Getting Started with the MapleSim FMI Connector

Getting Started with the MapleSim FMI Connector

This document was produced using Maple and DocBook.

Contents

Introduction

The MapleSim™ FMI Connector and FMI Connector package provides all of the tools you need to prepare and export your dynamic systems models into an FMU (Functional Mock-up Unit) archive file. You can create a model in MapleSim, simplify it in Maple™ by using an extensive range of analytical tools, and then generate FMU executables that you can incorporate into your MODELISAR toolchain.

Scope of Model Support

MapleSim is a comprehensive modeling tool where it is possible to create models that could go beyond the scope of this FMI Connector. In general, the MapleSim FMI 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 FMU archive. 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 FMI Code Generation Steps

This chapter describes how to use the FMI template and in the *Example: RLC Circuit Model (page 6)* section of this chapter, a step by step procedural example shows you how to create an FMU archive file. The **FMU Generation** template consists of the following steps to generate C code:

1. Subsystem selection
2. Port and parameter management
3. FMI code generation options
4. Generate FMI C code
5. View generated FMI C code

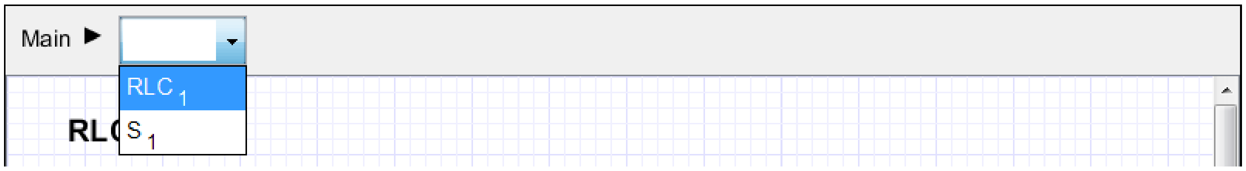
The FMI Connector package

The FMI Connector package is a collection of procedures for manually generating and compiling FMI code from MapleSim models. For information about the FMI Connector package, in Maple, enter `?FMIConnector` at a prompt in a Maple worksheet.

1.2 Opening an FMI Template

To open the FMU Generation template

1. Click **Templates** ( ) in the main toolbar and select the **FMU Generation** template.
2. In the **Attachment** field, provide a worksheet name and click **Create Attachment**. Your MapleSim model opens in a Maple worksheet in the Subsystem Selection window.
3. From the drop down list select a subsystem. The subsystem and its contents appear in the MapleSim embedded component window.



1.3 Using the FMU Generation Template

The MapleSim FMI Connector provides an **FMU Generation** template in the form of a Maple worksheet for manipulating and exporting MapleSim subsystems as FMU archive files. This template contains pre-built embedded components that allow you to export FMU archive files from a MapleSim subsystem.

With this template, you can define inputs and outputs for the system, set the level of code optimization, generate the source code, and choose the format of the resulting FMU component and library code. You can use any Maple commands to perform task analysis, assign model equations to a variable, group inputs and outputs and define additional input and output ports for variables.

Note: FMU component generation now handles all systems modeled in MapleSim, including hybrid systems with defined signal input (RealInput) and signal output (RealOutput) ports.

Example models are available in the **FMI Connector Examples** palette in MapleSim.

Step 1: Subsystem Selection

This part of the template identifies the subsystem modeling components that you want to export as a block component. Since FMI 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 to your model. 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 the FMU archive files. By creating a subsystem you not only improve the visual layout of a system in model workspace and but also prepare the model for export. The example in Chapter 2 shows you how to group all of the components into a subsystem.

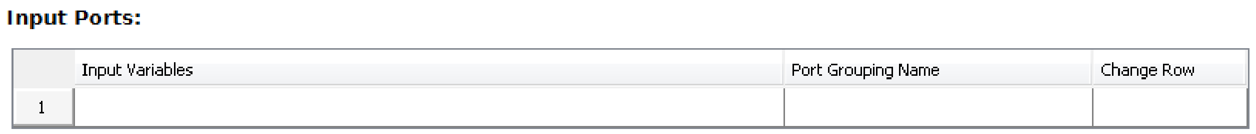
Note: For connectors you must use signal components since acausal connectors can not be converted to a signal.

You can select which subsystems from your model you want to export to an FMU archive file. After selecting a subsystem, click **Load Selected Subsystem**. All defined input and output ports are loaded.

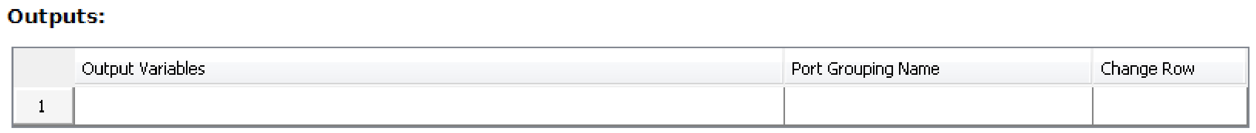
Step 2: Port and Parameter Management

The Port and Parameter Management interface 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.

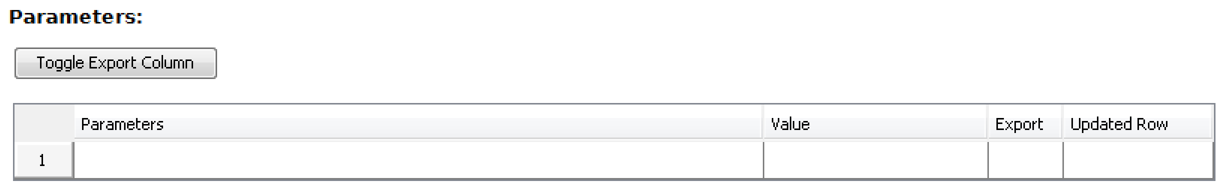
After the subsystem is loaded you can group individual input and output variable elements into a vector array, and add additional input and output ports for customized parameter values. Input ports can include variable derivatives, and output ports can include subsystem state variables.


Input Ports:

	Input Variables	Port Grouping Name	Change Row
1			


Outputs:

	Output Variables	Port Grouping Name	Change Row
1			


Parameters:

	Parameters	Value	Export	Updated Row
1				

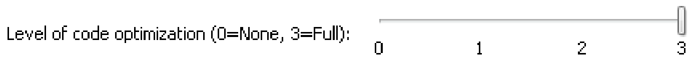
Note: If the parameters are not marked for export they will be numerically substituted.

Step 3: FMI Code Generation Options

The FMI Component Options 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): 3

Select one of the following options:

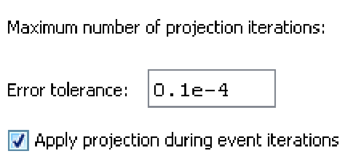
None (0): no optimization is performed; the default equations will be used in the generated code.

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** specifies whether the constraints are satisfied in a DAE system by using constraint projection in the generated FMU archive file. 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.

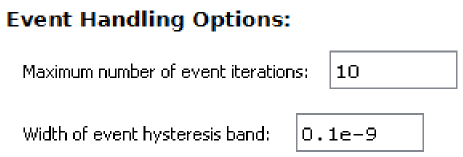
Set the **Error tolerance** to specify the desirable error tolerance to achieve after the projection.

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** specifies whether the events are satisfied in a DAE system by using event projection in the generated FMU archive file. 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.



Event Handling Options:

Maximum number of event iterations: 10

Width of event hysteresis band: 0.1e-9

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.

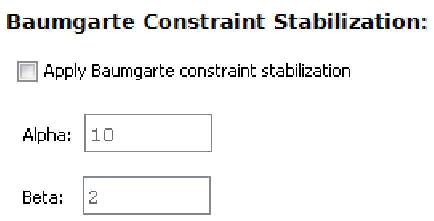
Set the **Width of event hysterias band** to specify the desirable error tolerance to achieve after the projection.

Event projection is performed using the **event 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.

Note: Currently, if the model has events, they are handled using the event handling functions in the generated Msim-Model.c file, and not the FMI provided Event Handling routines.

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

Alpha: 10

Beta: 2

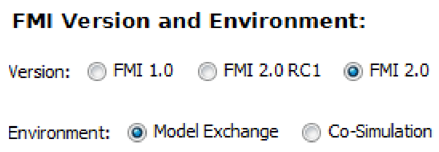
Baumgarte: Apply the Baumgarte constraint stabilization.

Alpha: Set the derivative gain for Baumgarte constraint stabilization.

Beta: Set the proportional gain for Baumgarte constraint stabilization.

Step 4: Generate FMI C Code

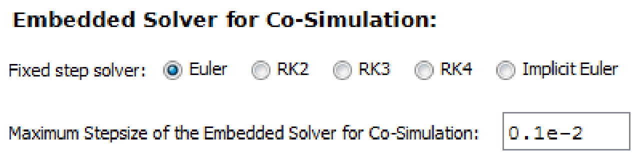
Select the FMI version and environment for your code. You can choose between **FMI 1.0**, **FMI 2.0 RC1**, and **FMI 2.0** for the version.


FMI Version and Environment:

Version: FMI 1.0 FMI 2.0 RC1 [✓ FMI 2.0]

Environment: [✓ Model Exchange] Co-Simulation

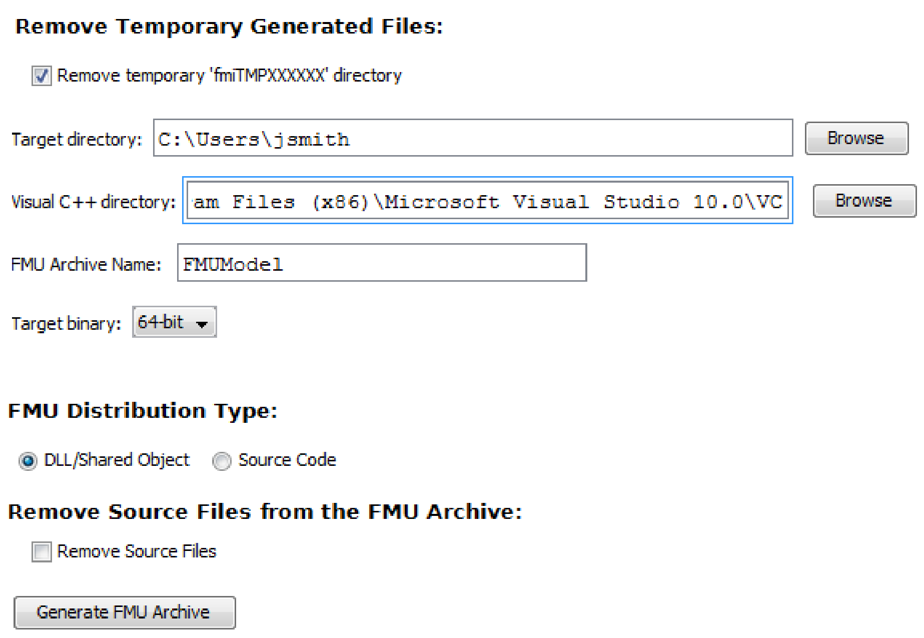
If you are exporting to the Co-Simulation environment, specify the fixed-step solver and maximum stepsize.


Embedded Solver for Co-Simulation:

Fixed step solver: [✓ Euler] RK2 RK3 RK4 Implicit Euler

Maximum Stepsize of the Embedded Solver for Co-Simulation: 0.1e-2

Generating the C code creates temporary files for viewing purposes.



Remove Temporary Generated Files:

[✓] Remove temporary 'fmiTMPXXXXXX' directory

Target directory: C:\Users\jsmith

Visual C++ directory: am Files (x86)\Microsoft Visual Studio 10.0\VC

FMU Archive Name: FMUModel

Target binary: 64-bit

FMU Distribution Type:

● DLL/Shared Object ○ Source Code

Remove Source Files from the FMU Archive:

[ ] Remove Source Files

Select **Remove temporary `fmiTMPXXXXXX` directory** to remove temporary files after code generation.

Specify the locations for the **Target Directory** and **Visual C++ Directory**, and provide a name for the generated **FMU Archive**.

Note: The Visual C++ Directory is not required for Linux and Macintosh® platforms.

If required, set the binary target to **32-bit** or **64-bit**.

Select **Remove Source Files from the FMU Archive:** to remove source files after code generation.

To generate an FMU archive click **Generate FMU Archive**.

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.

Step 5: View Generated FMI C Code

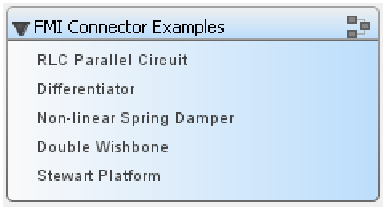
After the C code is generated, the **FMI C Code** and **MsimModel.c** components can be viewed.

1.4 Viewing Examples

Within MapleSim there are many examples for you to view.

To view an example:

1. In the **Libraries** tab on the left side of the MapleSim window, expand the **FMI Connector Examples** palette, and 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 **Project** tab, expand the **Attachments** palette and then expand **Documents**. You can open any of these documents by right-clicking (**Control**-clicking, for Macintosh) its entry in the list and clicking **View**. After you add a template to a model, it will be available from this list.

1.5 Example: RLC Circuit Model

In this example, you will generate an FMU archive file using an RLC circuit model created in MapleSim.

To generate an FMU archive file:

1. From the FMI Connector Examples palette, open the **RLC Parallel Circuit** example.
2. Click **Templates** ( ) in the main toolbar.
3. From the list, select the **FMU Generation** template.
4. In the **Attachment** field, enter **RLC Circuit** as the worksheet name.
5. Click **Create Attachment**. Your MapleSim model opens in Maple, using the selected template.
6. Browse to the **RLC Parallel Circuit 1** subsystem by selecting the subsystem name from the drop-down menu in the toolbar above the model diagram. This menu displays all of the subsystems and components in your MapleSim model.
7. Click **Load Selected Subsystem**. All of the template fields are populated with information specific to the subsystem displayed in the model diagram. You can now specify which subsystem parameters will be kept as configurable parameters in the generated block.
8. In the **FMI Code Generation Options** section, set the **Optimization Options** to **Full (3)**. 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.
9. In the **Generate FMI C Code** section of the template, specify the target, the Visual C++ directories and FMU archive name.
10. Click **Generate FMU Archive.** The .fmu zip file is created and saved in the target directory.

Note: Generating a block may require a few minutes.

2 Example: Exporting a Model as an FMU File

2.1 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 FMI file.

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 FMI file Generation template.

To open the slider-crank mechanism example

1. In MapleSim, under the **Libraries** tab, expand the **Examples** palette and then expand the **User's Guide Examples** submenu.

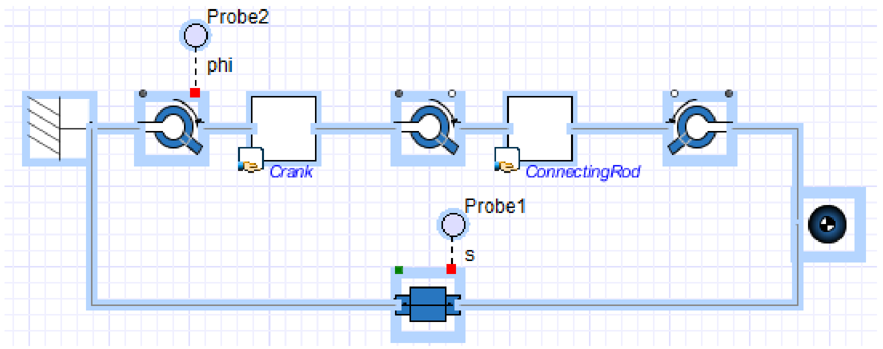
2. Open the **Planar Slider-Crank Mechanism** example in Chapter 6.

Converting the Model to a Subsystem

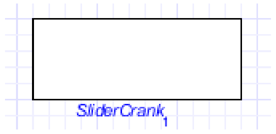
By converting your entire model or part of your model into a subsystem, you identify which parts of the model that you want to export. In this example, you will group all of the components into a subsystem.

To create 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**.
3. In the **Create Subsystem** dialog box, enter **SliderCrank** as the subsystem name.
4. Click **OK**. A **SliderCrank** subsystem block appears in the model workspace.



Defining Subsystem Inputs and Outputs

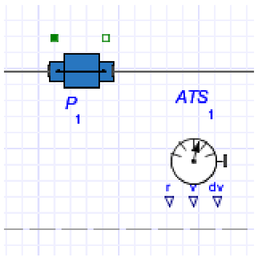
MapleSim uses a topological representation to connect interrelated components without having to consider how signals flow between them, whereas traditional signal-flow modeling tools require explicitly defined system inputs and outputs.

Since FMI 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 FMU file.

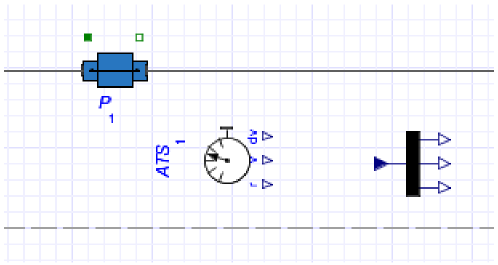
In this example, you will convert the displacements of the slider and the joint between the crank and connecting rod to output signals. The input signal needs to be converted to a torque that is applied to the revolute joint that represents the crank shaft.

To create a subsystem output port

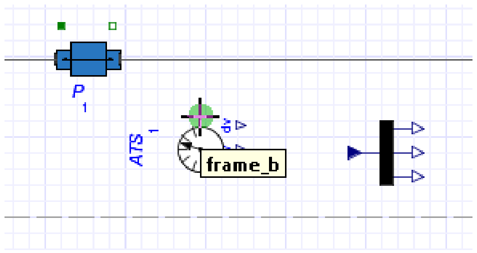
1. Double-click the subsystem block to view its contents. The broken line surrounding the components indicates the subsystem boundary, which can be resized by clicking and dragging its sizing handles.
2. Delete the probes that are attached to the model.
3. In the **Libraries** tab on the left side of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



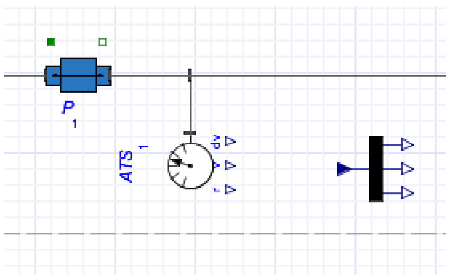
5. Right-click (**Control**-click for Macintosh®) the **Absolute Translation** component and select **Rotate Counterclockwise**.
6. From the **Signal Blocks** > **Routing** > **Demultiplexers** menu, drag a **Real Demultiplexer** component to the **Model Workspace** and place it to the right of the **Absolute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



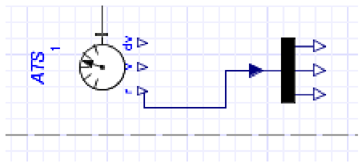
7. To connect the **Absolute Translation** component to the model, click the frame_b connector. The frame is highlighted in green when you hover your pointer over it.



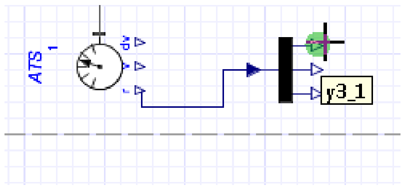
8. Draw a vertical line and click the connection line directly above the component. The sensor is connected to the rest of the diagram.



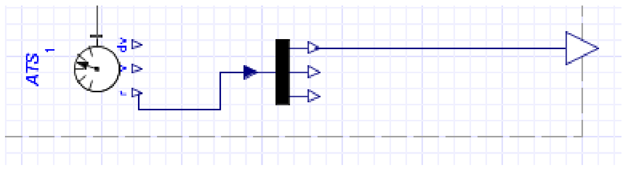
9. In the same way, connect the **r** output port (*TMOutputP*) of the **Absolute Translation** component to the input port of the demultiplexer. This is the displacement signal from the sensor in x, y, and z coordinates. Since the slider only moves along the x axis, the first coordinate must be an output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



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



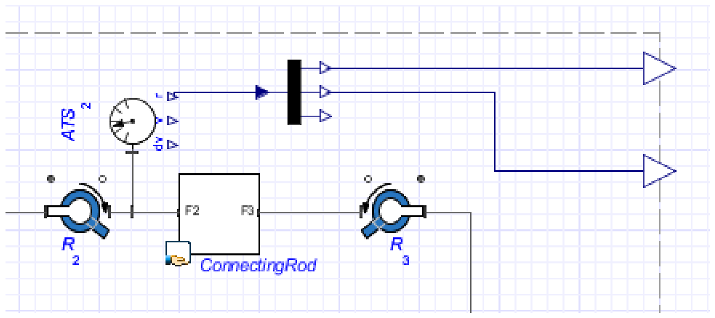
11. Drag your pointer to the subsystem boundary and then click the boundary once. A real output port is added to your subsystem.



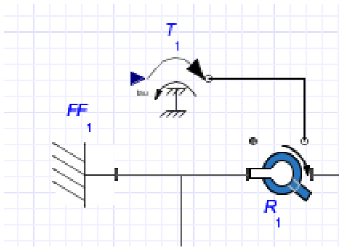
12. Add another **Absolute Translation** component above the **Connecting Rod** subsystem.

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

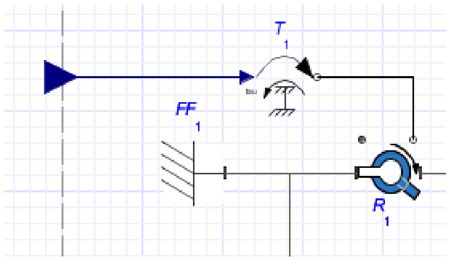
14. Add a **Real Demultiplexer** component to the right of the sensor and connect the components as shown below. Since the crank is moving in the x, y plane, you only need to output the first two signals. You are now ready to add a real input port to your subsystem to control the torque on the crank shaft.



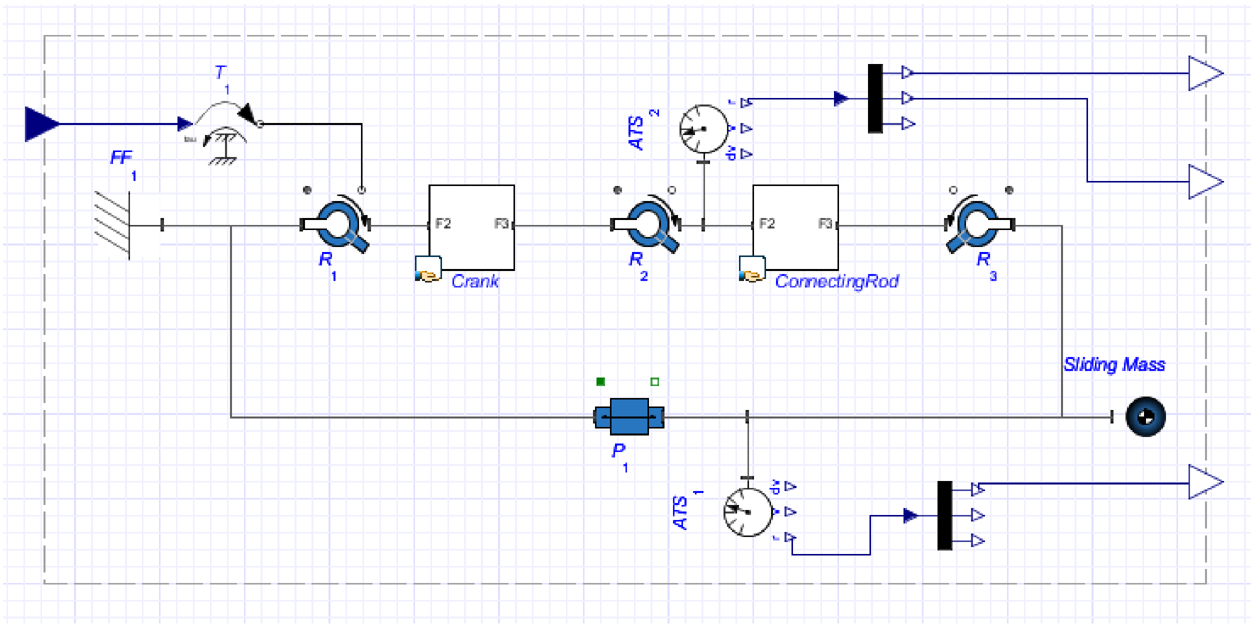
15. From the **1-D Mechanical** > **Rotational** > **Torque Drivers** menu, add a **Torque** component to the **Model Workspace** and place it above the **Fixed Frame** component.
16. Connect the white flange of the **Torque** component to the white flange of the leftmost **Revolute Joint**.



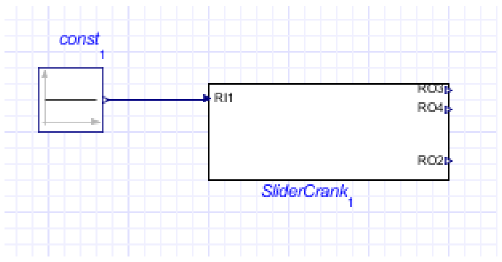
17. Click the input port of the **Torque** component, then drag your pointer to the subsystem boundary and click the boundary once. A real input port is added to your subsystem.



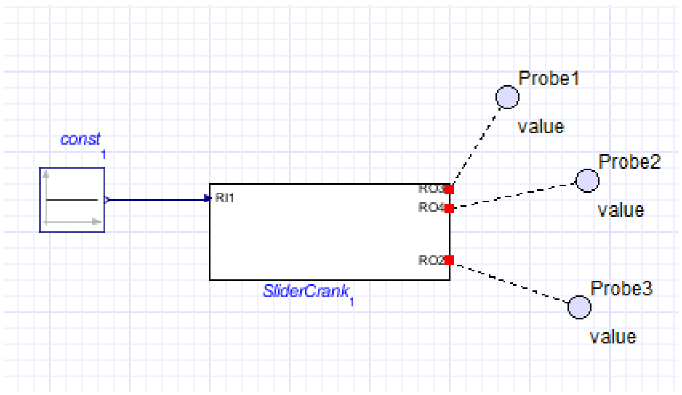
The complete subsystem appears below.



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 **Probe** ( ) above the **Model Workspace** and then click the top output port of the **SliderCrank** subsystem.
21. In the **Model Workspace**, click the probe once to position it.
22. In the same way, add probes to the other **SliderCrank** output ports as shown below.

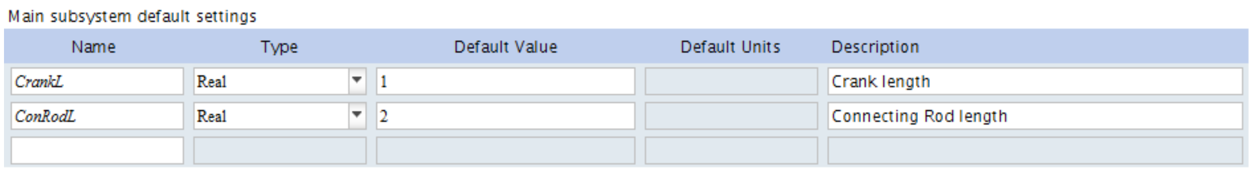


2.2 Defining and Assigning Subsystem Parameters

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

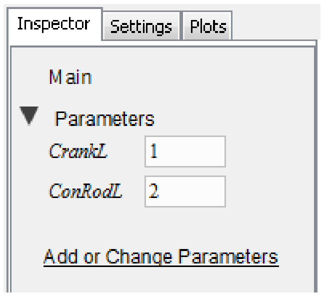
1. While in the detailed view of the **SliderCrank** subsystem, click **Parameters** ( ) above the **Model Workspace**. The parameter editor appears.
2. In the **New Parameter** field, define a parameter called **CrankL** and press **Enter**.
3. Specify a default value of **1** and enter Crank length 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 **Connecting Rod Length** as the description.



Main subsystem default settings

Name	Type	Default Value	Default Units	Description
CrankL	Real	1		Crank length
ConRodL	Real	2		Connecting Rod length

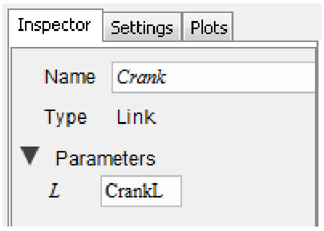
6. Click **Diagram** ( ) to switch to the diagram view. The parameters are defined in the **Parameters** pane.



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.

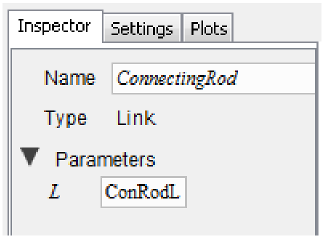


Name *Crank*

Type Link

L CrankL

9. Select the **ConnectingRod** subsystem and change its length value to **ConRodL**.



Name *ConnectingRod*

Type Link

L ConRodL

10. Click **Main** above the **Model 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 EMI block.

2.3 Exporting Your Model Using the FMU file Generation Template

After preparing the model, you can use the FMI file Generation template to set export options and convert the model to an FMU file.

To generate an FMU file

1. Click **Templates** ( ) in the **Main Toolbar**.
2. From the list, select **FMU Generation**. The **Create Attachment** window appears.
3. In the **Attachment** field, enter **Slider Crank FMI** as the worksheet name and click **Create Attachment**. The slider-crank model opens in the **Subsystem Selection** window in the **FMI Connector Template**.
4. From the **Subsystem Selection** toolbar select the **SliderCrank** subsystem and click **Load Selected Subsystem**. All of the template fields are populated with information specific to the subsystem.
5. Specify the location for the **Target Directory**, **Visual C++ Directory** and provide a name for the generated **FMU Archive**.
6. Click **Generate FMU Archive** to generate the .FMU zip file.

Index