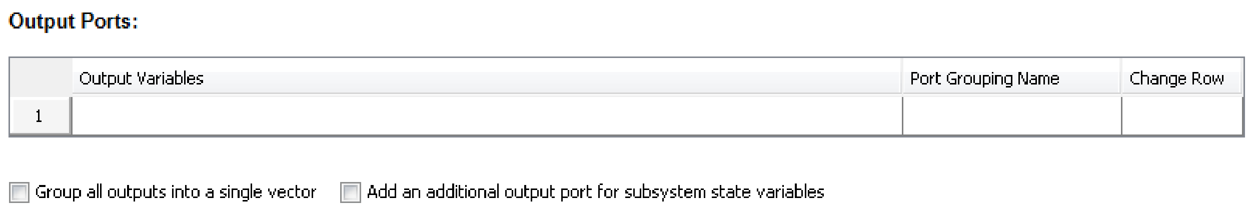
Input Ports:

	Input Variables	Port Grouping Name	Change Row
1			

☐ Group all inputs into a single vector ☐ Add additional inputs for required input variable derivatives

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



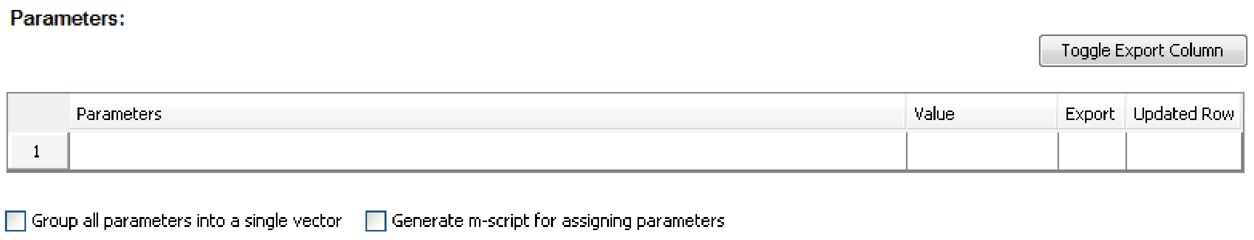
Output Ports:

	Output Variables	Port Grouping Name	Change Row
1			

☐ Group all outputs into a single vector ☐ Add an additional output port for subsystem state variables

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

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


Parameters:

Toggle Export Column

	Parameters	Value	Export	Updated Row
1				

☐ Group all parameters into a single vector ☐ Generate m-script for assigning parameters

Select **Group all parameters into a single vector** to 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.

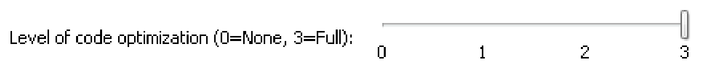
Press **Toggle Export Column** to toggle selected/unselected parameters for export.

S-Function Options

These settings specify the advanced options for the code generation process.

Optimization Options

Set the level of code optimization to specify whether equations are left in their implicit form or converted to an ordinary differential equation (ODE) system during the code generation process. This option specifies the degree of simplification applied to the model equations during the code generation process and eliminates redundant variables and equations in the system.


Level of code optimization (0=None, 3=Full): 0 1 2 3 (slider at 3)

Select one of the following options:

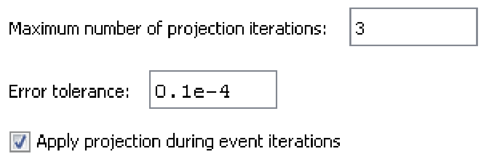
None (0): no optimization is performed; the default equations will be used in the generated code.

Partial (1, 2): removes redundant equations from the system.

Full (3): performs index reduction to reduce the system to an ODE system or a differential algebraic equation (DAE) system of index 1, and removes redundant equations.

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.


Maximum number of projection iterations: 3

Error tolerance: 0.1e-4

☑ Apply projection during event iterations

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

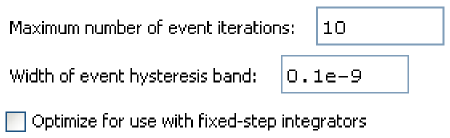
Set the **Error tolerance to** specify the desirable error tolerance to achieve after the projection.

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.


Maximum number of event iterations: 10
Width of event hysteresis band: 0.1e-9
☐ Optimize for use with fixed-step integrators

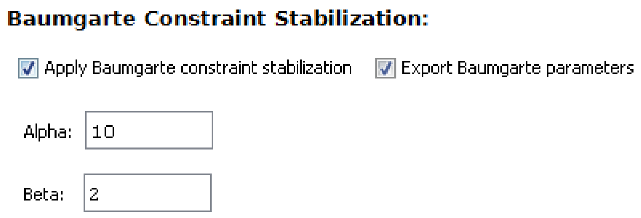
Set the **Maximum number of event iterations** to specify the maximum number of times that a projection is permitted to iterate to obtain a more accurate solution.

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

Select **Optimize for use with fixed-step integrators** to optimize the event iterations as a function of hysterias bandwidth.

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.


Baumgarte Constraint Stabilization:
☑ Apply Baumgarte constraint stabilization ☑ Export Baumgarte parameters
Alpha: 10
Beta: 2

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.

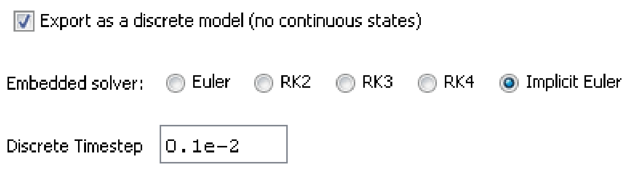
Discretization

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

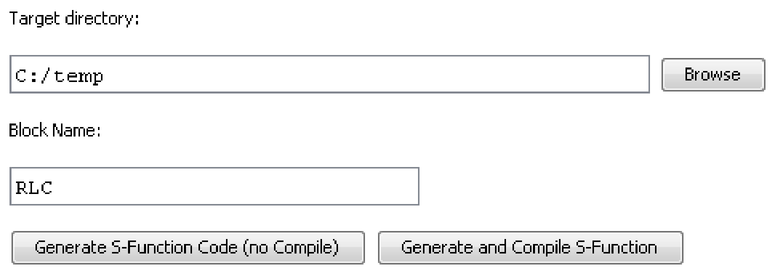
In this section, you can also set the **Discrete Timestep** (in seconds) for the discretization.


[✓] Export as a discrete model (no continuous states)

Embedded solver: Euler RK2 RK3 RK4 [✓ Implicit Euler]

Discrete Timestep: 0.1e-2

Generate S-Function


Target directory: C:/temp

Block Name: RLC

Browse
Generate S-Function Code (no Compile)
Generate and Compile S-Function

Provide a name and specify the location for the generated file.

To generate an S-Function block without a Simulink® connection, click **Generate S-Function (no Compile)**.

To generate an S-Function block, click **Generate and Compile S-Function**.

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 S-Function

After you generate the S-Function code and create 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

1.4 Viewing MapleSim Connector Examples

Toolbox examples are available in the **MapleSim Connector Examples** palette in MapleSim.

Each example includes a code generation template in its **Attachments** palette.

To view an example:

1. Expand the **MapleSim Connector Examples** palette on the left side of the MapleSim window, and click the entry for the model that you want to view.
2. In the **Project** tab, expand the **Attachments** palette and then expand **Documents**.

3. Right-click (**Control**-click for Macintosh) **Simulink® Component Block Generation** and select **View**. The template opens in Maple.

Some models include additional documents, such as templates that display model equations or define custom components. You can open any of these documents by right-clicking its entry and selecting **View**.

1.5 Example: RLC Circuit Model

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

Note: Before starting this tutorial, you must set up MATLAB® and the mex compiler in order to have the template appear in the list. For more information, see the MapleSimConnector,setup help page for more information.

To generate an S-function block

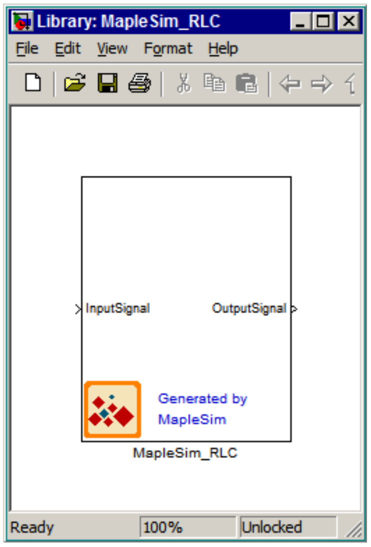
1. In the **MapleSim Connector Examples** palette, select the **RLC Parallel Circuit** example.
2. Click **Templates** (📎) in the **Main Toolbar**.
3. From the list, select **Simulink® Component Block Generation**.
4. In the **Attachment** field, enter **RLC Circuit** as the worksheet name and click **Create Attachment**. Your MapleSim model opens in the **Simulink® Component Block Generation** template in Maple.
5. Using the navigation controls above the model, select **Main > RLC**. The RLC subsystem appears in the workspace.
6. Click **Load Selected Subsystem**. All of the template fields are populated with information specific to the RLC subsystem.

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

7. In the **S-Function Option** section, set the **Level of code optimization** option to **Full (3)**.
8. In the **Generate F-Function** section, specify the location of generated files.
9. Click **Generate and Compile S-Function** to generate the S-function code and create the block.

Note: Generating a block may require a few minutes.

A MATLAB® command window opens and the block with the specified parameters is generated in Simulink®.



Double-clicking the block opens the mask that contains the symbolic parameters from the original model. This block can now be connected with any compatible Simulink® blocks.

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 template.

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

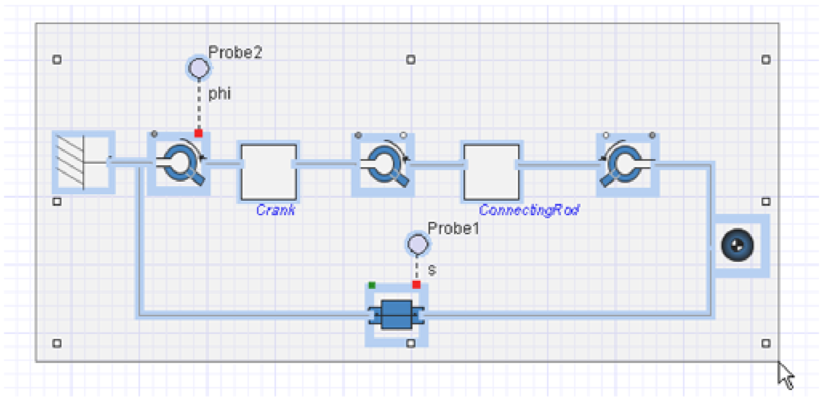
1. In MapleSim, under the **Libraries** tab, browse to the **Examples** > **User's Guide Examples** menu.
2. Open the **Planar Slider-Crank Mechanism** example in Chapter 6. The example appears in the **Model Workspace**.

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

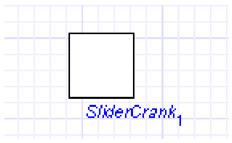
1. Using the selection tool ( ) located above the **Model Workspace**, draw a box around all of the components in the model.



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

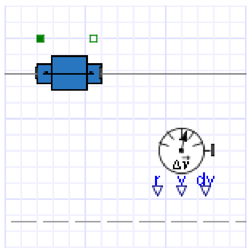
MapleSim uses a topological representation to connect interrelated components without having to consider how signals flow between them, whereas traditional signal-flow modeling tools require explicitly defined system inputs and outputs. Since Simulink® only supports data signals, properties on acausal 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.

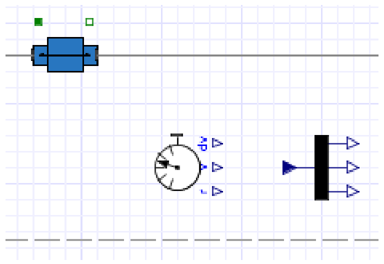
In this example, you will convert the displacements of the slider and the joint between the crank and connecting rod to output signals. The input signal needs to be converted to a torque that is applied to the revolute joint that represents the crank shaft.

To convert the system signals

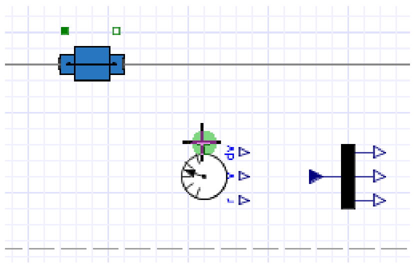
1. Double-click the subsystem block to view its contents. The broken line surrounding the components indicates the subsystem boundary, which can be resized by clicking and dragging its sizing handles.
2. Delete the probes that are attached to the model.
3. On the left side of the MapleSim window, expand 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.



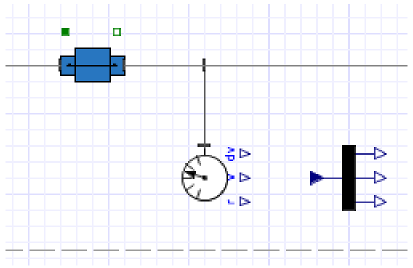
5. Right-click (**Control**-click for Macintosh®) the **Absolute Translation** component and select **Rotate Counterclockwise**.
6. From the **Signal Blocks** > **Routing** > **Demultiplexers** menu, drag a **Real Demultiplexer** component to the **Model Workspace** and place it to the right of the **Absolute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



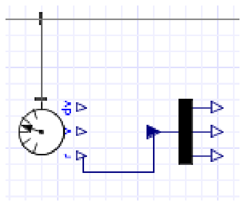
7. To connect the **Absolute Translation** component to the model, click the frame_b connector. The frame is highlighted in green when you hover your pointer over it.



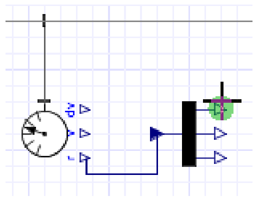
8. Draw a vertical line and click the connection line directly above the component. The sensor is connected to the rest of the diagram.



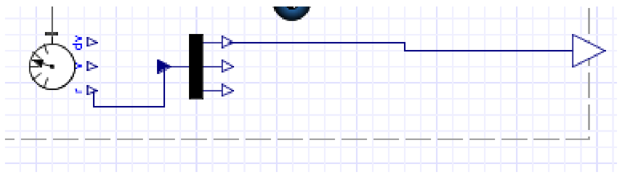
9. In the same way, connect the **r** output port (*TMOutputP*) of the **Absolute Translation** component to the demultiplexer Real input signal (u) port. This is the displacement signal from the sensor in x, y, and z coordinates. Since the slider only moves along the x axis, the first coordinate needs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



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



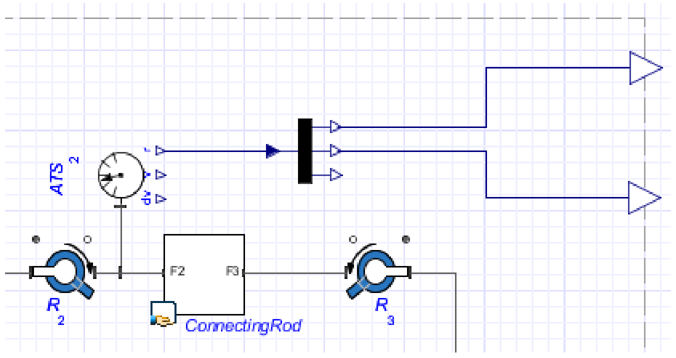
11. Drag your pointer to the subsystem boundary and then click the boundary once. A real output port is added to your subsystem.



12. Add another **Absolute Translation** component above the **Connecting Rod** subsystem.

13. Right-click (**Control**-click for Macintosh) the **Absolute Translation** component and select **Flip Vertically**. Right-click the **Absolute Translation** component again and select **Rotate Clockwise**.

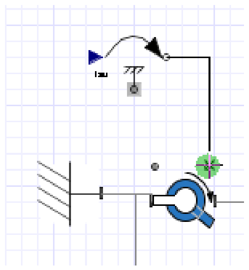
14. Add a **Real Demultiplexer** component to the right of the sensor and connect the components as shown below.



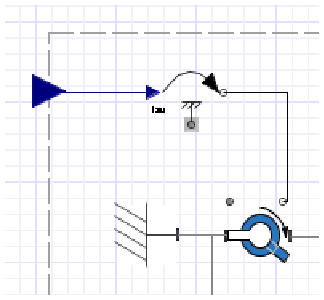
Note: Since the crank is moving in the x-y plane, you only need to output the first two signals.

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

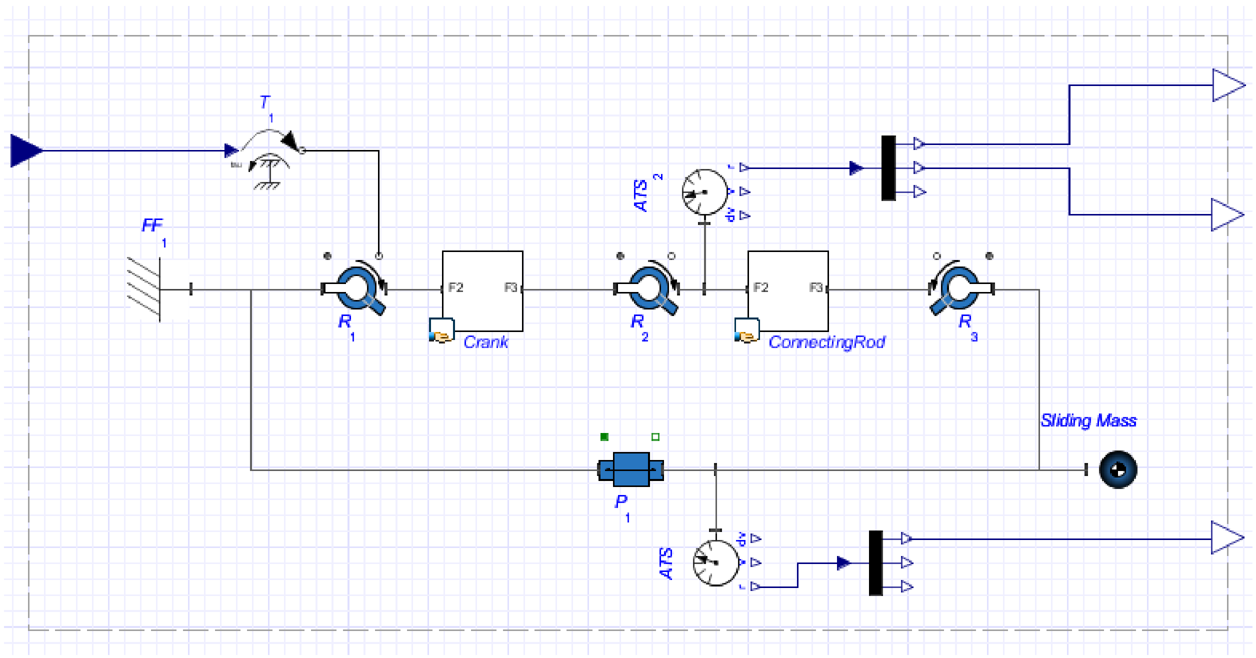
15. From the **1-D Mechanical** > **Rotational** > **Torque Drivers** menu, add a **Torque** component to the **Model Workspace** and place it above the **Fixed Frame** component.
16. Connect the white flange of the **Torque** component to the white flange of the leftmost **Revolute Joint**.



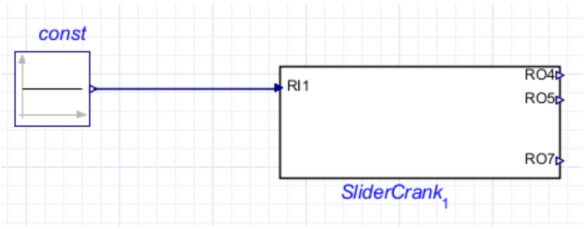
17. Click the input port of the **Torque** component, then drag your pointer to the subsystem boundary and click the boundary once. A real input port is added to your subsystem.



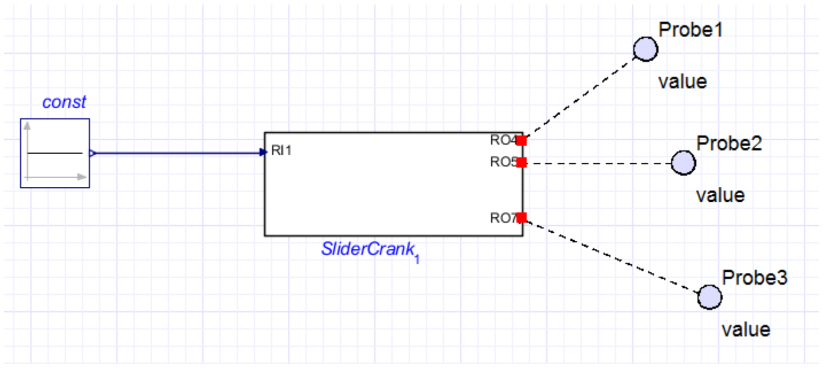
The complete subsystem appears below.



18. Click **Main** above the **Model Workspace** to 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** ( ) above the **Model Workspace** 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.

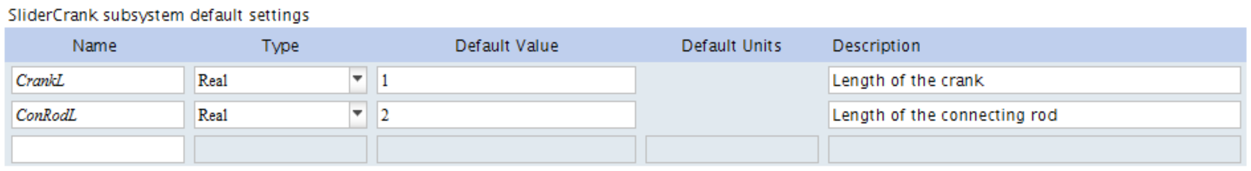


Define and Assign Subsystem Parameters

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.

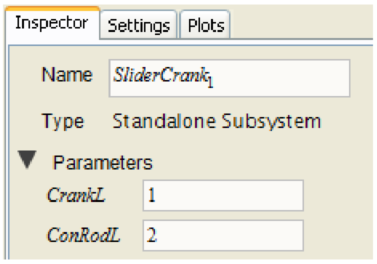
To edit parameters

1. While in the detailed view of the **SliderCrank** subsystem, click **Parameters** ( ) above the **Model Workspace**. The parameter editor appears.
2. In the **New Parameter** field, define a parameter called **CrankL** and press **Enter**.
3. Specify a default value of **1** and enter **Length of the crank** as the description.
4. In the second row of the table, define a parameter called **ConRodL** and press **Enter**.
5. Specify a default value of **2** and enter **Length of the connecting rod** as the description.


SliderCrank subsystem default settings

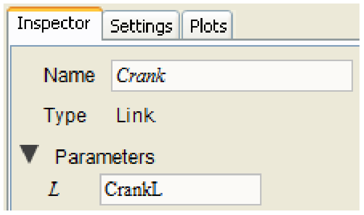
Name	Type	Default Value	Default Units	Description
CrankL	Real	1		Length of the crank
ConRodL	Real	2		Length of the connecting rod

6. Click **Diagram** ( ) to switch to the diagram view. The parameters are defined in the **Parameters** pane.



Name *SliderCrank*$_1$
Type Standalone Subsystem
Parameters
CrankL 1
ConRodL 2

7. In the **Model Workspace**, select the **Crank** subsystem.
8. In the **Parameters** pane, change the length value (**L**) to **CrankL**. The **Crank** subsystem now inherits the numeric value of **CrankL** that you defined.



Name *Crank*
Type Link
Parameters
L CrankL

9. Select the **ConnectingRod** subsystem and change its length value to **ConRodL**.
10. Click **Main** in the **Navigation Toolbar** above the workspace 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 Template

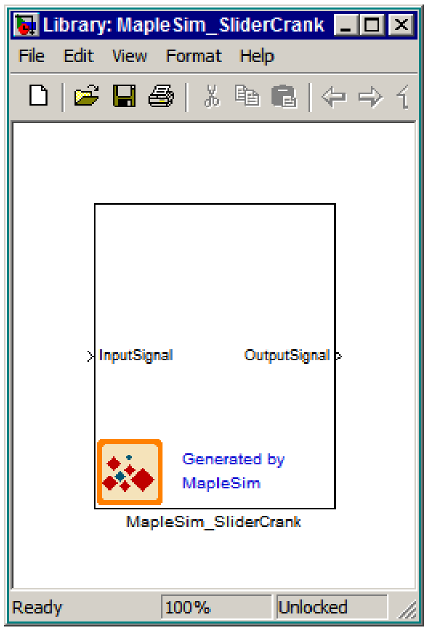
After preparing the model, you can use the **Simulink® Component Block Generation** template to set export options and convert the model to an S-function block.

To export your model

1. Click **Templates** ( ) in the **Main Toolbar**.
2. From the list, select **Simulink® Component Block Generation**.
3. In the **Attachments** field, enter **Slider Crank S-Function** as the worksheet name and click **Create Attachment**. The slider-crank subsystem opens in the **Simulink® Component Block Generation** template in Maple.
4. From the drop-down menu above the model, select **SliderCrank**.
5. In **Step 1: Subsystem Selection** of the template, click **Load Selected Subsystem**. All of the template fields are populated with information specific to the subsystem.
6. In the **Setting Parameters** section, in the **Parameter Name** list, select the **ConRodL** parameter that you defined in the previous section.

Note: The **Keep as Block Parameter** box is selected by default. Also, by default, all input and output ports, and parameters in the model are kept as configurable parameters.

7. Click **Generate and Compile S-Function** to generate the S-function code and create the block. A MATLAB® command window opens and the block with the specified parameters is generated in Simulink®.



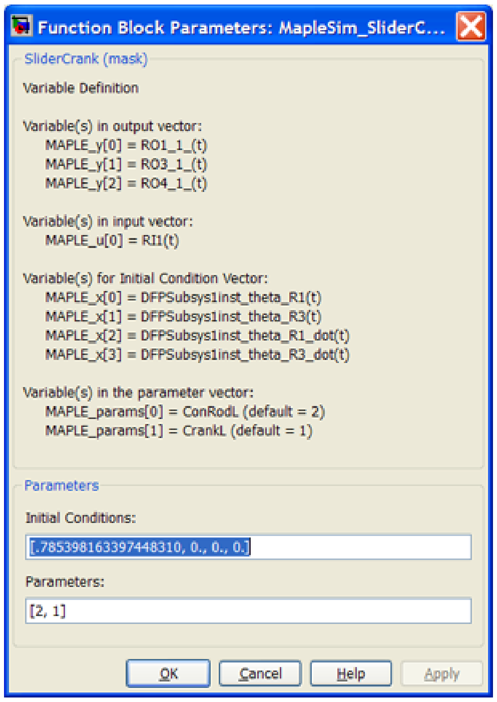
Note: Generating a block may require a few minutes.

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.



Function Block Parameters: MapleSim_SliderC...

SliderCrank (mask)

Variable Definition

```
Variable(s) in output vector:
    MAPLE_y[0] = RO1_1_(t)
    MAPLE_y[1] = RO3_1_(t)
    MAPLE_y[2] = RO4_1_(t)

Variable(s) in input vector:
    MAPLE_u[0] = RI1(t)

Variable(s) for Initial Condition Vector:
    MAPLE_x[0] = DFPSubsys1inst_theta_R1(t)
    MAPLE_x[1] = DFPSubsys1inst_theta_R3(t)
    MAPLE_x[2] = DFPSubsys1inst_theta_R1_dot(t)
    MAPLE_x[3] = DFPSubsys1inst_theta_R3_dot(t)

Variable(s) in the parameter vector:
    MAPLE_params[0] = ConRodL (default = 2)
    MAPLE_params[1] = CrankL (default = 1)
```

Parameters

Initial Conditions: [.785398163397448310, 0., 0., 0.]

Parameters: [2, 1]

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.

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:

$$> sys := TransferFunction\left(\frac{1}{s^2 + a \cdot s + b}\right)$$

$$sys := \left[\begin{array}{c} \textbf{Transfer Function} \\ \text{continuous} \\ \text{1 output(s); 1 input(s)} \\ \text{inputvariable} = [u1(s)] \\ \text{outputvariable} = [y1(s)] \end{array}\right. \qquad (2.1)$$

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

> *PrintSystem*(*sys*)

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

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:

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

$$par := [a = 1, b = 1] \quad (2.3)$$

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) :
```

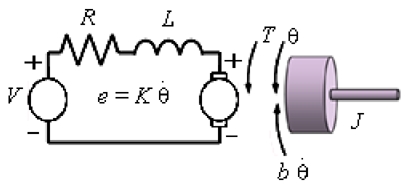
```
> SaveCode("MyTransferFunction", extension = "c", script[1],
      interactive = true) :
```

```
> 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:

$> eq1 := L\left(\frac{\mathrm{d}}{\mathrm{d}t}\, i(t)\right) + R\, i(t) = v(t) - K\left(\frac{\mathrm{d}}{\mathrm{d}t}\, \theta(t)\right) :$

$> eq2 := J\left(\frac{\mathrm{d}^2}{\mathrm{d}t^2}\, \theta(t)\right) + b\left(\frac{\mathrm{d}}{\mathrm{d}t}\, \theta(t)\right) + Ks\, \theta(t) = K\, i(t) :$

Transfer Function Model:

$> sys\_de := DiffEquation(\, [eq1, eq2], [v(t)], [\theta(t), i(t)]\,) :$
$sys\_tf := TransferFunction(sys\_de) :$
$sys\_tf{:-}tf[1,1];\ sys\_tf{:-}tf[2,1];$

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

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

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	$K = 0.1$	$\left[\!\left[\frac{N \cdot m}{A}\right]\!\right]$
Motor coil resistance	$R = 1$	$[\![\Omega]\!]$
Motor coil inductance	$L = 0.5$	$[\![H]\!]$
External Spring Load Constant	$Ks = 0$	$[\![N \cdot m]\!]$

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

$> ics := [\, i(0) = 0, \theta(0) = 0, \mathrm{D}(\theta)(0) = 0\,] :$

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

$> (cSFcn,\ MBlock) := SBlock(sys, sys{:-}inputvariable, sys{:-}$
$\quad outputvariable,$ "MyTransferFunction", $parameters = params,$
$\quad initialconditions = ics) :$

```
> 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. Enter **?DynamicSystems** and **?MapleSimConnector** at a prompt in a Maple worksheet for more information about the commands used in this guide.

Index