FMI Toolbox User's Guide 2.6.4

FMI Toolbox User's Guide 2.6.4

Publication date 2018-07-23

Ideon Science Park

SE-22370 LUND

<info@modelon.com>

Self publishing

Table of Contents

Chapter 1. Introduction

1.1. The FMI Toolbox for MATLAB/Simulink

The FMI Toolbox for MATLAB integrates Modelica-based physical modeling into the MATLAB/Simulink environment. FMI Toolbox offers the following main features:

- Simulation of compiled dynamic models, FMUs, in Simulink. FMUs may be generated by an FMI-compliant tool such as SimulationX or Dymola. The Simulink FMU block offers configuration of parameter and start values as well as block outputs. The Simulink import supports FMI version 1.0 and 2.0 for both Model Exchange and Co-Simulation FMUs.

- Export of Simulink models to FMUs. FMUs may be simulated in FMI compliant simulation tools such as SimulationX or Dymola. Export is supported for FMI version 1.0 and 2.0 for Model Exchange and Co-Simulation. Requires Simulink Coder and FMI Toolbox Coder add-on.

- Simulation of compiled dynamic models, FMUs, using MATLAB's built in integrators (e.g., ode45 and ode15s). This feature makes FMI Toolbox useful also for users without access to Simulink.

- Static and dynamic analysis of FMUs through design-of-experiments (DoE) functions for optimization, calibration, control design, and robustness analysis. The dynamic analysis features require the MATLAB Control System Toolbox.

- The FMI Toolbox supports FMI version 1.0 and 2.0 import in MATLAB for both Model Exchange and for Co-Simulation. DoE analysis is supported for Model Exchange 1.0.

- FMU blocks are supported by Simulink Coder/Real-Time Workshop. It is possible to build a Simulink model containing an FMU block and run it on e.g dSPACE´s DS1006 platform for HIL (hardware-in-the-loop) simulations.

1.2. The Functional Mock-up Interface

The Functional Mock-up Interface is a standard for exchange of compiled dynamic models, and is intended to promote model reuse and tool interoperability. Several tools provide export of Functional Mock-up Units (FMUs), all of which can be used with the FMI Toolbox for MATLAB. FMI provides two different formats for exchange of models:

- FMI for Model Exchange (FMI-ME). The FMI-ME specification is based on a continuous-time hybrid Ordinary Differential Equation (ODE) representation. The FMU-ME provides inputs and outputs and exposes functions for setting parameters and computing the derivatives of the ODE. Environments importing FMU-MEs need to provide an integrator, or ODE solver, that integrates the dynamics of the model.

- FMI for Co-Simulation (FMI-CS). The FMI-CS specification provides a model representation where both the model and an integrator (ODE solver) is encapsulated inside the FMU-CS. Similar to the FMI-ME, the FMU-CS provides inputs and outputs and means to set model parameters. It also provides a function to integrate the dynamics of the model for a specified time interval. Environments importing FMU-CS' therefore do not have to provide an integrator.

The FMI Toolbox for MATLAB supports import for both the Model Exchange and Co-Simulation specifications. Simulink models can be exported as Model Exchange or Co-Simulation FMUs.

Chapter 2. Installation

2.1. Supported platforms

The FMI Toolbox for MATLAB is supported on Windows 7, Windows 10, and Ubuntu 14.04 (Trusty Tahr). There are 2 different installers, one for Windows (32- and 64-bit) MATLAB and one for Linux (64-bit) MATLAB. The installer's name indicates which one of these it is. FMIToolbox is supported for MATLAB 2010b or later on Windows and for MATLAB 2015a to MATLAB 2017b on Ubuntu 14.04 (64-bit).

Make sure you install the right one by typing `computer` in the MATLAB **Command Window** to get the computer type on which MATLAB is executing. Then look in table below to find the right installer to use.

```
>> computer

ans =

PCWIN
```

In this example, the installer `FMI Toolbox-X.X-win.exe` should be used. `X.X` are the placeholder the version numbers of FMI Toolbox installer.

Table 2.1 Different installers for MATLAB

Computer type on which MATLAB is executing	Installer to use
PCWIN	`FMI Toolbox-X.X-win.exe`
PCWIN64	`FMI Toolbox-X.X-win.exe`
GLNX86	`FMI-Toolbox_X.X_linux.tar.gz`
GLNXA64	`FMI-Toolbox_X.X_linux.tar.gz`

2.2. Prerequisites

Please make sure that all prerequisites are fulfilled before installing the product.

2.2.1. MATLAB/Simulink

Verify that your software version is amongst those supported in the tables below.

2.2.1.1. FMU import

On Windows, all MATLAB versions from 2010b to 2018a are supported. On Linux (Ubuntu) MATLAB versions between 2015a and 2017b are supported. This table lists the MATLAB/Simulink versions that are tested for import and simulation of FMUs.

Table 2.2 Supported MATLAB/Simulink

MATLAB version (Simulink version)	Supported on Windows	Supported on Linux
MATLAB 9.4 - R2018a (Simulink 9.1)	Yes	No[a]
MATLAB 9.3 - R2017b (Simulink 9.0)	Yes	Yes
MATLAB 9.2 - R2017a (Simulink 8.9)	Yes	Yes
MATLAB 9.1 - R2016b (Simulink 8.8)	Yes	Yes
MATLAB 9.0 - R2016a (Simulink 8.7)	Yes	Yes
MATLAB 8.6 - R2015b (Simulink 8.6)	Yes	Yes
MATLAB 8.5 - R2015a (Simulink 8.5)	Yes	Yes
MATLAB 8.4 - R2014b (Simulink 8.4)	Yes	No
MATLAB 8.3 - R2014a (Simulink 8.3)	Yes	No
MATLAB 8.2 - R2013b (Simulink 8.2)	Yes	No
MATLAB 8.1 - R2013a (Simulink 8.1)	Yes	No
MATLAB 8 - R2012b (Simulink 8)	Yes	No
MATLAB 7.14 - R2012a (Simulink 7.9)	Yes	No
MATLAB 7.13 - R2011b (Simulink 7.8)	Yes	No
MATLAB 7.12 - R2011a (Simulink 7.7)	Yes	No
MATLAB 7.11.2 - R2010b SP 2 (Simulink 7.6.2)	Yes	No
MATLAB 7.11.1 - R2010b SP 1 (Simulink 7.6.1)	Yes	No
MATLAB 7.11.0 - R2010b (Simulink 7.6)	Yes	No

[a]Due to a glibc bug that exists for the glibc of Ubuntu 14.04 MATLAB 2018a is currently not supported on Linux (Ubuntu 14.04)

The features for dynamic DoE analysis (linearization, bode, and step response plots) require the MATLAB Control System Toolbox.

2.2.1.2. Simulink Coder/Real-Time Workshop

Note: This is **ONLY** required for exporting FMUs from Simulink or when Simulink Coder/Real-Time Workshop is used to build models that contains an FMU block. Verify that your Simulink Coder/Real-Time Workshop and target compiler is amongst those supported, see Table 2.3 and Table 2.4 respectively.

In order to build Simulink models with the target rti1006.tlc that contains the FMU blocks, the FMU must contain it's source code. For a list of FMUs that has been tested, see Table 2.6. For a list of targets that are supported, see Table 2.5. For more even more details see, Section 3.4.

Note: It is not yet possible to build Simulink models that contains the FMU blocks on Linux.

Table 2.3 Supported Simulink Coder/Real-TimeWorkshop

Simulink Coder (MATLAB version)	Supported on Windows	Supported on Linux
Simulink Coder 8.14 (R2018a)	Yes	No[a]
Simulink Coder 8.13 (R2017b)	Yes	Yes
Simulink Coder 8.12 (R2017a)	Yes	Yes
Simulink Coder 8.11 (R2016b)	Yes	Yes
Simulink Coder 8.10 (R2016a)	Yes	Yes
Simulink Coder 8.9 (R2015b)	Yes	Yes
Simulink Coder 8.8 (R2015a)	Yes	Yes
Simulink Coder 8.7 (R2014b)	Yes	No
Simulink Coder 8.6 (R2014a)	Yes	No
Simulink Coder 8.5 (R2013b)	Yes	No
Simulink Coder 8.4 (R2013a)	Yes	No
Simulink Coder 8.3 (R2012b)	Yes	No
Simulink Coder 8.2 (R2012a)	Yes	No
Simulink Coder 8.1 (R2011b)	Yes	No
Simulink Coder 8.0 (R2011a)	Yes	No
Real-Time Workshop 7.6.2 (R2010b Service Pack 2)	Yes	No
Real-Time Workshop 7.6.1 (R2010b Service Pack 1)	Yes	No
Real-Time Workshop 7.6 (R2010b)	Yes	No

[a]Due to a glibc bug that exists for the glibc of Ubuntu 14.04 MATLAB 2018a is currently not supported on Linux (Ubuntu 14.04)

Table 2.4 Supported C compilers

Compilers on Windows	Compilers on Linux
Microsoft Visual C++ 2015 (Professional)	GCC 4.7
Microsoft Visual C++ 2013 (Professional)	GCC 4.9
Microsoft Visual C++ 2012 (Professional)	
Microsoft Visual C++ 2010 (Professional & Express)	
Microsoft Visual C++ 2008 (Express)	
Microsoft Visual C++ 2005 (Professional & Express)	

Table 2.5 Targets supported and tested with the FMU blocks

Target	Note	Target developer vendor
grt.tlc		MathWorks
grt_malloc.tlc		MathWorks
rsim.tlc		MathWorks
rtwsfcn.tlc		MathWorks
fmu_me1.tlc		Modelon
fmu_me2.tlc		Modelon
fmu_cs1.tlc		Modelon
fmu_cs2.tlc		Modelon
rti1006.tlc	Tested for dSPACE´s RCP & HIL releases: **7.2** with service pack 1 and **2013-A** with service pack 1.	dSPACE

Table 2.6 Source code FMUs tested and supported by the FMU block

FMU Generation tool
Dymola 2014 FD01
Dymola 2014
Dymola 2015 FD01 - Refresh - 20141218
Dymola 2016
Dymola 2016 FD01
Dymola 2017
Dymola 2017 FD01

2.3. Installation procedure

The following steps are needed to install and enable FMI Toolbox in MATLAB.

2.3.1. For Windows

1. Double click on the *FMI Toolbox-X.X_win.exe* installer to run it and follow the installation instructions.

   After the installation has completed you will find a folder for the FMI Toolbox in the Windows Start menu. From the Start menu the User's Guide in PDF format can be reached and an uninstaller for FMI Toolbox.

2. To enable the toolbox in MATLAB, a search path to the MATLAB folder in the FMI Toolbox installation folder must be added. To add this path in MATLAB, see Section 2.3.3.

3. Without a license file installed, FMI Toolbox will run in Demo mode. Please read, Section 2.4, for further information.

2.3.2. For Linux

1. The FMI Toolbox for Linux comes in a **.tar.gz* file. This is a compressed archive file. To decompress and extract the files, open a terminal window. Go to the folder where the `*.tar.gz` file is found, in this example it's on the `Desktop`.

   ```
   ba@ba-desktop:~$ cd $HOME/Desktop
   ```

2. Extract the `*.tar.gz` file using tar with the `-xvf` option followed by the name of the `*.tar.gz` file and then the `-C` option followed by the installation folder. In this example the FMI Toolbox is installed in the `$HOME` folder. *Notice, this command overwrites existing files, but does not remove files, i.e `license.lic` is not changed.*

   ```
   ba@ba-desktop:~/Desktop$ tar -xvzf FMI_Toolbox-X.X-linux.tar.gz -C $HOME/
   ```

   This creates a new folder called `Modelon` in `$HOME` where all the files are put in.

2.3.3. Set MATLAB path

The MATLAB path must include the path to the FMI Toolbox installation folder in order to enable the toolbox. To to this, follow the procedure below.

1. In MATLAB change current directory to the installation folder of FMI Toolbox, e.g:

   ```
   >> cd 'C:\Program Files (x86)\Modelon\FMI Toolbox X.X'
   ```

2. Run the setup function to set the MATLAB path

   ```
   >> setup
   ```

2.3.4. Unattended installation

2.3.4.1. Windows

The Windows installer can be run unattended from the command prompt. The installer takes the arguments in Table 2.7.

Table 2.7 Installer arguments

Arguments	Description
/NCRC	Disables the CRC check, unless CRCCheck force was used in the script.
/S	Runs the installer or uninstaller silently.

Arguments	Description
/D	Sets the default installation directory ($INSTDIR), overriding InstallDir and InstallDirRegKey. It must be the last parameter used in the command line and must not contain any quotes, even if the path contains spaces. Only absolute paths are supported.

MATLAB can be run from the command prompt, for more information see http://www.mathworks.se/help/matlab/ref/matlabwindows.html. This can be used to setup the MATLAB path and install a license file.

In the example below, the installer is run from the command prompt. MATLAB is then started with a command line argument that runs inside MATLAB. This argument changes current directory in MATLAB to where FMI Toolbox is installed and then runs the *setup* function. *setup* updates and saves the MATLAB path and installs the provided license file.

```
>"FMI Toolbox-X.X-win.exe" /S /D=C:\Program Files (x86)\Modelon\FMI Toolbox X.X
>matlab -r "cd 'C:\Program Files (x86)\Modelon\FMI Toolbox X.X';setup('C:\temp\license.lic', true);exit;"
```

For more information on how to install the license file, see the help for the *setup* function in MATLAB.

Note: The installer must run with administrator privileges to eliminate the User Account Control dialog that otherwise appear when the installer is started.

Note: MATLAB must run with administrator privileges to make sure that the MATLAB path is properly saved.

2.3.4.2. Linux

The unattended installation procedure for FMI Toolbox on Linux is done in the following steps:

1. Extract the **.tar.gz* file to the installation folder, see Section 2.3.2 for more information.

2. Update MATLAB path to include the FMI Toolbox files.

MATLAB can be run from the command prompt, for more information see http://www.mathworks.se/help/matlab/ref/matlabunix.html. This can be used to setup the MATLAB path and install a license file.

In the example below, the installer is run from the command prompt. MATLAB is then started with a command line argument that runs inside MATLAB. This argument changes current directory in MATLAB to where FMI Toolbox is installed and then runs the *setup* function. *setup* updates and saves the MATLAB path and installs the provided license file.

```
>tar -xvzf FMI_Toolbox-X.X-linux.tar.gz -C <install_dir>
>matlab -r "cd '<install_dir>/Modelon/FMI_Toolbox_X.X';setup('<license_file_dir>/license.lic', true);exit
```

For more information on how to install the license file, see the help for the *setup* function in MATLAB.

Note: MATLAB must run with administrator privileges to make sure that the MATLAB path is properly saved.

2.4. License information

The section references from below are part of Modelons license instructions for the Flex enabled products in Chapter 9.

For instruction on how to retrieve a license file, see Section 9.1.

For instructions on how to install a license file, see Section 9.2.

For instructions on how to install a license server, see Section 9.3.

For trouble shooting and contacting Modelon support, see Section 9.4.

To use full version of FMI Toolbox, a **FMI Toolbox license** is required. To use the FMU export from Simulink, a **FMI Toolbox Coder add-on license** is also required. The licenses has a linger time of 2 minutes. Linger time means that the license stays checked out for the specified period of time beyond its check in. The license is checked in when the MATLAB session is closed. This is a safety precaution in case that a MATLAB session is accidentally closed or purposely restarted so that no other user checks out the license in between the MATLAB sessions.

Note that there is a function in MATLAB that can print useful license information, `fmitoolbox_license`. Use the help command in MATLAB to get further information.

2.4.1. Demo mode

Running the program in demo mode limits the FMUs that can be used to the ones that are distributed as examples.

2.5. Uninstallation procedure

2.5.1. For Windows

FMI Toolbox provides an uninstaller. The following steps uninstalls the FMI Toolbox.

1. Make sure that MATLAB is closed so that all files can be removed.
2. Run the uninstaller that is found in the start menu or in the installation directory.
3. To complete the uninstallation, the search paths for FMI Toolbox in MATLAB must be manually removed. Next time you start MATLAB, a list of missing path folder will appear in the command window. In MATLAB, open the **Set Path...** dialog in the **File** menu and click **Save**.

2.5.2. For Linux

FMI Toolbox does not provide an uninstaller for Linux. The following steps uninstalls the FMI Toolbox.

1. Make sure that MATLAB is closed so that all files can be removed.

2. Remove the files in the installation folder using the *rm -rf* command in the terminal window.

   In this example the FMI Toolbox was installed in the *$HOME* directory. To remove the whole Modelon folder, type:

   ```
   > cd $HOME
   > rm -rf Modelon
   ```

   In order to keep the license file in the *Modelon/Common* folder, do only remove the FMI Toolbox files with the command:

   ```
   > cd $HOME
   > rm -rf Modelon/FMI_Toolbox-X.X/
   ```

3. To complete the uninstallation, the search paths for FMI Toolbox in MATLAB must be manually removed. Next time you start MATLAB, a list of missing path folder will appear in the command window. In MATLAB, open the **Set Path...** dialog in the **File** menu and click **Save**.

2.5.3. Unattended uninstallation

2.5.3.1. Windows

The Windows uninstaller can be run unattended from the command prompt. The uninstaller is found in the installation directory and must be run with the silent flag */S*. Before the uninstaller is started, make sure that FMI Toolbox is not used. The MATLAB path should be updated before the uninstaller is run. The following example demonstrates how to uninstall FMI Toolbox:

```
> matlab -r "cd 'C:\Program Files (x86)\Modelon\FMI Toolbox X.X';remove;exit;"
> "C:\Program Files (x86)\Modelon\FMI Toolbox X.X\Uninstall.exe" /S
```

MATLAB is first started with a command line argument that runs inside MATLAB. This argument changes current directory in MATLAB to where FMI Toolbox is installed and then runs the *remove* function. *remove* updates and saves the MATLAB path.

The uninstaller is then run and removes all the files in the installation folder.

Note: The installer must run with administrator privileges to eliminate the User Account Control dialog that otherwise appear when the installer is started.

2.5.3.2. Linux

The unattended uninstallation procedure for FMI Toolbox on Linux is done in the following steps:

1. Remove the FMI Toolbox path from the MATLAB path.

2. Remove the files in the installation folder using the *rm -rf* command.

The following example demonstrates how to uninstall FMI Toolbox:

```
> matlab -r "cd '$HOME/Modelon/FMI_Toolbox-X.X';remove;exit;"
> rm -rf $HOME/Modelon/FMI_Toolbox-X.X/
```

MATLAB is first started with a command line argument that runs inside MATLAB. This argument changes current directory in MATLAB to where FMI Toolbox is installed and then runs the *remove* function. *remove* updates and saves the MATLAB path.

The files are then removed.

2.6. Support

Support inquires are sent to support@modelon.com.

Chapter 3. Simulation with Simulink

3.1. Introduction

An FMU is a file containing functions for evaluation of the equations of a model. An FMU can be generated by an FMI-compliant tool such as Dymola. An FMU model can be simulated in Simulink using an FMU block. The FMU block loads an FMU model and can then be configured from the **FMU setup window**. The FMU block can have input and output ports that makes it possible to incorporate the FMU model with other Simulink blocks. In the next section, the basic steps for simulating an FMU model in Simulink are demonstrated. The functionality is intuitive but it can be helpful to go through two simple examples showing some combinations of the functionalities and how they can be used.

Generated FMUs for the FMI for Model Exchange (1.0/2.0) and the FMI for Co-simulation (1.0/2.0) standards are supported. The FMI Toolbox has two Simulink blocks, one for each FMU kind (Model Exchange and Co-Simulation). The blocks are partially inlined S-functions and are then supported by Simulink Coder/Real-Time workshop.

3.2. Getting started

This tutorial gives a walk-through of the steps required to simulate an FMU using Simulink. In this walk-through, the Model Exchange block loaded with FMI 1.0 FMUs is used but all the steps looks the same for other FMUs if nothing else is mentioned.

1. Create a new Simulink model.

   Start the **Simulink Library Browser** from MATLAB using the command:

   ```
   >> simulink
   ```

   and create a new model by clicking the **New Model** buttom. A new **Simulink model window** will appear.

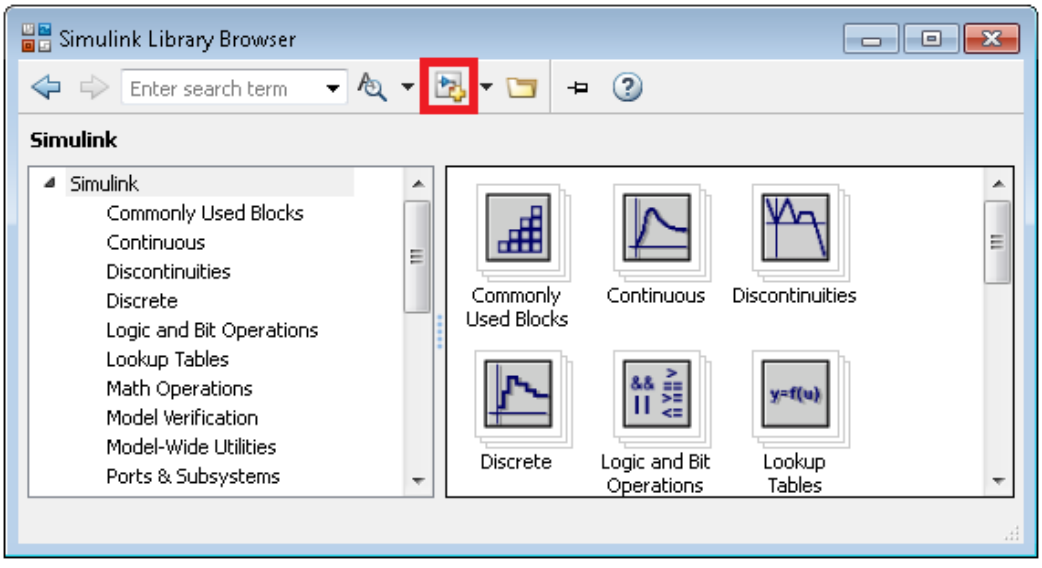


Figure 3.1 Create a new Simulink model.

2. Add the FMU block to the Simulink model.

   In the **Simulink Library Browser,** locate and select the FMI block in the tree view on the left side. On the right side, two FMU blocks will appear. One for each FMI kind, `FMU CS` and `FMU ME`.

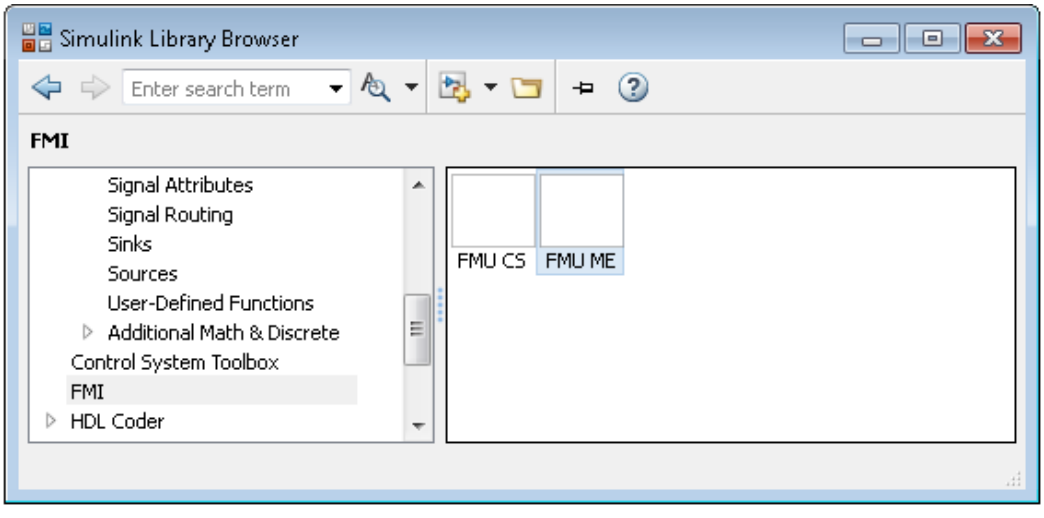


Figure 3.2 Locate the FMU block in Simulink Library Browser.

Drag the model exchange block into the **Simulink model window**. If the model to simulate is a Co-Simulation FMU, the Co-Simulation block should be used instead.

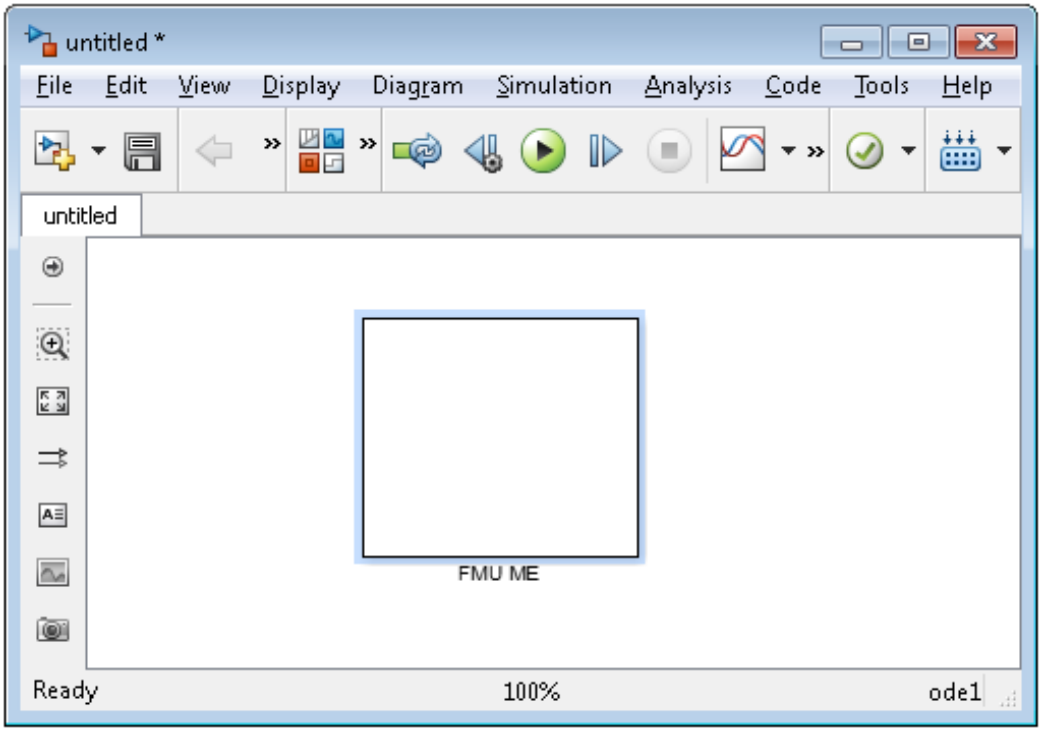


Figure 3.3 Add the FMU block to the Simulink Model.

3. Load an FMU into the FMU block.

   Double-click the FMU block to open the **FMU setup window**.

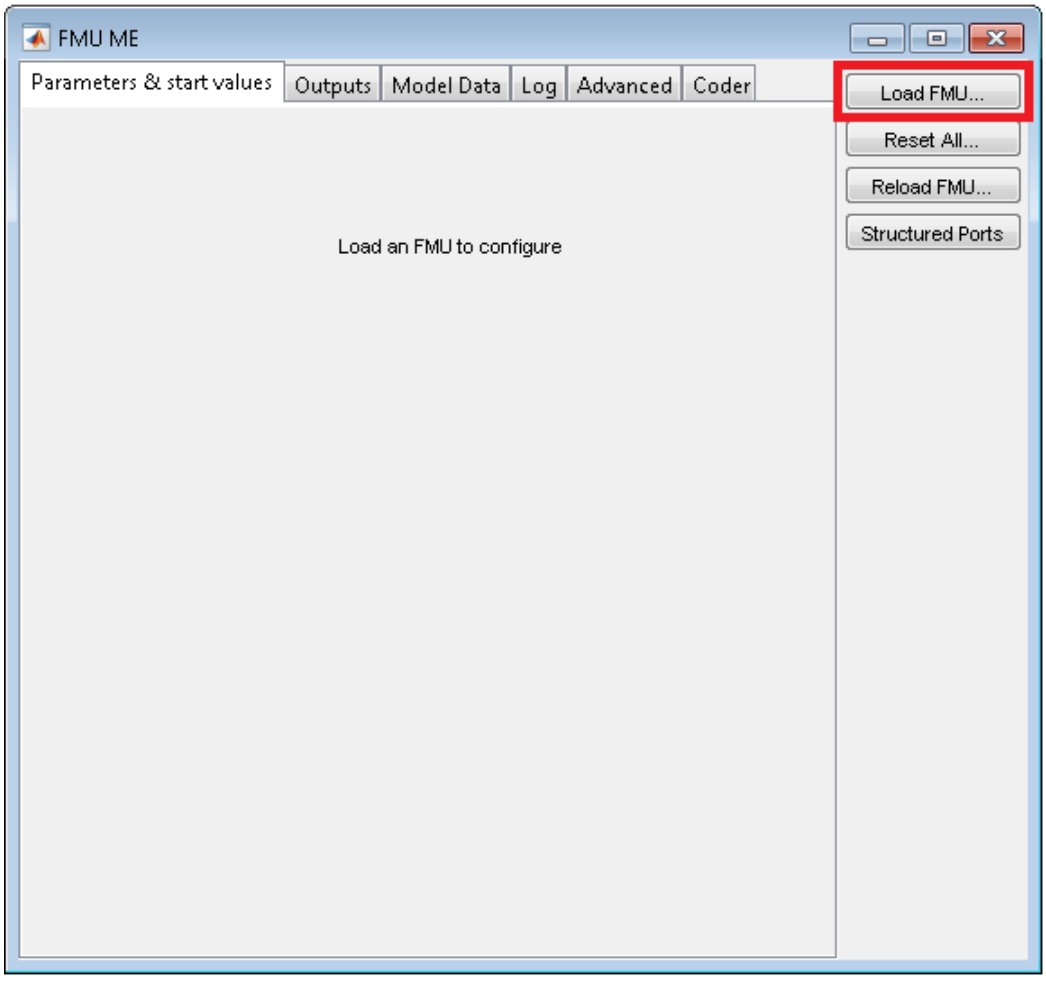


Figure 3.4 Load an FMU model in the FMU setup window.

Click the **Load FMU** button and locate your FMU file in the file browser that will appear. Click **Open** to load the FMU model into the FMU block.

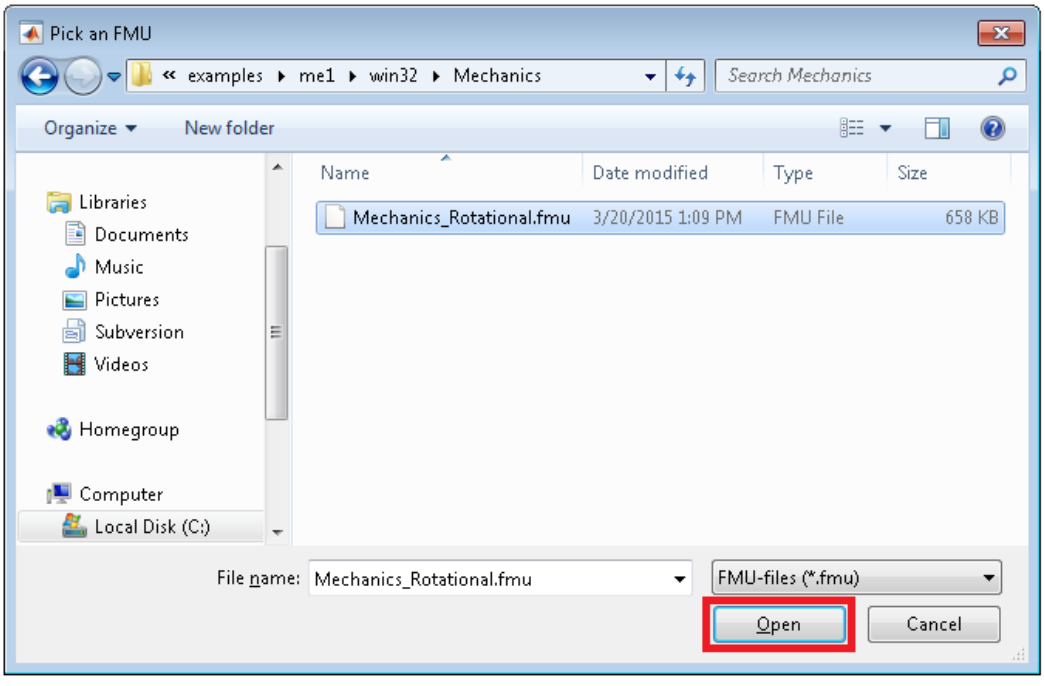


Figure 3.5 Load the FMU block by selecting an FMU model in the file browser.

If the FMU model has top level inputs or outputs they will show up as input and output ports on the block. The example FMU used here, `Mechanics_Rotational.fmu`, has one input and three output ports. The FMU file is found in the installation directory of FMI Toolbox under `examples\me1\<platform>\Mechanics\`.

4. Simulate the model.

   To visualize the simulation results of the `Mechanics_Rotational.fmu`, a `Scope` block and a `Mux` block found in the **Simulink Library Browser** are added to the model. A `Sine Wave` block is also added as source to the input port. If an FMU does not have any top level outputs, other variables can be added as output ports. See section **Output ports** for how to configure the output ports of the FMU block.

   Simulate the FMU model by clicking the **Start simulation** button in the **Simulink model window**.

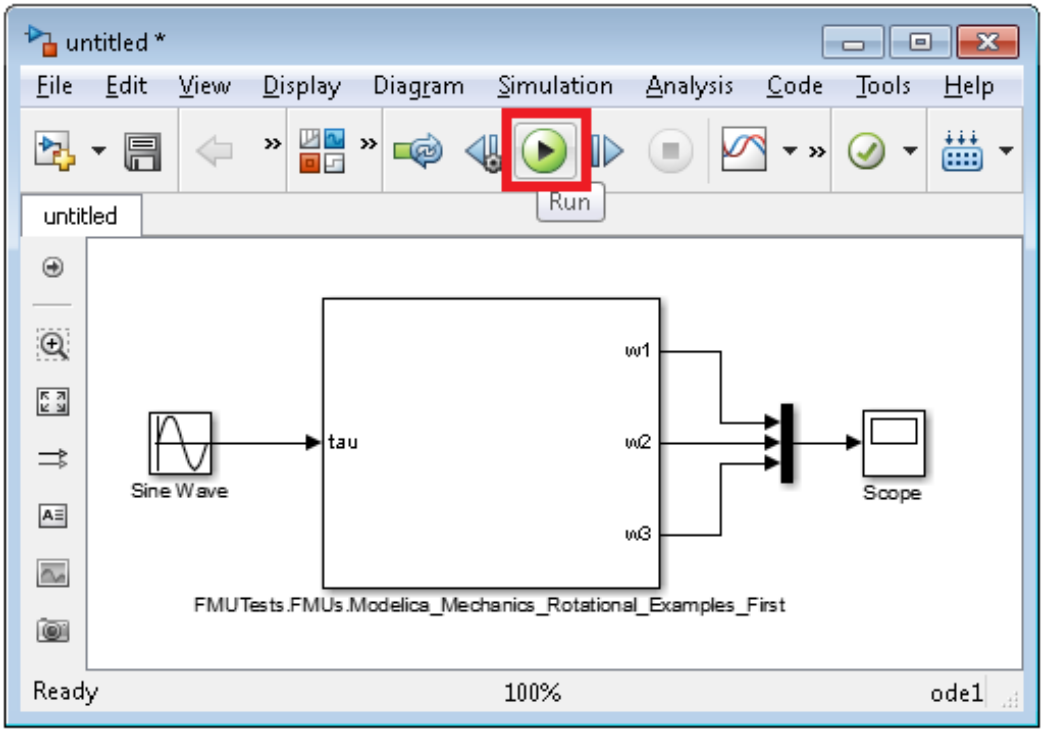


Figure 3.6 Simulate the Mechanics_Rotational FMU with one input signal and three output signals.

The simulation results for the variables `w1`, `w1` and `w1` are visualized in the Scope block. Double-click the `Scope` block to make the `Scope` window appear.

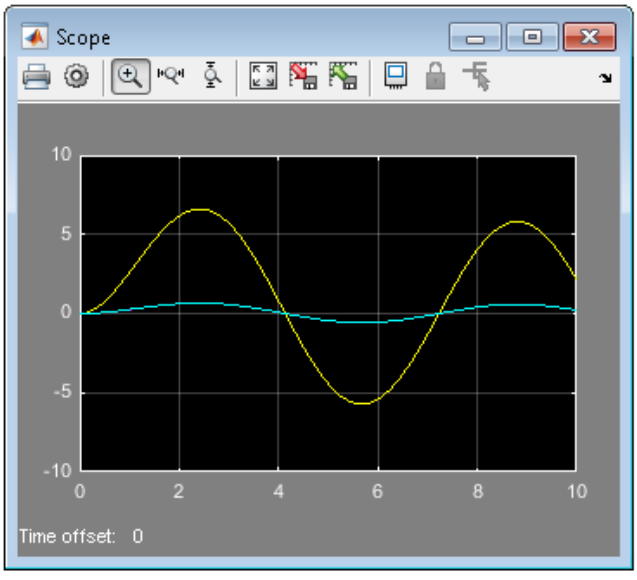


Figure 3.7 Simulink Scope block visualize the results from the FMU above simulated from time 0 to 10.

3.3. FMU block properties

All configurations of the block described in this section are made within the **FMU setup window**. The **FMU setup window** has six tabs, **Parameters & Start Values**, **Outputs**, **Log**, **Model Data**, **Advanced** and **Coder**. Block configurations are made in the **Parameters & Start Values**, **Outputs**, **Log**, **Advanced** and **Coder** tabs whereas general FMU information is contained in the **Model Data** tab. All tabs have the same meaning and general layout for Model Exchange (1.0/2.0) and Co-Simulation (1.0/2.0) FMUs but details may differ, all these differences will be mentioned.

3.3.1. Set parameters and variables start values

In the **Parameters & Start Values** tab, see Figure 3.8, all the parameter and variable information is found. The start values can also be set here. Only variables and parameters with the start attribute set are listed. No inputs are shown.

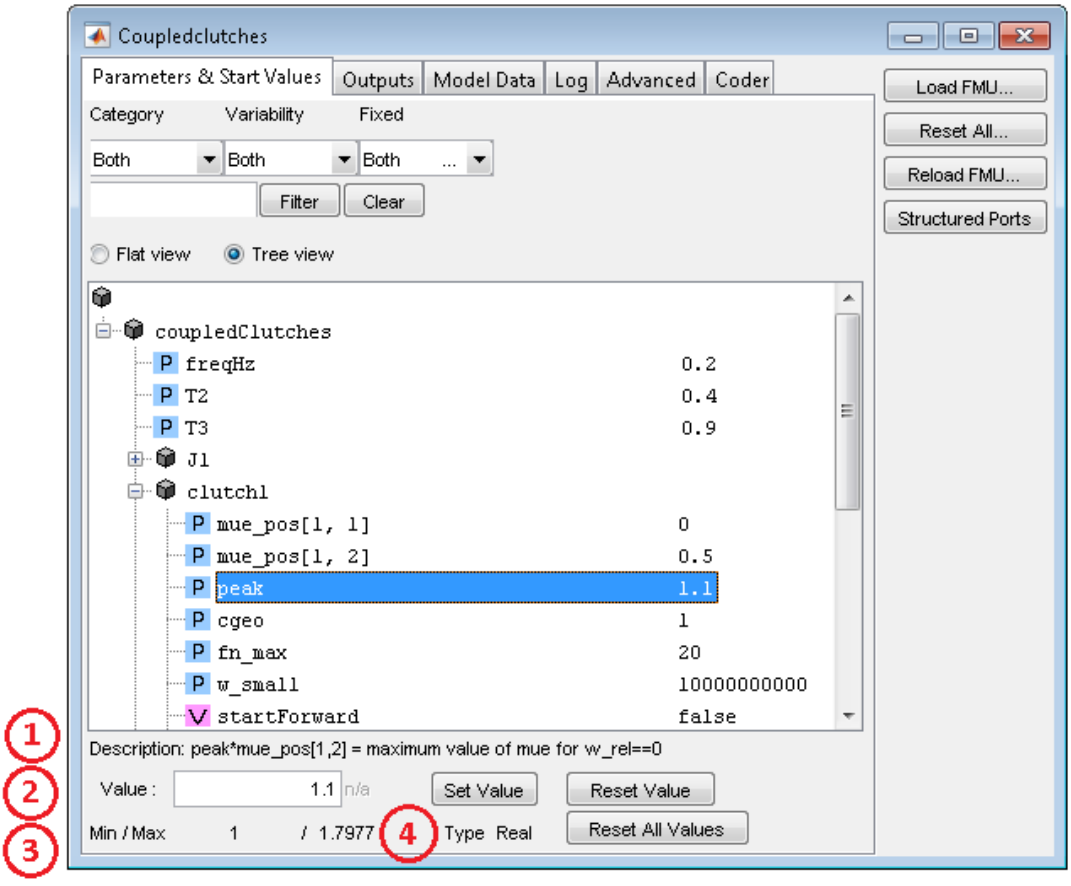


Figure 3.8 Parameters & Start Values tab shows all parameter and variable information.

To access information of a variable, select the variable of interest. Different fields are then populated with information. The descriptions of the different fields are found in the table below.

① Description	Description string of the variable.
② Value	Start value of variable or parameter. If trying to update the value with a different data type, the update is ignored.
③ Min/Max	Minimum value of variable (value $\geq$ min). Maximum value of variable (value $\leq$ max).
④ Type	Real/Integer/Boolean

The variable icons are described in the tables below.

Table 3.1 Variable icons for FMI 1.0 FMUs.

Boolean Integer Real	The colors of the nodes in the lists represent the different data types, Boolean/Integer/Real.
P P P Parameters	A parameter is marked with a **P**. The value of the parameter does not change after initialization.
V V V Variables	A variable is marked with a **V**. A variable is either a continuous or discrete variable. A discrete variable only changes during initialization and at event instants. A continuous variable has no restrictions on value changes.
C C C Constants	A constant is marked with a **C**. The value is fixed and does not change.
V V V **Fixed initial value** V V V **Initial guess value**	The shade of the icons indicates that the variable has either a **Fixed initial value**(light icon) or a **Initial guess value**(dark icon). A **Fixed initial value** has this value after the model has been initialized. A **Initial guess value** may has been changed.

Table 3.2 Variable icons for FMI 2.0 FMUs.

Boolean Integer Real	The colors of the nodes in the lists represent the different data types, Boolean/Integer/Real.
P P P Parameters	A parameter is marked with **P**. The value of the parameter does not change during initialization.
cP cP cP Calculated Parameters	A calculated parameter is marked with **cP**. The value of the parameter does not change after initialization.
L L L Local	A local variable is marked with **L**. A local variable, its value should not be used outside of the FMU.
L P L P L P **Exact initial value** L cP L cP L cP **Guessed initial value** L cP L cP L cP **Calculated initial value**	The shade of the icons indicates that the variable has either an **Exact initial value**(light icon), **Guessed initial value**(slightly darker icon) or a **Calculated initial value**(darkest icon). An **Exact initial value** has this value after the model has been initialized. A **Guessed initial value** may have been changed. A **Calculated initial value** cannot be set and is not showed in this tab, only in the **Output** tab, it is presented here for completion .

To set a start value, select the variable in the list and change the value in the value field next to **Value**. Press **Enter** or the **Set value** button to set the value. A asterix(*) is added to the variable name in the list if the value is different

from the default. The value that is set can be an expression like `sin(x)` where x is a variable in the base workspace. These expressions are evaluated just before the simulation starts.

To reset the value of a variable, select the variable in the list and click **Reset** and the default value will be set.

To reset all start values to the default, click the **Reset All Values** button.

3.3.2. Input ports

The input ports of the block are set according to the top level input variables in the FMU model. These are set when the model is loaded and *can not be changed*. The input ports are scalar and the data type is set to the corresponding type of the variable (integer, real, boolean). The name of the input port on the block is set to the name of the variable.

3.3.3. Output ports

The output ports of the block are by default set according to the top level outputs of the FMU model and is marked with bold text. These can be both scalar or vector output ports. In the **FMU setup window**, **Output** tab, see Figure 3.9, the output ports of the FMU block may be configured.

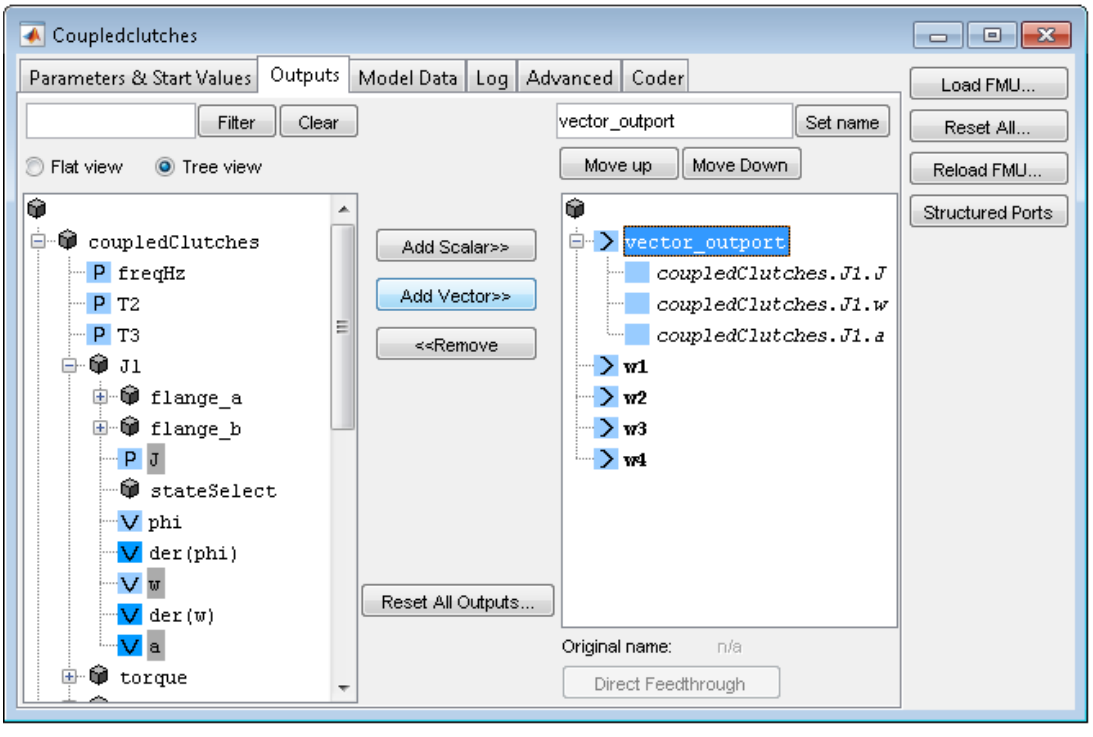


Figure 3.9 Outputs tab in the FMU setup window with 1 vector port and 4 scalar ports.

In the list to the left are all the variables that can be added as output ports on the FMU block. In the list to the right are all the output ports that are currently set. The list to the right contains the variables that are selected as outputs of the FMU block. The corresponding FMU block is shown in the Figure 3.10.

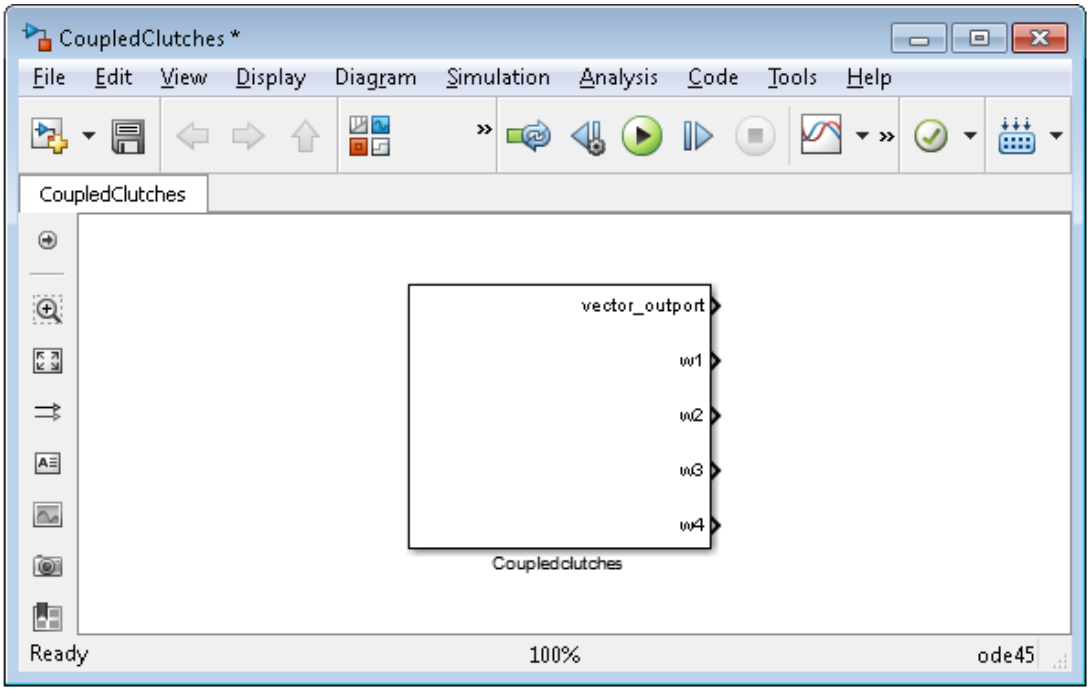


Figure 3.10 Output ports on the FMU block with the output configuration from the Figure above.

The nodes in the list to the right has a hierarchy with two levels. The first level has an icon with a right arrow and represents a port that is seen on the FMU block. If a port has children, they have a blanc icon (level two-nodes). These represent the variables that are contained in that output port. We call a port containing several outputs a *vector port* and a port with no children a *scalar port*. All children in a vector port must be of the same data type.

Ports are added and removed using the **Add >>**, **Add Vector>>...** and **<< Remove** buttons. These buttons are sensitive to which nodes that are selected in the two lists. For example if trying to make an illegal add, the button will get disabled, grayed.

To add a variable as an output port, select the variable in the left list and then select a port in the right list to insert the port after. Click the **Add >>** button to add the port. It is possible to add multiple scalar ports simultaneously by selecting multiple variables in the left list.

To create a vector output, select multiple variables in the left list. All variables must be of the same data type. Select an existing port in the right list after which the new port will be inserted. Then click the **Add Vector >>...**

button and a input dialog box will appear that asks the user to enter a name of the port. Default port name is set to the name of the first variable that was selected. Press **OK** to add the new port.

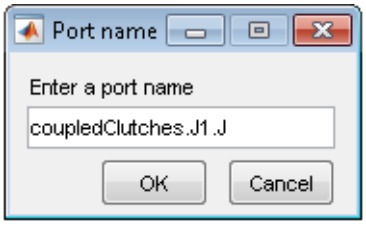


Figure 3.11 Input dialog box asks the user to enter the name of the vector port that will be added.

To add a variable to an already existing output vector port, select a child node in the vector port after which the new variable should be inserted and click the **Add >>** button.

To rename a port, select the port to rename in the list to the right. The port name will appear in the **Rename field** where the name can be changed, and then click the **Set name** button. Renaming a child in a vector output has no visual effect on the FMU block.

To remove a port, select the port to remove and click the **<< Remove** button. You can also remove a single child variable in a vector output.

To change the position of a port on the FMU block, select the port you want to move in the list to the right and use the **Move Up** and **Move Down** buttons.

To reset the output ports to default, use the **Set default outputs** button.

3.3.3.1. Direct Feedthrough

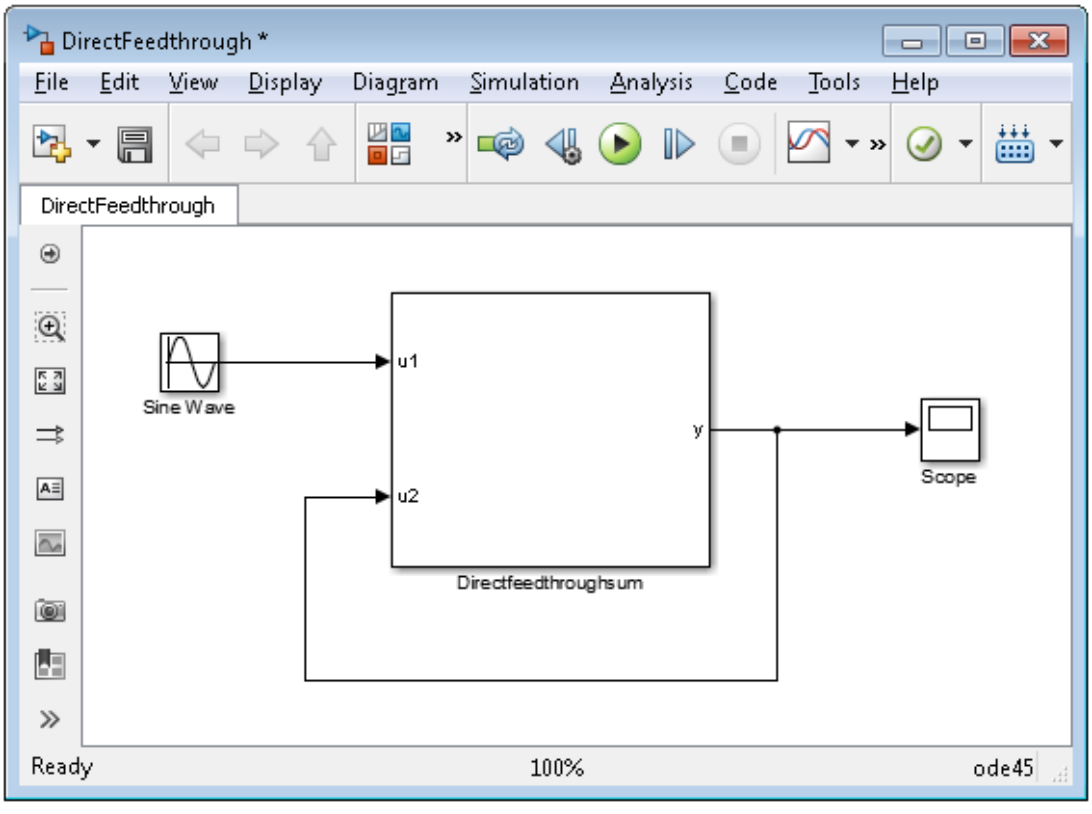


Figure 3.12 System with direct feedthrough.

Direct feedthrough in a Simulink block means that the output port depends directly on the input port. To solve an algebraic loop, the input port must be configured for direct feedthrough. All input ports that are listed in the FMU are set with direct feedthrough. The direct feedthrough concept corresponds to the direct dependency in the FMI documentation. This means that even though all input ports to the FMU block in Simulink can be connected in an algebraic loop, it may not be supported by the FMU. Due to the definition of the direct dependency, the output ports in an FMU, lists the input ports that can be used in an algebraic loop.

The Output tab in the FMU setup window for the system above, is seen in Figure 3.13.

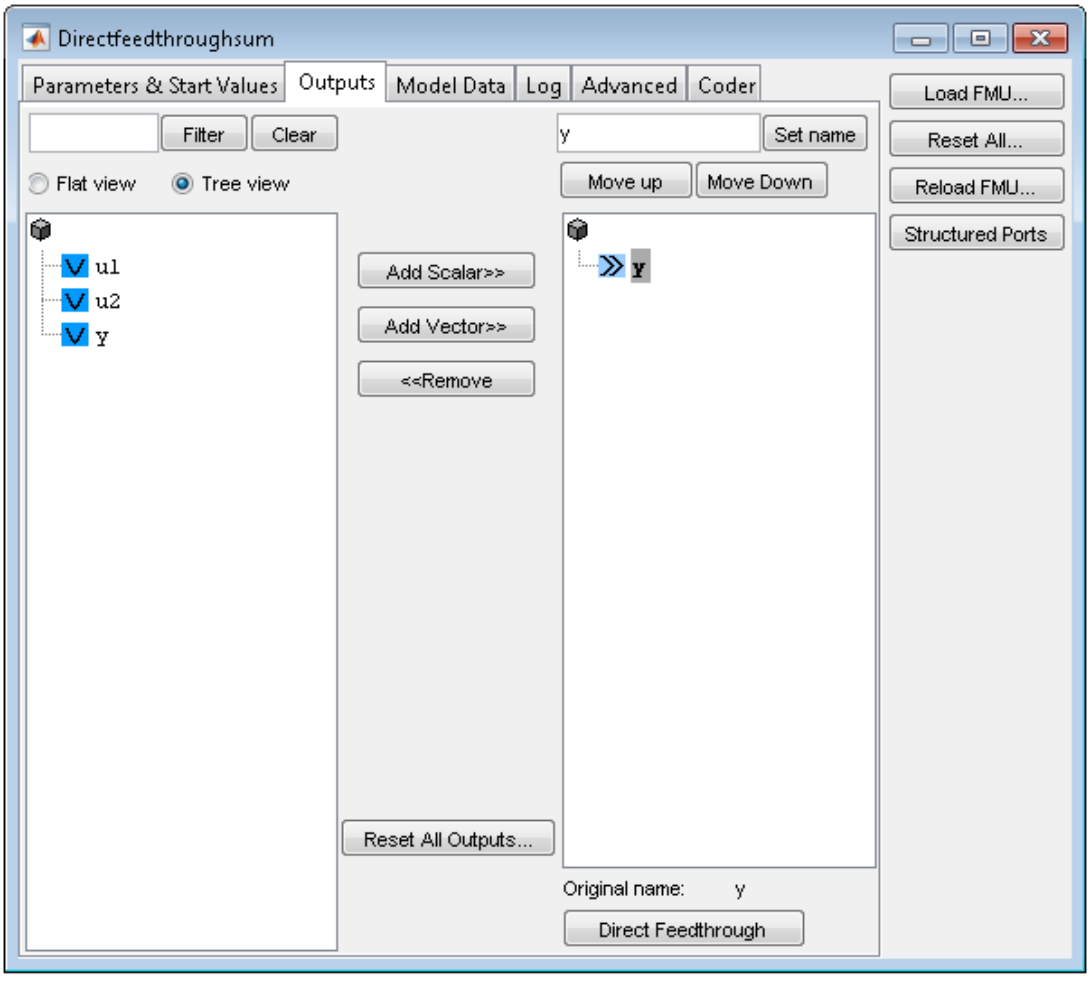


Figure 3.13 The output variable y sets some input with direct feedthrough.

An output port that depends directly on an input port has the icon » with double arrows. This indicates that the output port may be used in an algebraic loop. When an output port with direct feedthrough is selected, the **Direct Feedthrough** button is enabled. This button opens a dialog box, listing the input ports that may be used in an algebraic loop and that the output port depends directly on.

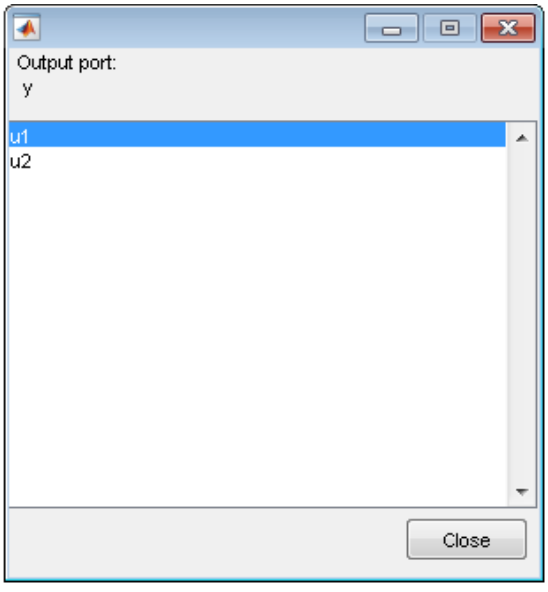


Figure 3.14 Dialog for the selected output port y that lists the input ports u1 and u2 it depends directly on.

3.3.4. FMU model information

In the **Model Data** tab general information of the FMU model is found, see Figure 3.15 and Figure 3.16. The information is extracted from the FMU model and can not be changed.

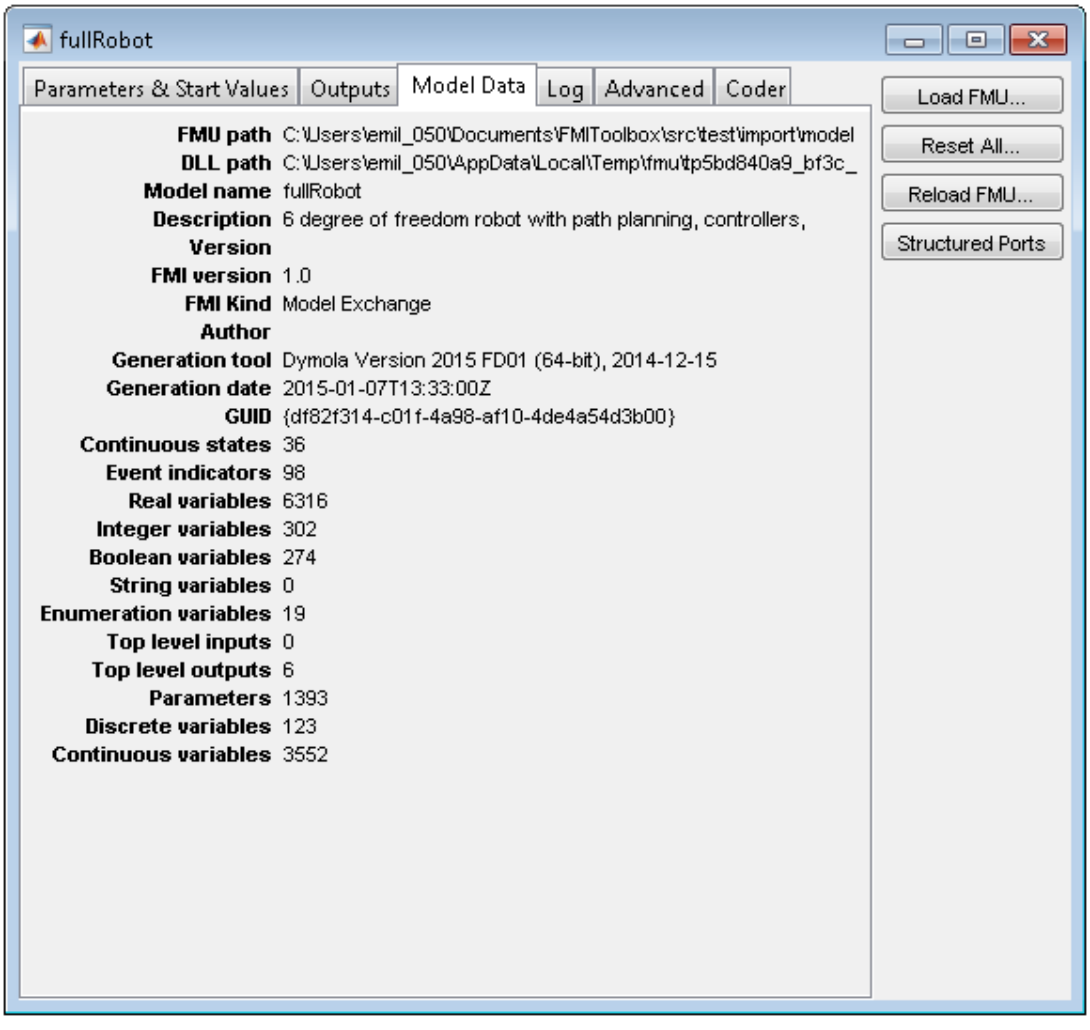


Figure 3.15 The Model Data tab for an FMI 1.0 FMU.

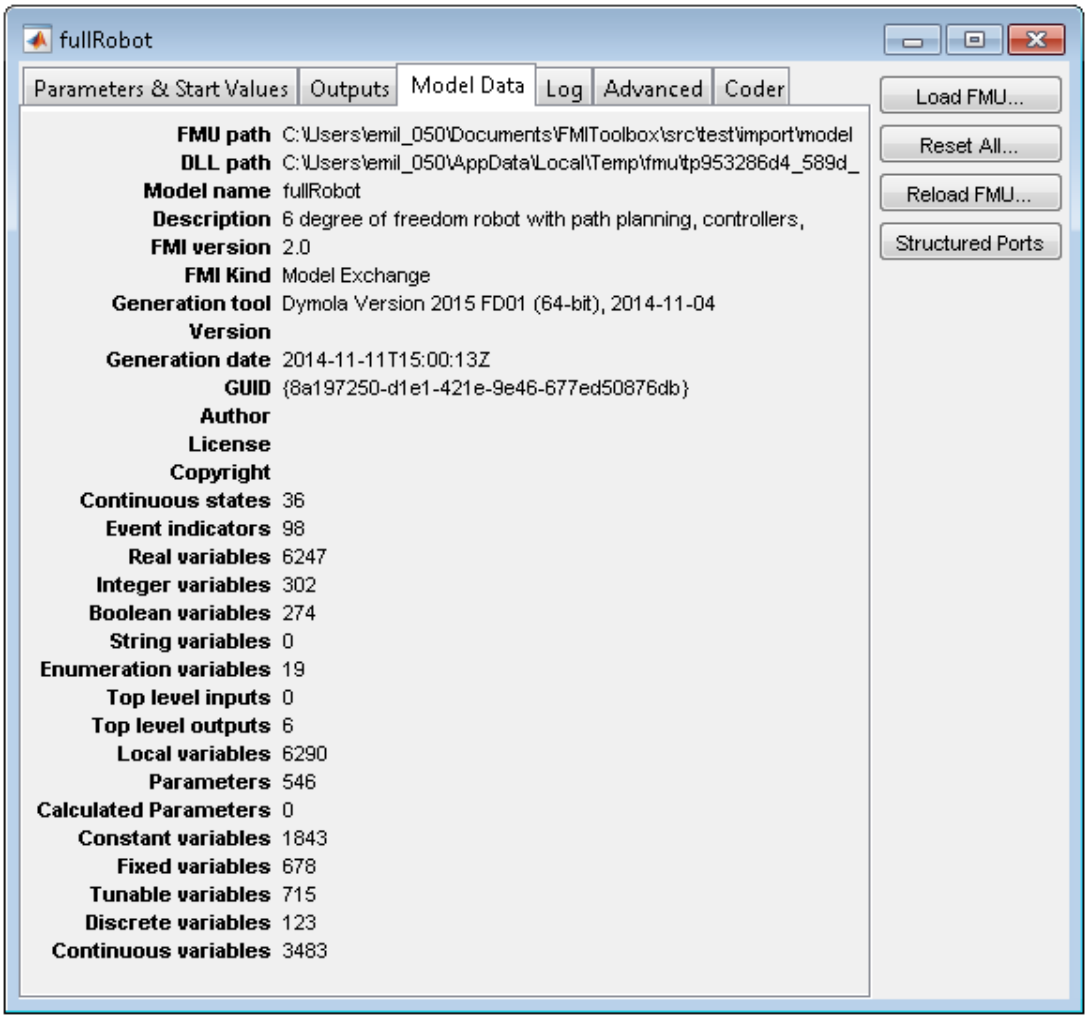


Figure 3.16 The Model Data tab for an FMI 2.0 FMU.

3.3.5. Log

In the **Log** tab the user can configure the logging behavior of the block. In Figure 3.17 the **Log** tab is shown.

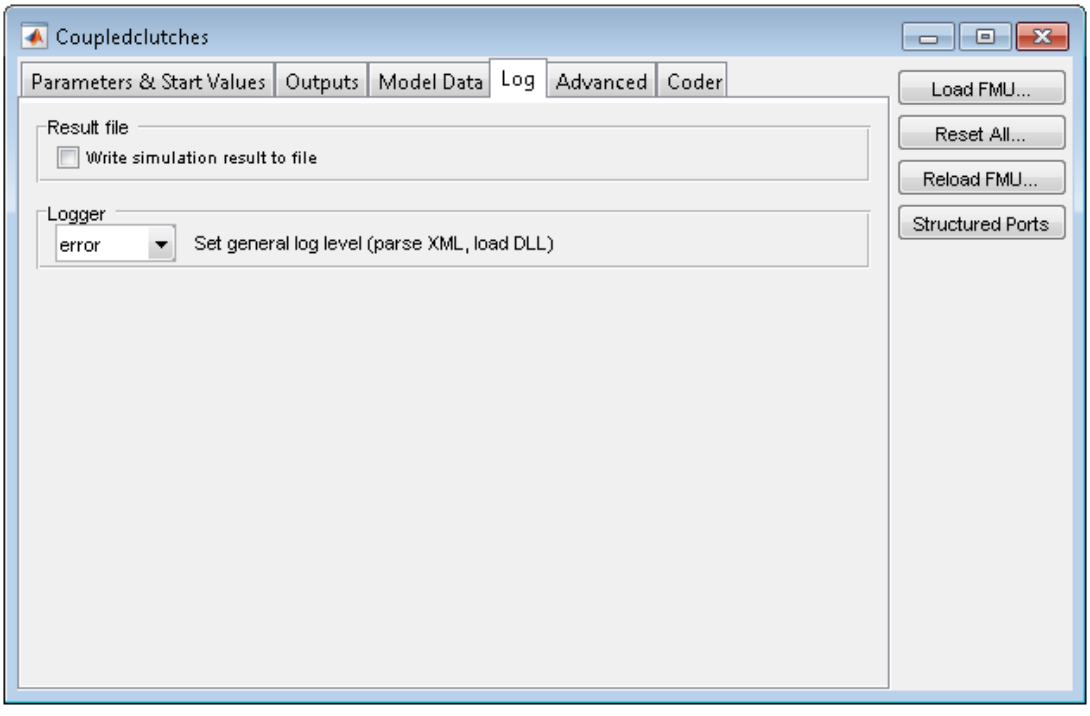


Figure 3.17 The block's logging can be configured from the Log tab.

3.3.5.1. Create result file

A Dymola formated result file in textual format is created if the check box **Write simulation result to file** is checked. Please note that the **Write simulation result to file** option may affect the simulation speed for large models. This file contains the result data from all the variables, parameters and constants.

The result file is created in the current directory and is named to the FMUs model name with the extension `_results.txt`.

The result file can be loaded into the MATLAB workspace using the `loadDSResult` script. To extract the result data of a variable the `getDSVariable` function is used. A short example is given here on how to use these functions.

Assume that an FMU named `CoupledClutches` with the variable `coupledClutches.J2.a` has successfully been simulated. To plot the result of this variable, load the result file in the MATLAB workspace.

```
>> ResData=loadDSResult('CoupledClutches_results.txt');
```

Extract the simulation result of the variable `coupledClutches.J2.a` and plot.

```
>> [T,Y]=getDSVariable(ResData,'coupledClutches.J1.w');
>> plot(T,Y);
```

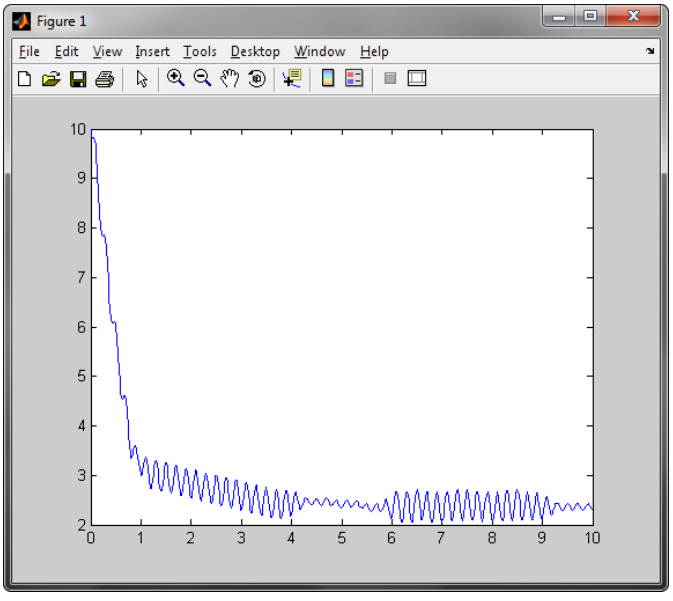


Figure 3.18 Plotted simulation results loaded from a result file that was generated from a Simulink simulation.

3.3.5.2. Logger

The log messages from the FMU are printed to the `Command Window`. In the drop down menu, a list of log levels are listed that can be used to filter the messages. The selected log level prints the selected log level messages and all other messages that has a higher precedence in the list, i.e if **verbose** is selected, all messages are printed since it has lowest precedence. The log levels listed in precedence order starting with the lowest.

1. **verbose**
2. **info**
3. **warning**
4. **error** (default)
5. **fatal**
6. **nothing**

3.3.6. Advanced

In the **Advanced** tab the user can change some of the block configurations regarding the block behavior and simulation settings. In this tab different options are available for the different blocks (`FMU ME` and `FMU CS`) and the

different FMI versions (1.0 and 2.0). The **Tolerances** panel is always available for the `FMU ME` block and for FMI 2.0 FMUs in the `FMU CS` block. The **Sample times** panel is always available in the `FMU CS` block. In Figure 3.19 the Co-Simulation block's advanced tab is shown with an FMI 2.0 FMU loaded.

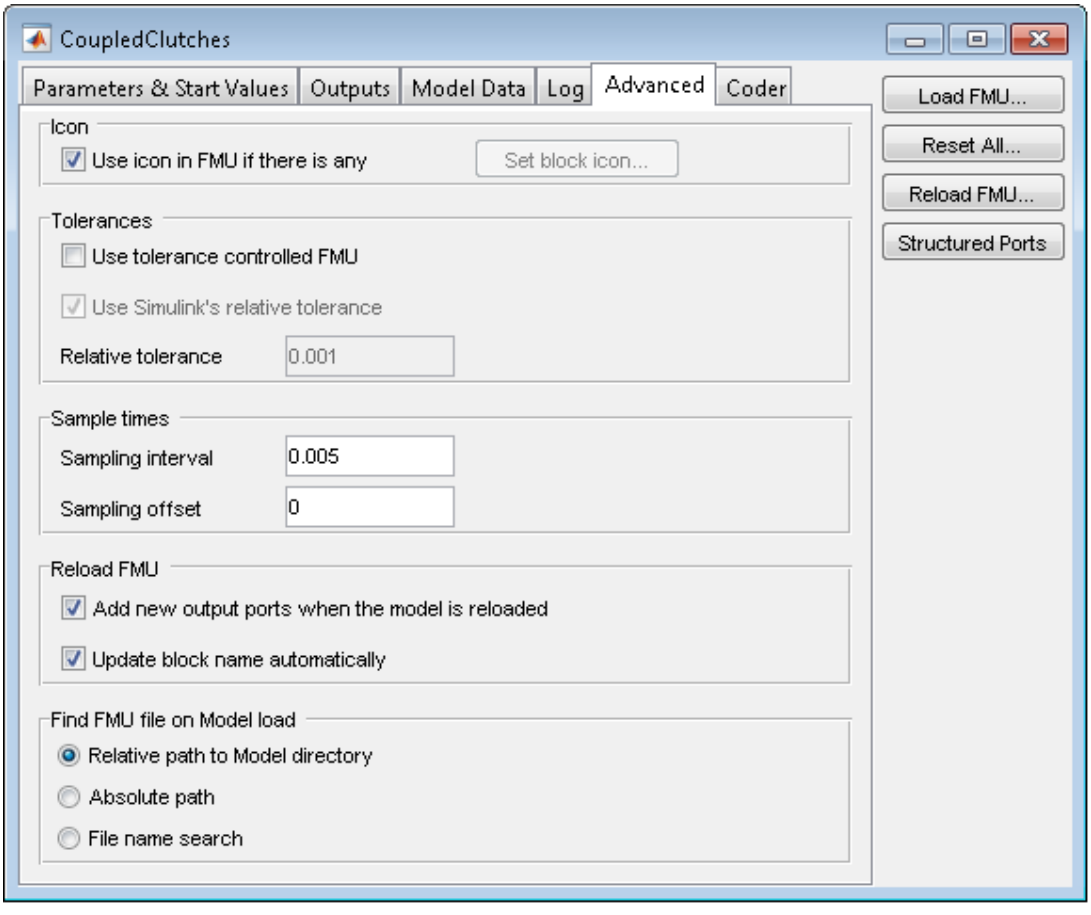


Figure 3.19 The block behavior can be configured from the advanced tab. This is a figure of the FMU CS 2.0 advanced tab.

3.3.6.1. Block icon and mask

The FMU block´s mask is set by a callback function. This function overwrites the `Drawing commands` in the `Mask Editor` every time the block has been changed. To modify the block icon, click the **Set block icon** button and choose your icon image in the file browser that opens. By the default, the check box **Use icon in FMU if there is any** is checked and uses the `model.png` file in the FMU file. If there is no such file, the block is set to be white.

3.3.6.2. Tolerances (Not for FMU CS 1.0)

An FMU model may solve equations internally up to a certain accuracy based on some tolerances. Before the simulation starts, a relative tolerance can be set by the simulation environment to the FMU model. There is also an indicator flag for whether or not the FMU model should use this tolerance. If the FMU model should use the relative tolerance set by the simulation environment, then check the check box **Use tolerance controlled FMU**. By default this it is unchecked. To use Simulink's relative tolerance, check the **Use Simulink´s relative tolerance**. *Note, the relative tolerance is only available when variable step solvers are used*. If the **Use Simulink´s relative tolerance** is unchecked, a relative tolerance can be typed in the **Relative tolerance** field below.

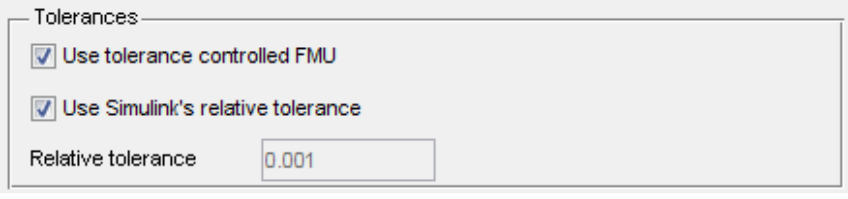


Figure 3.20 The **Tolerance** panel in the Advanced tab for the model exchange block.

3.3.6.3. Sample times (FMU CS block only)

The **Sampling interval** is the length of time steps used when simulating the FMU model. The **Sampling offset** is the delay before the first time step is taken after the simulation started.

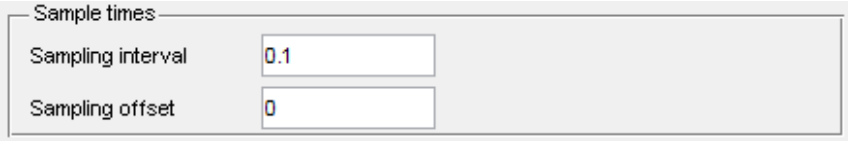


Figure 3.21 The **Sample times** panel in the Advanced tab for the co-simulation block.

3.3.6.4. Reload FMU

When an FMU model is reloaded, new output ports that are added in the FMU model can be added last in the FMU block. This is done if the **Add new output ports when the model is reloaded** option is checked. Default is unchecked.

The block name is updated when an FMU model is loaded or reloaded. It is set to the model name. To disable this automatic update, uncheck **Update block name automatically**.

3.3.6.5. Find FMU file on Model load

There are three different alternatives that can be configured for how the FMU file should be found when the model(*.mdl) is opened.

1. Relative path to the Model directory.

2. Absolute path.

3. File name search.

   This alternative searches for the FMU file in the following order:

   i. Model(*.mdl) directory

   ii. Current working directory

   iii. MATLAB path

Default is that relative path is used.

3.3.7. Coder

The Coder tab is used configure properties for the block that influence the code generation when the model is built for a specific target with Simulink Coder.

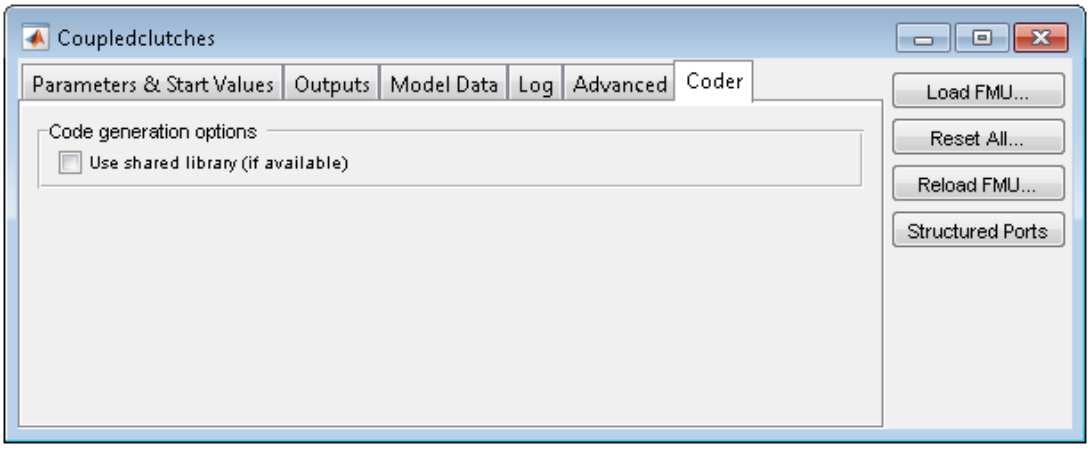


Figure 3.22 The FMU setup window with the Coder tab selected.

Enable **Use shared library** to generate code that loads the shared library of the FMU.

When the target is an FMU, the block´s shared libraries will be contained in the generated FMUs resources. For targets other than FMUs, a resource folder is created in the same directory as the target.

The rti1006.tlc target does **only** work with source code FMUs.

3.3.8. Scripting FMU block

The following functions are available for scripting with the FMU blocks. Use the help command in MATLAB to get more information about the functions.

Table 3.3 Functions available for scripting with the FMU blocks.

Functions	Short description
fmuGetInputPortsSimulink	Returns the input ports from an FMU block.
fmuGetModelDataSimulink	Returns model data.
fmuGetOptionSimulink	Returns an FMU block option.
fmuGetOutputPortsSimulink	Returns the output ports from an FMU block.
fmuGetValueSimulink	Returns the start value for a variable.
fmuLoadFMUSimulink	Loads an FMU block with an FMU file.
fmuReloadFMUSimulink	Reloads an FMU block.
fmuResetAllOutputPortsSimulink	Resets all output ports to default.
fmuResetAllSimulink	Resets all parameter and start values, and all output ports.
fmuResetAllValuesSimulink	Resets all parameter and start values.
fmuResetValueSimulink	Resets one or multiple parameter and start values.
fmuSetOptionSimulink	Sets an FMU block option.
fmuSetOutputPortsSimulink	Sets the output ports for an FMU block.
fmuSetValueSimulink	Sets a parameter or start value in an FMU block.
fmuStructuredPortsSimulink	Uses the structured naming of the ports to make them buses.
fmuRemoveStructuredPortsSimulink	Removes the bus structure added by fmuStructured-PortsSimulink.

Here is an example of a script running a Van der Pol oscillator model with different start values `x1_0` and `x2_0`. The output of the Simulink model `VDP.mdl` are the variables `x1` and `x2`. The phase plane for the Van der Pol oscillator is shown in Figure 3.23.

```
model_name = 'VDP.mdl';             %Simulink model
block_name = 'VDP/VDP';             %FMU block in the Simulink model
simopt = simset('solver','ode15s'); %Solver used
final_time=20;
N_points=11;

x1_0=linspace(-3,3,N_points);
```

```
x2_0=zeros(size(x1_0));

%Simulink model must be open to set parameters and start values
open_system(model_name);

for k=1:N_points
    %Set initial conditions in model
    fmuSetValueSimulink(block_name,'x1_0',x1_0(k));
    fmuSetValueSimulink(block_name,'x2_0',x2_0(k));
    %Simulate
    [T,Y]=sim(model_name,[0,final_time],simopt);
    %Plot simulation results
    plot(Y(:,1),Y(:,2));
    hold on
end
xlabel('x1');
ylabel('x2');
```

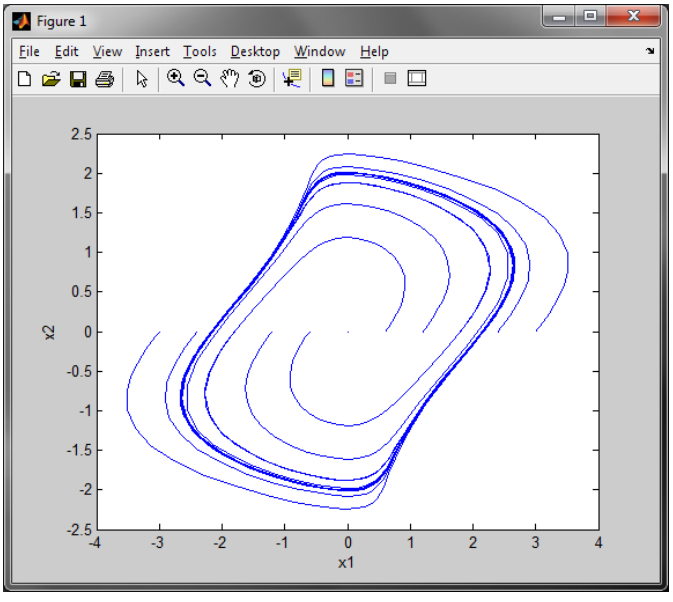


Figure 3.23 Phase plane for the Van der Pol oscillator that was generated from the script above.

3.3.9. Load FMU model

To load an FMU model into the FMU block, open the **FMU setup window** by double clicking the FMU block. Click the **Load FMU** button in the sidebar to the right and a file browser will appear. Locate and select the FMU file to load the FMU block with and click **Open**. If a Simulink model with an FMU block is opened but the path to the FMU file is changed, a file browser will appear that prompts you to relocate the FMU file. The FMU file will then be reloaded.

3.3.10. Reset an FMU model

To reset an FMU model, use the **Reset All** button found in the **Setup** tab. This operation resets all values to default and sets the output ports to default.

3.3.11. Reload FMU model

If an FMU file has been changed it can be reloaded. To reload an FMU, click the **Reload FMU** button. Start values and ports are updated in the new FMU.

This feature is useful if an FMU that has been imported into Simulink has been changed. In this case, the reload feature attempts to keep the configuration of the FMU block even though the updated FMU may have different variables than the original FMU.

3.3.12. Add Structured Ports to the FMU Block

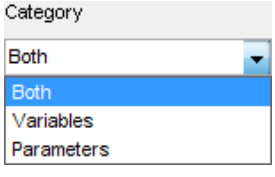
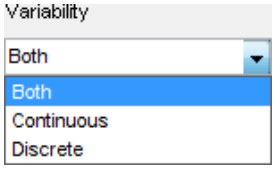
If an FMU Block has inputs or outpus that have a structured naming (model.bus.signal) then this can be used to create a structure of buses based on this naming. To do this click the **Structured Ports** button.

For a practical example see the Configure ports using structural naming example in Section 3.5.3.

3.3.13. Using the filter functions

To ease the use of the variable lists, different filter functions is at hand to search in the lists. A summary of how they work and should be used is described below.

Table 3.4 Filtering for FMI 1.0 variables

	Filter variables by name. Use the **Clear** button to reset the filter function.
	Filter on **Category**. There are two defined categories, Variables and Parameters. Use the drop down list to select the category to filter. Choose between **Parameters**, **Variables** or **Both**. **Variables** can be **Continuous** or **Discrete** which can be filtered using the **Variability** drop down list.
	Filter on **Variability**. This function only apply to **Variables** and is disabled for **Parameters**. A variable can be either discrete or continuous.

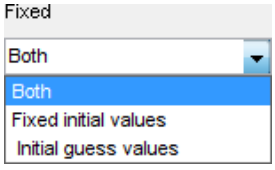
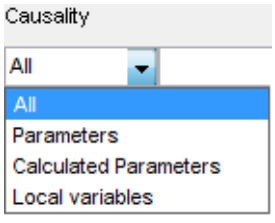
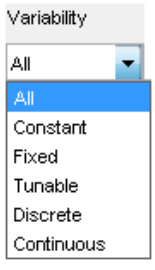
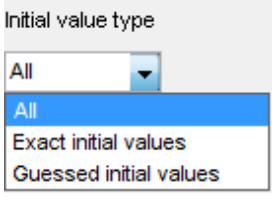
	Filter on fixed **Variables**. If **Fixed initial values** is selected, only variables with fixed initial values are shown. If Initial guess values is selected, only variables with initial guess values are shown.

Table 3.5 Filtering for FMI 2.0 variables

	Filter variables by name. Use the **Clear** button to reset the filter function.
	Filter on **Causality**. There are three defined categories, Parameters, Calculated parameters and Local variables. Use the drop down list to select the category to filter. Choose between **Parameters**, **Calculated parameters**, **Local variables** or **All**.
	Filter on **Variability**. There are five defined categories, Constant, Fixed, Tunable, Discrete and Continuous. Use the drop down list to select the category to filter. Choose between **Constant**, **Fixed**, **Tunable**, **Discrete**, **Continuous** or **All**.
	Filter on **Initial value type**. If **Exact initial values** is selected, only variables with Exact initial values are shown. If Guessed initial values is selected, only variables with initial value types Approx are shown.

3.4. FMU block and Simulink Coder

It is possible to use Simulink Coder with a Simulink model containing an FMU block loaded with a source code FMU. This makes it possible to access the capability of many Simulink Coder targets for configurations of Simulink blocks and FMUs, for example building a Simulink model with an FMU to a real-time platform.

The FMU block is tested with the targets listed in Table 2.5 and FMUs from Dymola versions listed in Table 2.6. However, other configurations are expected to work if:

1. The FMI version of the source code FMU is 2.0
2. The targets compiler can compile C89/C90 (ANSI) and `fmit_portable_stdint.h`
3. The source code FMU can be compiled with the targets compiler

FMI 1.0 is only supported for Dymola FMUs.

Check with tool vendors if their exported source code FMUs can be compiled with the compiler of the desired Simulink Coder target. If there is any problem with the compilation of `fmit_portable_stdint.h` it may be possible to extend it to work for your target/compiler. You can open the file in MATLAB with the command

```
>> edit(fullfile(fmitoolboxdir(), 'fmitoolbox', 'fmuBlock', 'fmit_portable_stdint.h'))
```

where more information is given.

Note that Dymola FMUs from 2015 FD01 and forward needs to be exported from Dymola using `Advanced.AllowMultipleInstances=false` if you your target platform is the dSPACE rti1006.tlc target. This may be true for similar targets and the compilation error will then look like: `error: thread-local storage not supported for this target`.

Note that Dymola source FMUs from 2015 FD01 and forward always are compiled with `NO_FILE` defined. This is because there is no guarantee that the target platform have a file system.

Note that building a model containing many FMUs may cause name conflicts when compiling the C code. This might happen if it contains two 1.0 FMUs or two 2.0 FMUs with the same model identifier.

3.5. Examples

Two examples covering the basic functionalities of the FMU block are described in this section. The first example demonstrates how to set start values of the FMU model in the GUI. The second example demonstrates how to configure the output ports.

3.5.1. Changing start values and using the filter functions

This example shows how to change the start values of different variables in a robot model from the Modelica Standard Library. To do this, the different filter functions will be used. The FMU file used in this example is found in the installation directory of FMI Toolbox under */examples/me1/<platform>/Robot/* and it is generated by Dymola.

The robot model is taken from the Modelica MultiBody Library and is a demonstration model of the Manutec 3d robot. The robot model contains variables corresponding to the starting angle of some robot axis. These variables are named `startAngle`. To find these variables, type `startAngle` in the field next to the **Filter** button and click **Filter**. In Figure 3.24 we see the result after filtering.

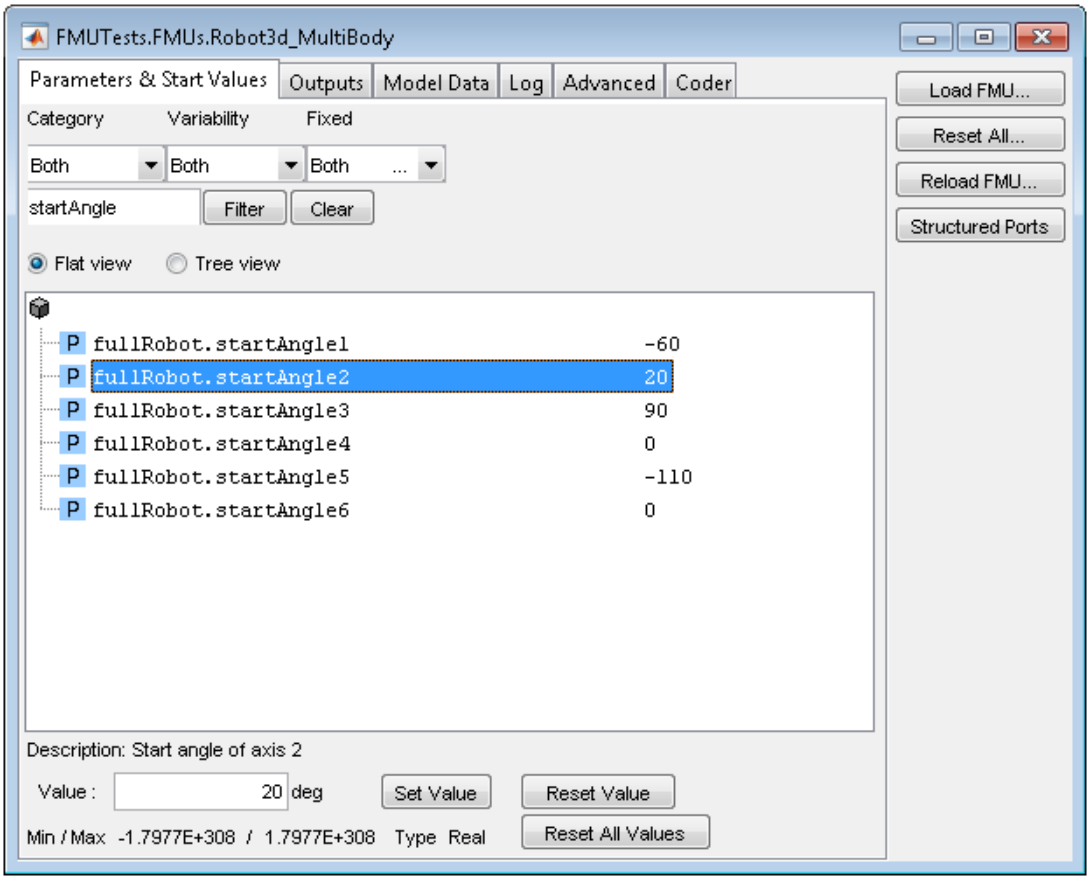


Figure 3.24 Result after filtering for "startAngle" in a Robot model.

To set a new start value, select the variable and type the new value in the **Value:** field. Then click the **Set value** button.

To set the start value of a variable that is **discrete** and has **initial guess values**, filter for this so no other variables and parameters are shown. To do that, select **Variables** in the **Category** drop down list. All **Parameters** will now disappear from the list. Select **Discrete** in the **Variability** drop down list and then **Initial guess values** in the **Fixed** drop down list.

The filtered results are shown in Figure 3.25 below. The icons of the nodes that are visual indicates that they are variables due to the V, the dark color indicates they are initial guess values and the pinkish color that they are Boolean values.

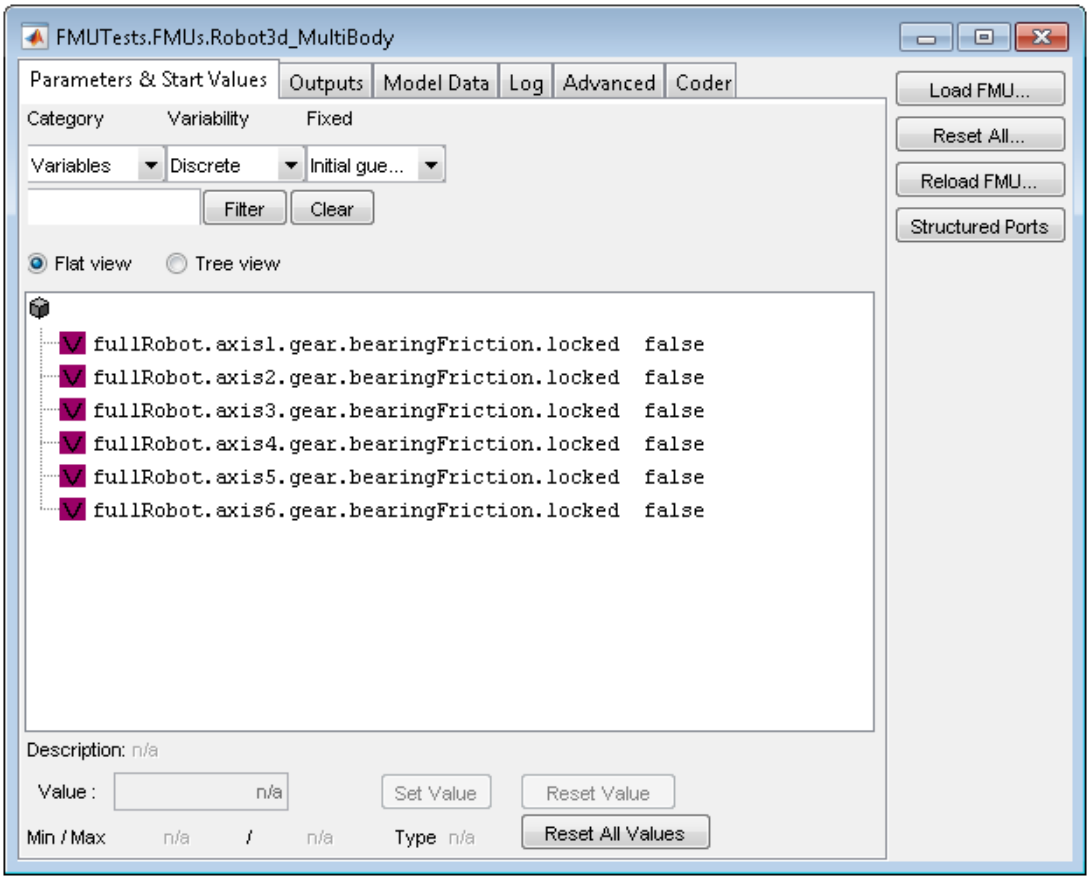


Figure 3.25 Filtering result for: Variables + Discrete + Initial guess values .

3.5.2. Configure outputs

This example shows how to change the output ports of a vehicle model. We will start by removing and renaming output ports and then add a new port, a vector port.

The output configuration we start with is seen in Figure 3.26 below. These are the top level output variables that are set when the model is loaded.

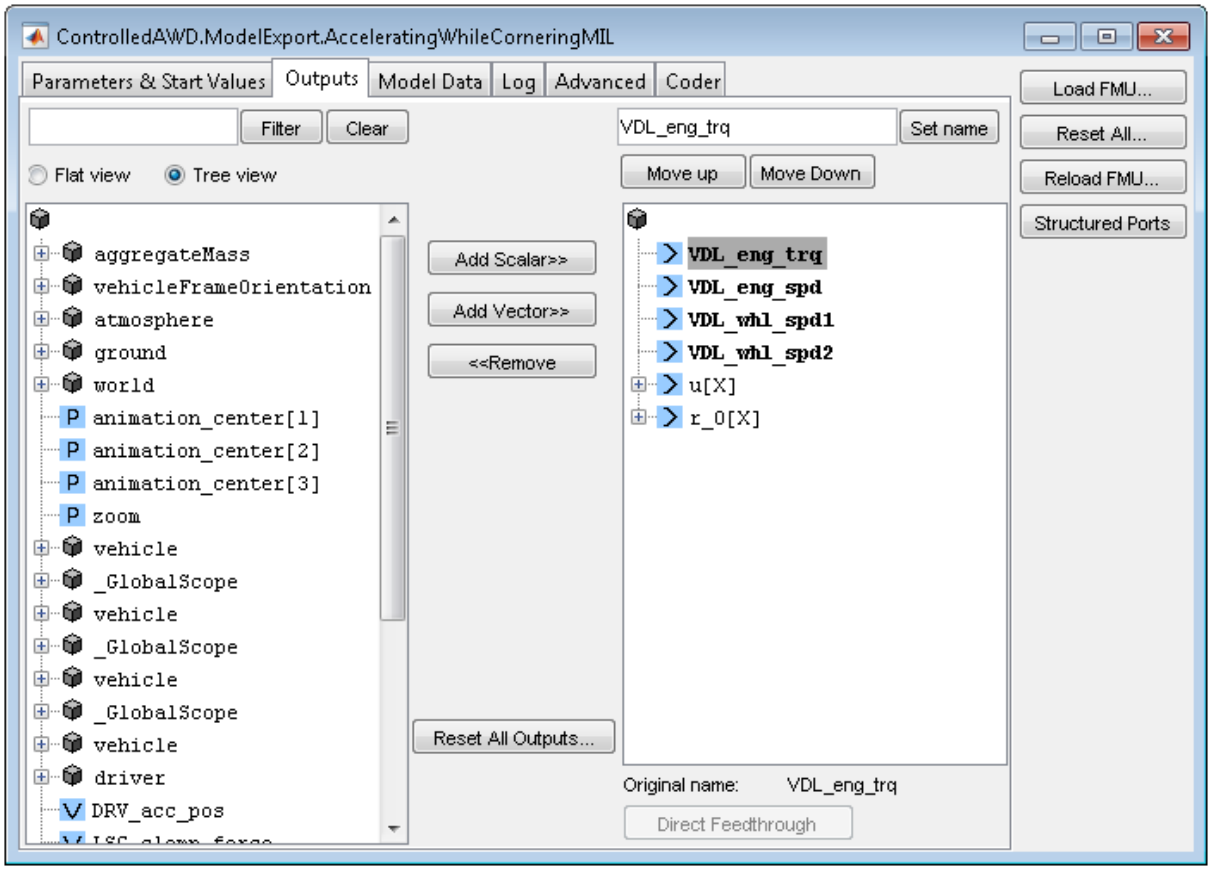


Figure 3.26 The output configuration of the vehicle model we start with.

In order to rename the output port `u[X]`, select the port in the right list. The field next to the Set name button will then be populated with the node´s name, `u[X]`. Type a new name, `u[1],u[3]`, in the field and click the **Set name** button. The new name is immediately set and it can be seen in Figure 3.27 below.

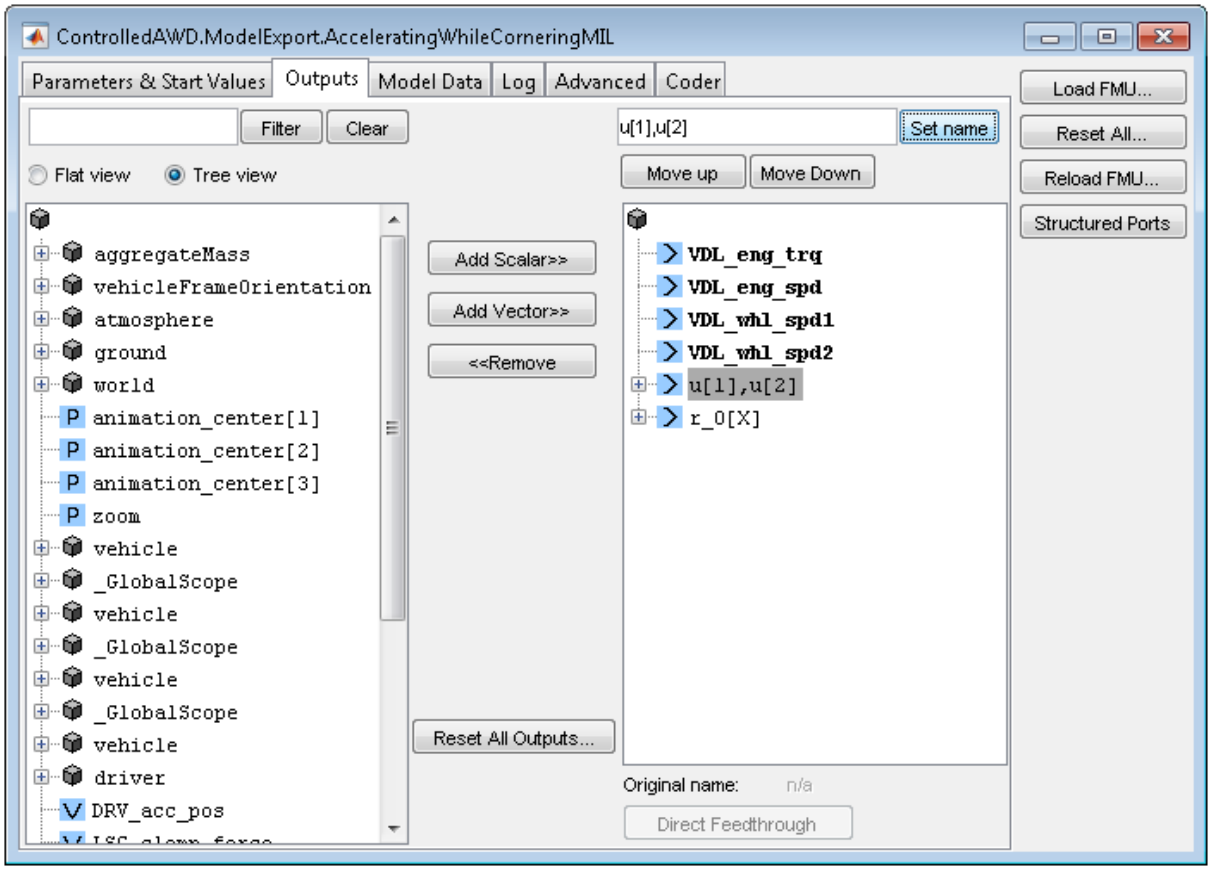


Figure 3.27 The name of the top most port was changed from u[X] to u[1],u[3].

In the next step the child variable `u[2]` will be removed from the vector output port `u[1],u[3]` that just had its name changed. The vector port `r_0[X]` will also get removed. This is done in two steps. First, select the child node `u[2]` in the vector port `v[1]`, `u[3]` and click the << **Remove** button. Secondly, select the other node, `r_0[x]` and click the << **Remove** button. The result is seen in Figure 3.28 below.

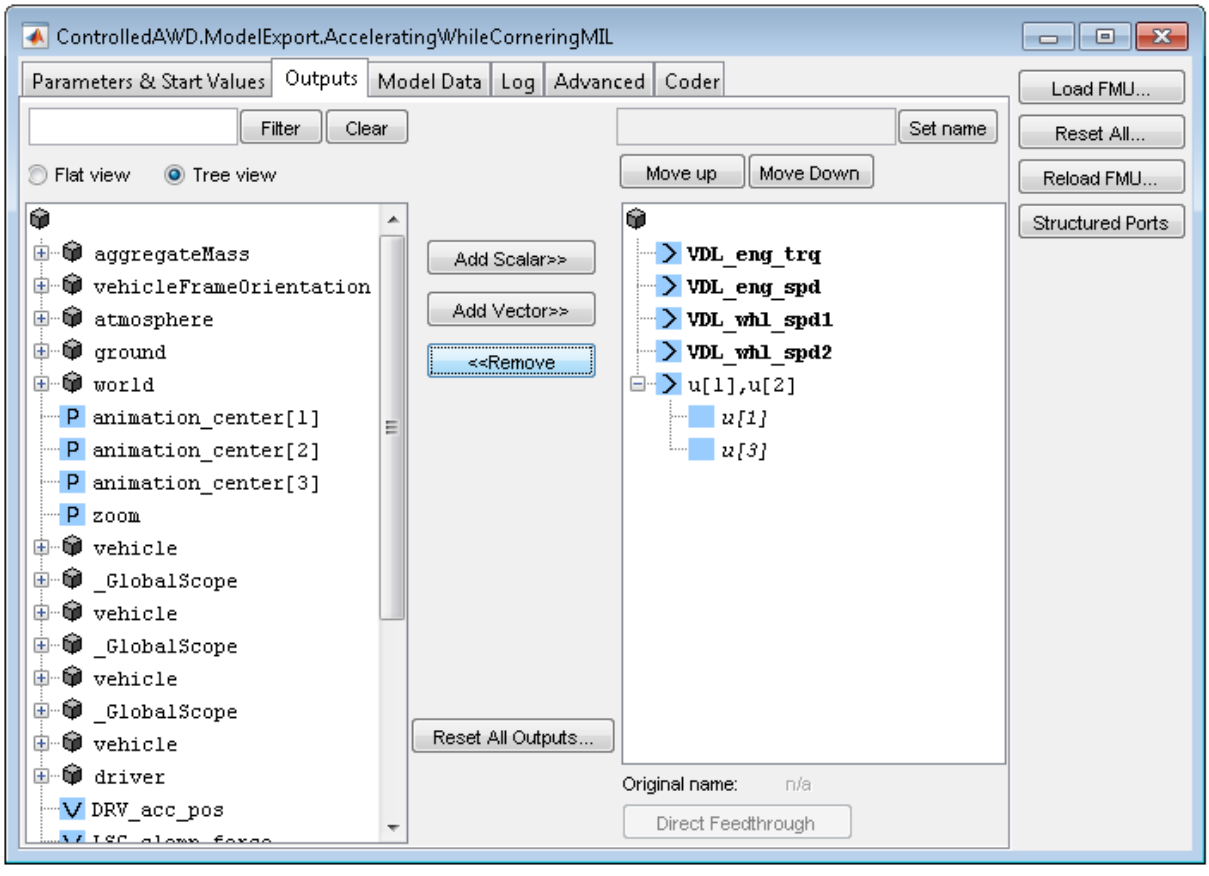


Figure 3.28 Output port configuration after the variable u[2] and the whole vector port r_0[X] have been removed.

Now add a vector port consisting of the vector variables `vehicleAcceleration.y`. Filtering for this name results in the tree variables `y[1]`, `y[2]` and `y[3]` visualized in the left list, seen in Figure 3.29. Select these variables by Ctrl+mouse click all of them. In the list to the right, select the port it should be added to. The port `u[1],u[3]` is selected in the right list here.

Now click the **Add vector >>...** button. A input dialog box appear where port name `y[X]` is set. The final result of how the block output port configuration looks like in the FMU setup window and on the FMU block is seen in the two figures below, Figure 3.29 and Figure 3.30.

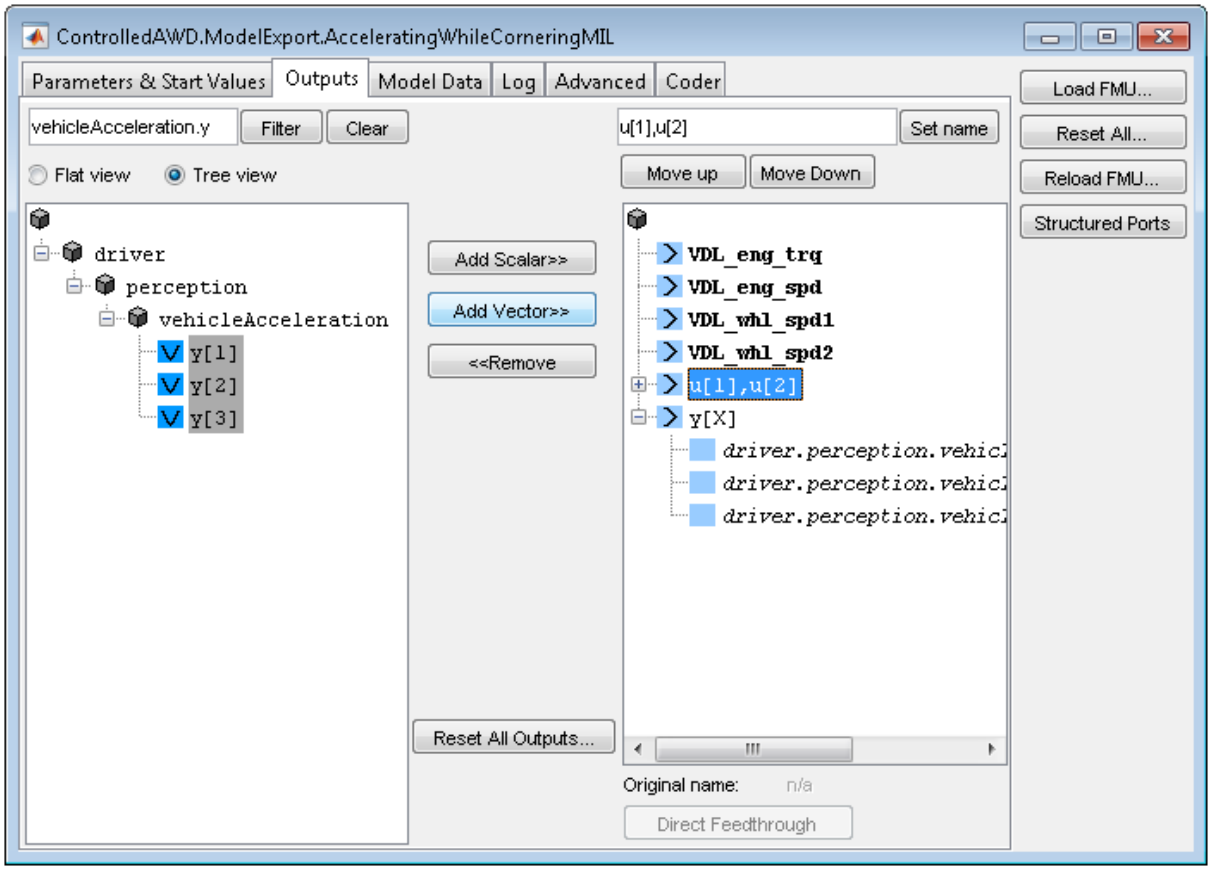


Figure 3.29 Final output port configuration of the vehicle FMU model.

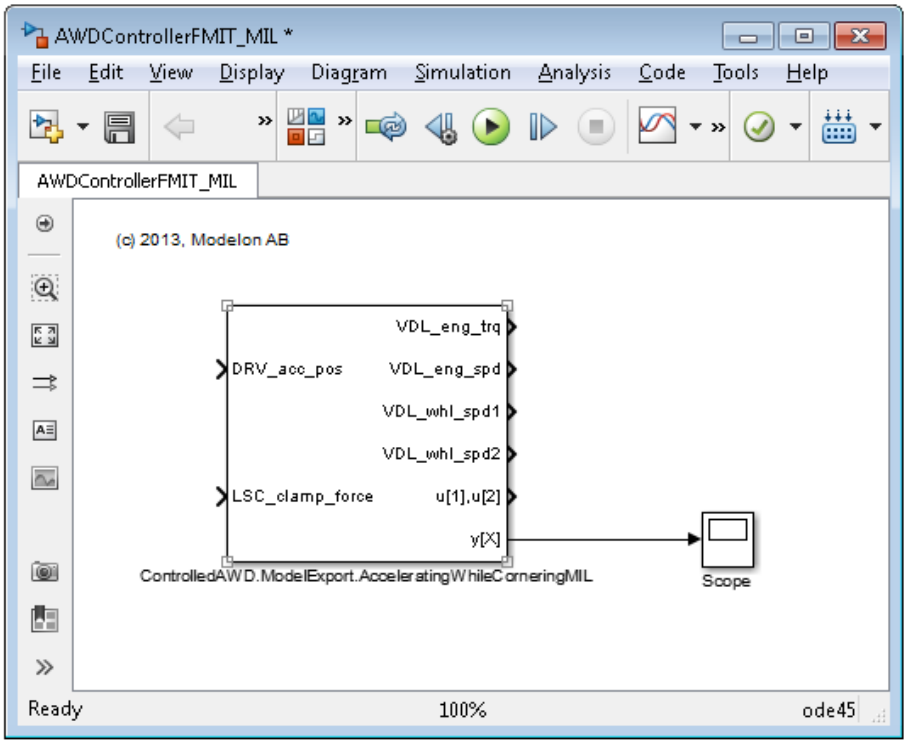


Figure 3.30 FMU block with the final output port configuration.

3.5.3. Configure ports using structural naming

This example show how the **Structured Ports** feature can be used. We will start by adding some new outputs to the CoupledClutches example and then take advantage of the new ports structural naming.

Open the Coupledclutches model and go to the **Output** tab. Mark the variables coupledClutches.J1.J and coupledClutches.J1.phi and add them as outputs by clicking the **Add Scalar >>** button, see figure Figure 3.31. The FMU blocks port configuration should now be the same as seen in figure Figure 3.32.

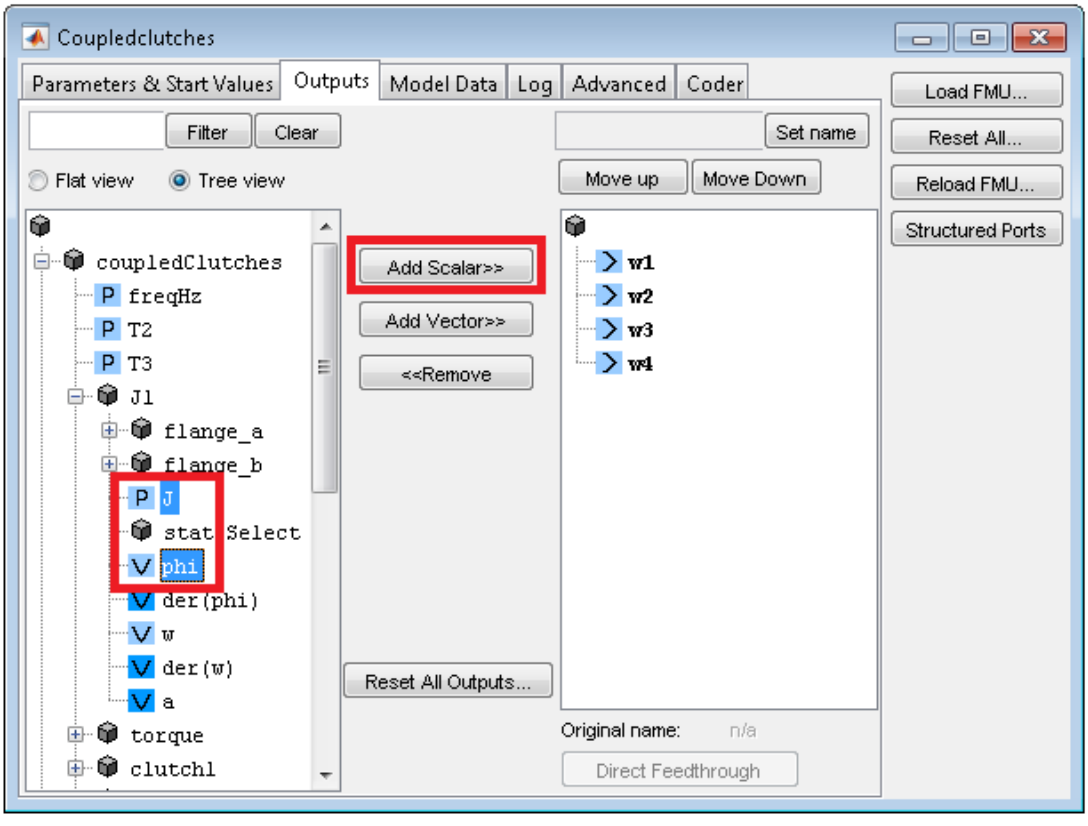


Figure 3.31 coupledClutches.J1.J and coupledClutches.J1.phi marked to be added as outputs.

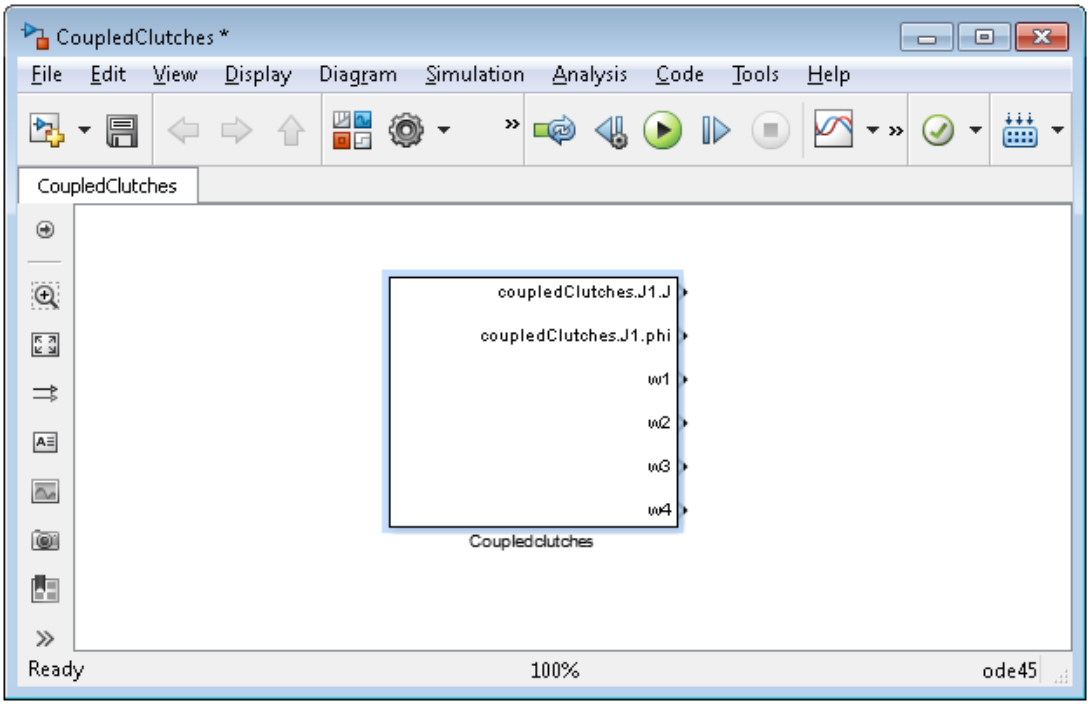


Figure 3.32 FMU block with additional outputs.

Now click the **Structured Ports** button, see figure Figure 3.33. This should have changed the FMU blocks port configuration. To see what happened add a Bus Selector block and connect it to the new output coupledClutches as displayed in figure Figure 3.34.

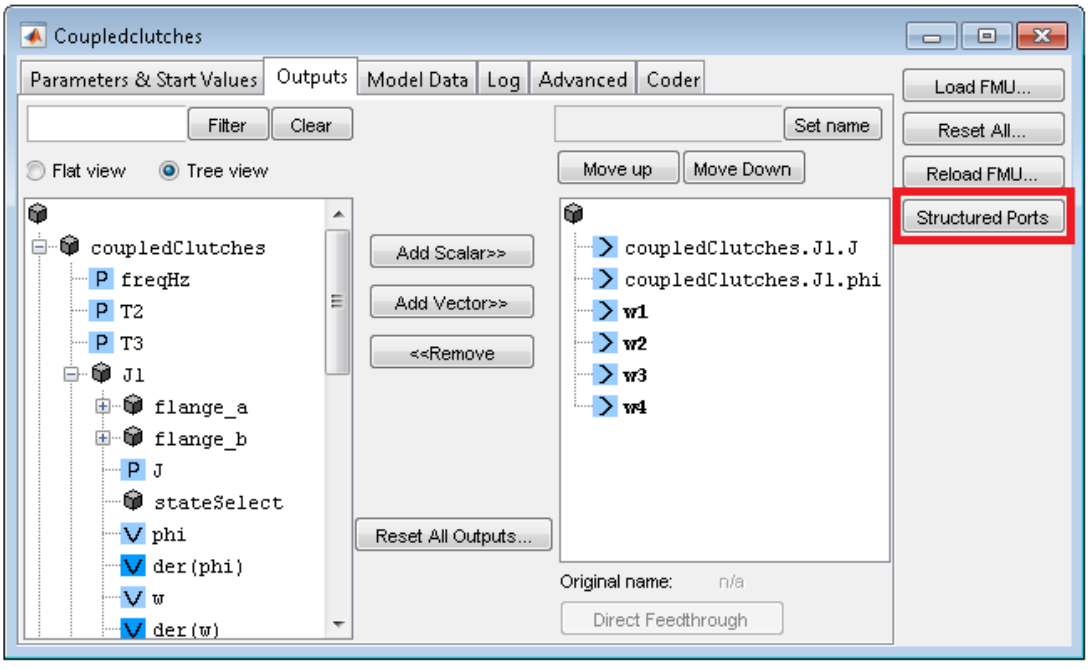


Figure 3.33 The Structured Ports button highlighted.

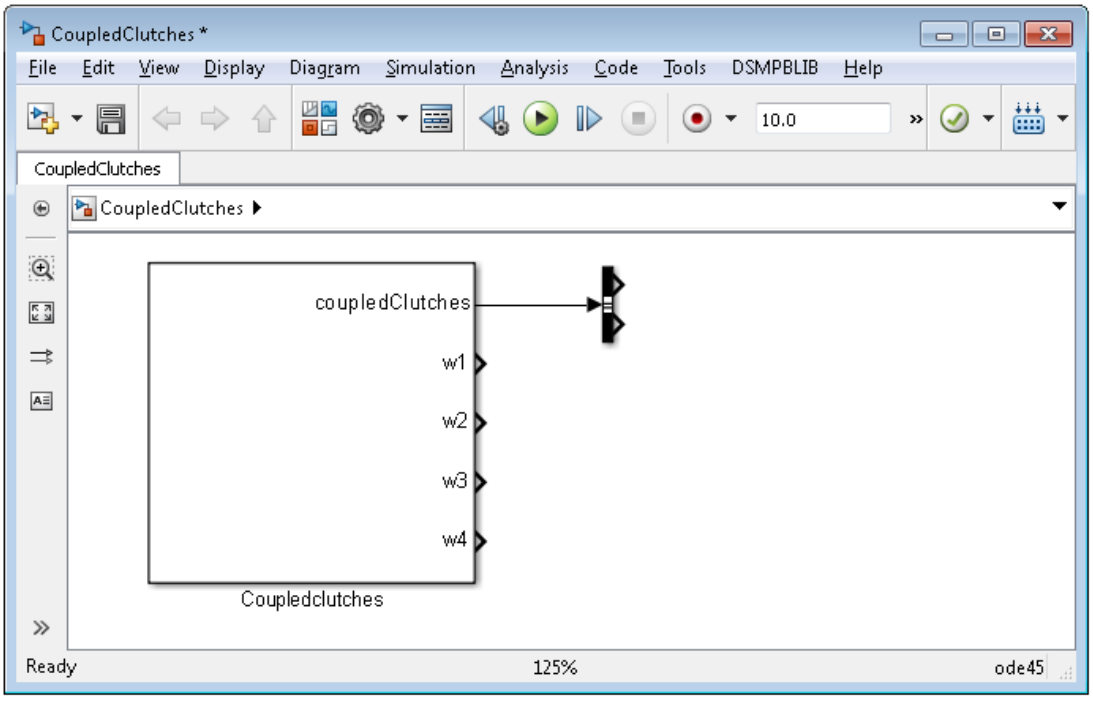


Figure 3.34 FMU block with Structured Ports and a Bus Selector block.

Now double click the Bus Selector block and examine the 'Signals in the bus', it should look similar to figure Figure 3.35. The structured names has been used to create a bus structure as output! Note that for coupledClutches.J1.phi you can see that coupledClutches is the name of the port and J1 and phi are signal names in the bus structure.

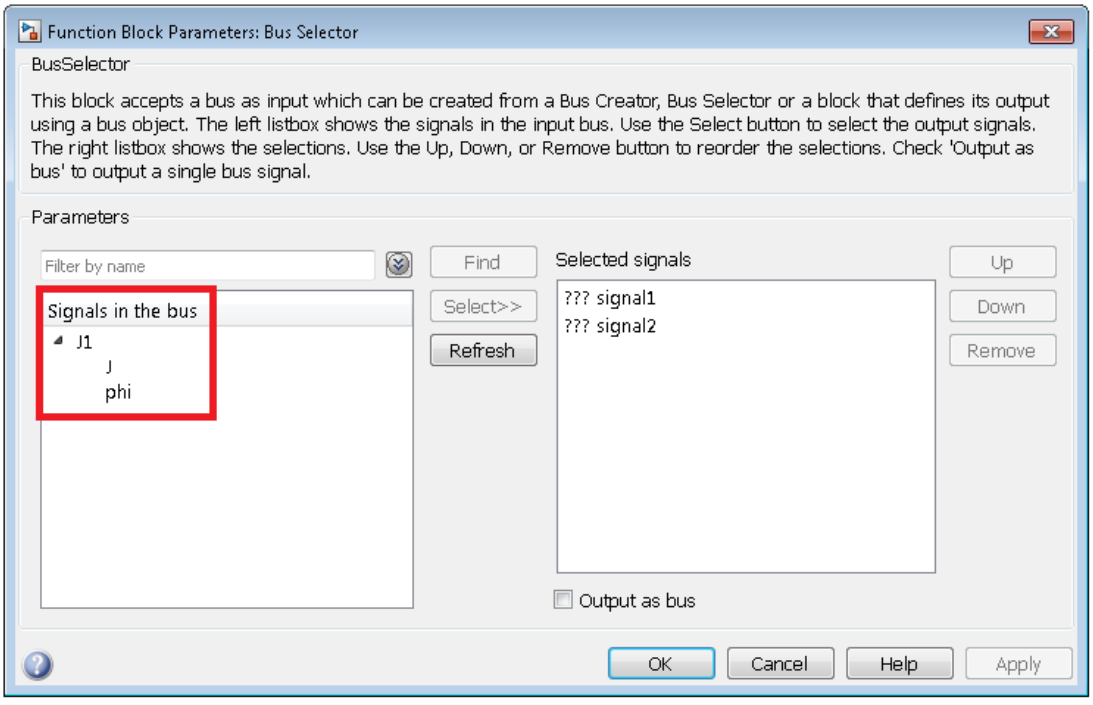


Figure 3.35 The signals in the Bus as shown in the Bus Selector GUI.

This feature can be used when inporting FMUs that have a lot of inputs/outputs with an advanced structure. Note that changes to the outputs, realoding FMUs and loading FMUs can only be done in the mode with regular ports.

3.5.4. Build target containing an FMU block

In this example, a Simulink model containing an FMU block, will be built for the FMU Co-Simulation 1.0 target. Note that this example is not distributed in the FMI Toolbox example folder and that the FMU is a source code FMU. For building targets containing an FMU that is not a source code FMU see Section 3.3.7.

1. The source code FMU that is used in this example is generated with Dymola 2014. The model can be seen in the block diagram in Figure 3.36 . The model has one input signal and output signal where the output signal is equal to the integrated input signal.

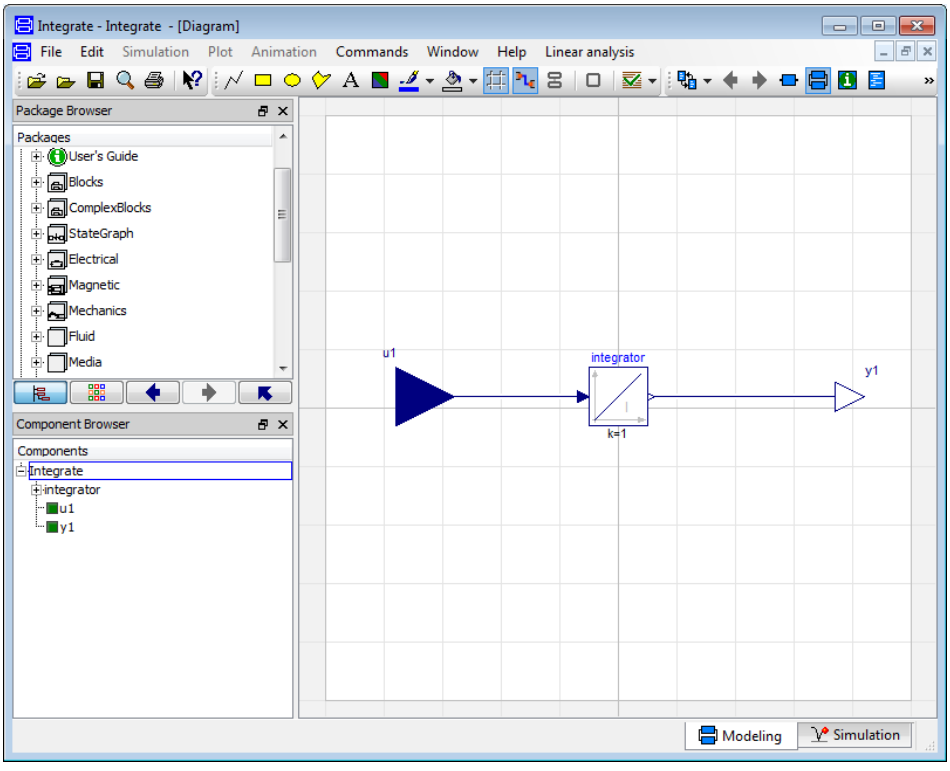


Figure 3.36 Block diagram view of the FMU model in Dymola.

2. Create a new Simulink model and add the FMU block for Co-Simulation to the model.

3. Load the FMU block with the Co-Simulation FMU. Make sure that the FMU contains the source code. See Section 3.2 for more details.

4. Add an input port and an output port and connect the blocks.

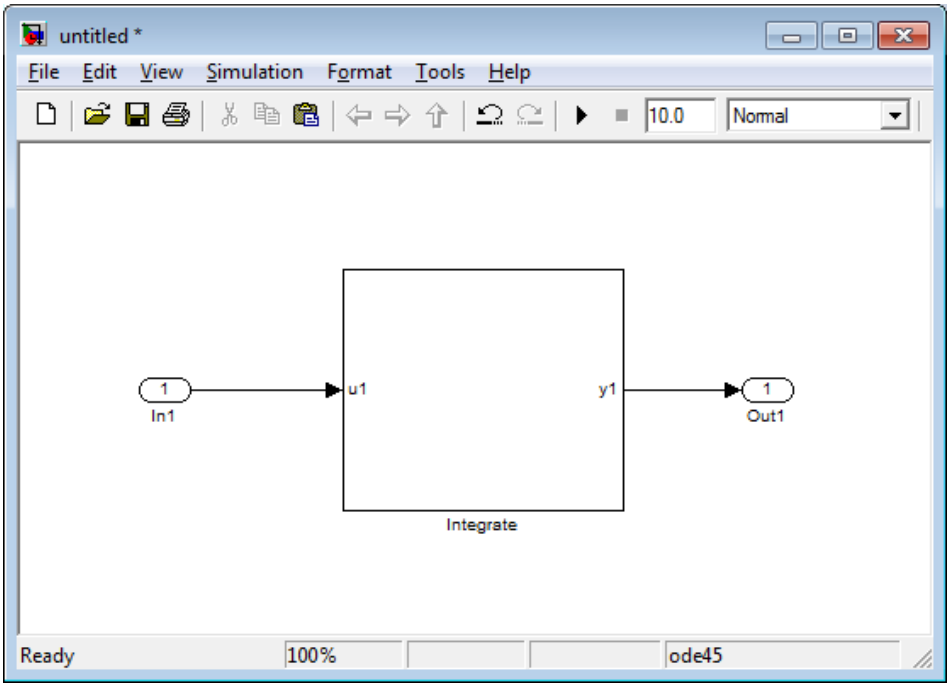


Figure 3.37 Simulink model with the FMU loaded.

5. Configure the model for FMU Co-Simulation export target, see Chapter 5 for details.
6. Build the Simulink model target. A Co-Simulation FMU will now be created in the current directory.
7. Create a new Simulink model and import the newly created FMU.
8. Connect a sine signal and a scope to the input and output port on the FMU block.

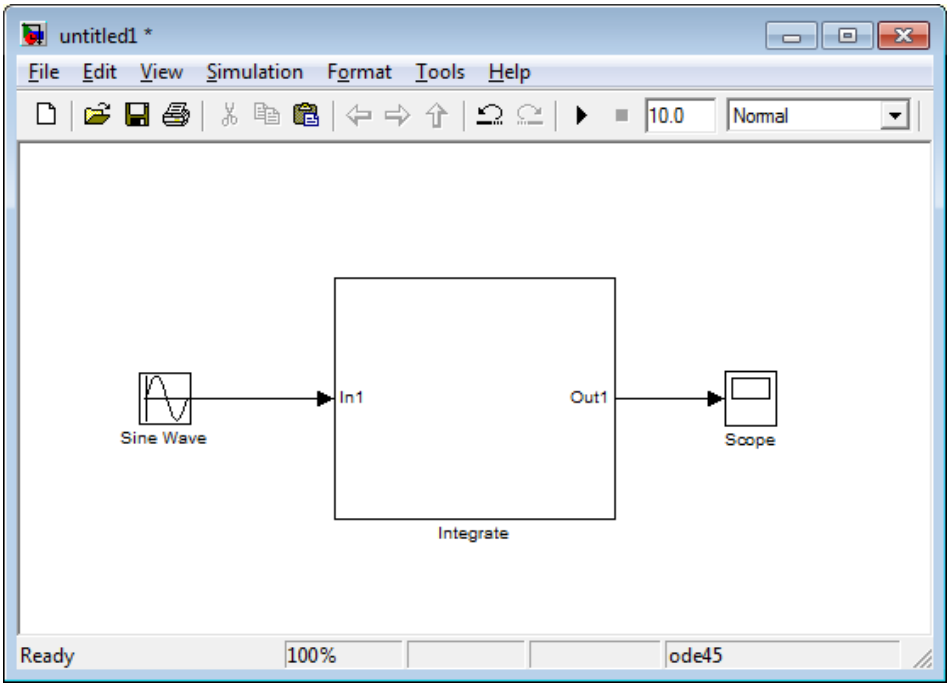


Figure 3.38 The FMU generated from Simulink is loaded into a new model.

9. Simulate the model. The results from the scope is seen in Figure 3.39.

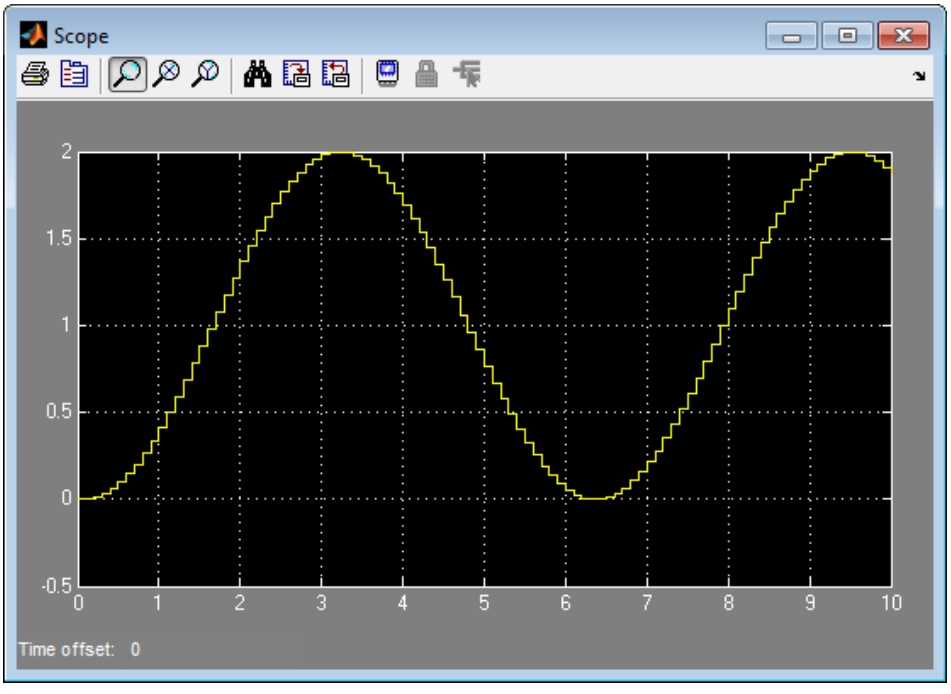


Figure 3.39 Simulation results of the integrated sine wave from the imported Co-Simulation FMU.

3.5.5. Build rti1006.tlc target containing an FMU block

In this example, a Simulink model containing an FMU block for Model Exchange, will be built and run on the dSPACE´s DS1006 platform. Note that this example is not distributed in the FMI Toolbox example folder and that the FMU is a source code FMU. The example assumes that the appropriate dSPACE software is installed and that the RTI Platform Support RTI1006 is activated.

1. The source code FMU that is used in this example is generated with Dymola 2014. The model can be seen in the block diagram in Figure 3.40 . The model has one input signal and output signal where the output signal is equal to the integrated input signal. Make sure that Dymola exports the model as an FMU for Model Exchange 1.0 and includes the source code for the model.

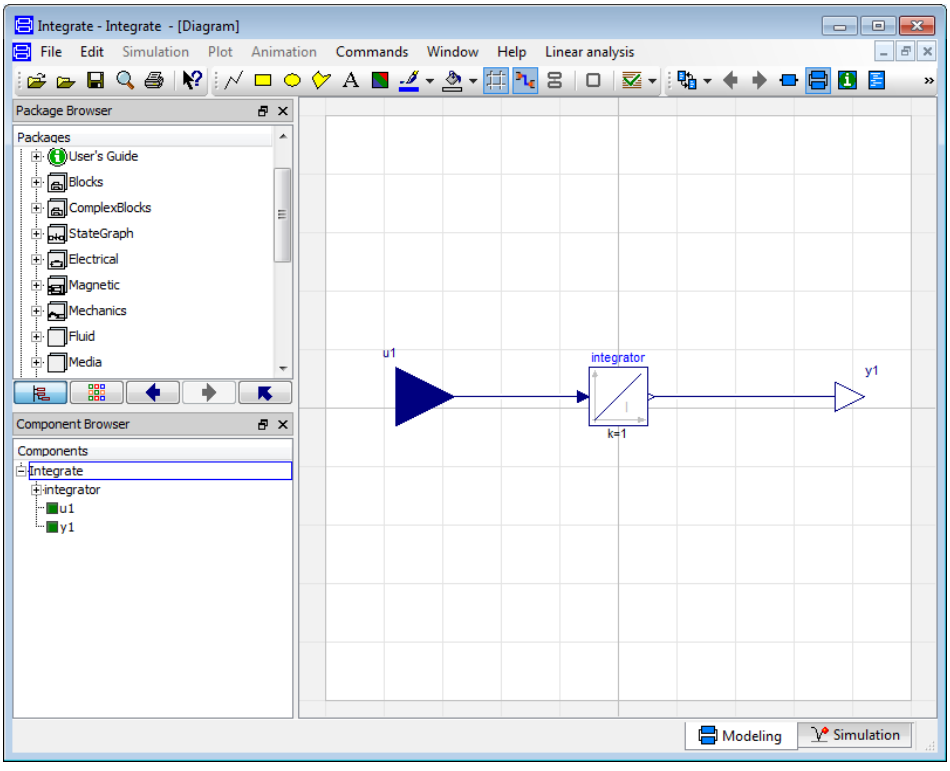


Figure 3.40 Block diagram view of the FMU model in Dymola.

2. Create a new Simulink model and add the FMU block for Model Exchange to the model.

3. Load the FMU block with the Model Exchange FMU. Make sure that the FMU contains the source code. See Section 3.2 for more details.

4. Add an input port and an output port and connect the blocks.

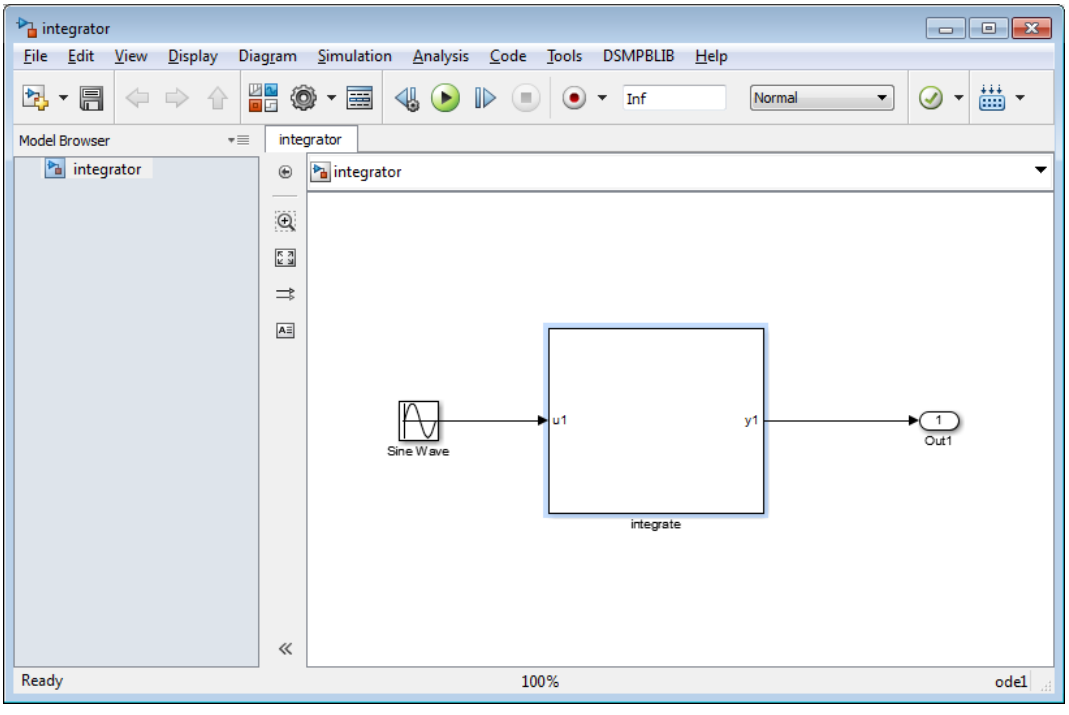


Figure 3.41 Simulink model with the FMU loaded.

5. Configure the model for the rti1006.tlc target if it is not already done. In this example, the new Simulink model is automatically configured for the rti1006.tlc target when it was created.

6. Build the Simulink model. The Simulink model compiles and downloads the executable to the dSPACE DS1006 system.

7. The output signals from the FMU block can now be viewed from dSPACE´s ControlDesk® Next Generation, see Figure 3.42.

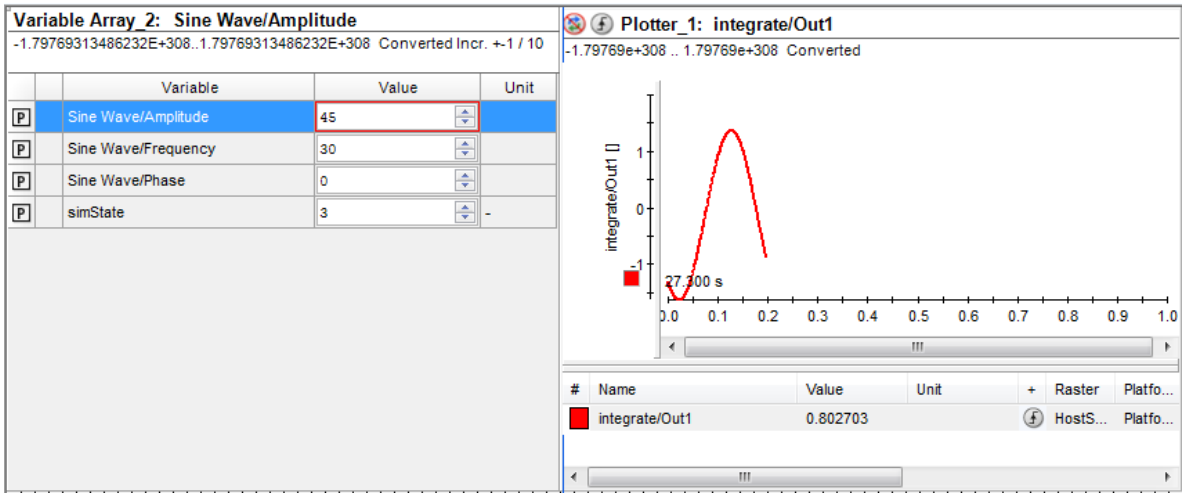


Figure 3.42 A **Variable Array** and a **Plotter** instrument displaying the parameter values and output signal in dSPACE`s ControlDesk Next Generation.

3.5.5.1. Set start values and parameters

In order to set a start value, one can use the script function or GUI, seeSection 3.3.1, and then rebuild the model according to the above procedure Section 3.5.5. An additional approach to set a new start value is to change the parameter from inside ControlDesk Next Generation and then restart the simulation. That can be achieved in following steps:

1. Add the parameters to change start values for, to a **Variable Array** instrument in the ControlDesk Next Generation layout.

2. Add the **simState** parameter to the **Variable Array**. It can be located in the main group of the variable description file. **simState** is a parameter to read or set the simulation state of the application. This variable can take on the states STOP (0), PAUSE (1), or RUN (2).

3. Change the **simState** value to 0, to stop the simulation.

4. Change the start values of the parameters.

5. Change the **simState** value to 2, to start the simulation.

6. The FMU uses the new start values to instantiate and initialize the model.

Chapter 4. Simulation in MATLAB

4.1. Introduction

An FMU model can be loaded in to the MATLAB environment and be simulated using MATLAB´s ODE solvers. Start values and parameters can be set, and if the model has inputs, these can be set with user defined input data. The output from the simulation can be configured by the user, such as variables and result files. FMI versions 1.0 and 2.0 for Model Exchange and Co-Simulation are supported.

4.2. A first example

An FMU model loaded in the MATLAB workspace is a class object. An FMU model class is created with the `loadFMU` function, or one of the specific model class constructors `FMUModelME1`, `FMUModelME2`, `FMUModelCS1` or `FMUModelCS2`. `loadFMU` automatically chooses the right constructor and loads and instantiates the model. When using the constructors directly, the FMU has to be manually instantiated.

Table 4.1 Different FMU model object creators

FMU class constructor and factory functions	FMU type
`loadFMU`	Will check the FMU and use the appropriate constructor. Also calls the instatiate method of the returned class object.
`FMUModelME1`	Model Exchange 1.0
`FMUModelME2`	Model Exchange 2.0
`FMUModelCS1`	Co-Simulation 1.0
`FMUModelCS2`	Co-Simulation 2.0

The version independent way to load an FMU into MATLAB is:

```
>> fmu = loadFMU('bouncingBall.fmu');
```

The variable `fmu` is now representing the FMU model. The model description file is parsed and all the FMI functions are ready to be called.

Once loaded, the model can be simulated using the `simulate` function. Due to the differences between Model Exchange and Co-Simulation models, the inputs to the function have slightly different meanings. The minimal set of arguments required is the time interval to be simulated, which for ME models can be specified either as an interval `[tStart, tEnd]` or as a vector of time points, `[t_0, t_1, ..., t_end]`. In the first case the returned result is sampled at each solver step, while in the second case the output will be sampled at the specified time points. Co-Simulation models, on the other hand, require a set of communication points, defining the simulation intervals for the model's internal solver. This means that the second notation is always used, and if only `[tStart, tEnd]` is supplied, the start and end points will be the only samples returned.

The most general call to `simulates` is thus

```
>> [tout, yout, ynames] = fmu.simulate([tStart : stepSize : tEnd])
```

For Co-Simulation models, `simulate` will call the internal solver of the model, while Model Exchange models will be simulated using one of MATLAB's solvers (the default is ode15s).

The simulation can also be controlled by options provided to `simulate`, e.g.:

```
>> fmu = loadFMU('bouncingBall.fmu'); %as before
>> outdata.name = {'h','v'};
>> opts = odeset('RelTol', 1e-6);
>> [tout, yout, yname] = fmu.simulate([0,2.5], 'Output', outdata, 'Options', opts);
>> plot(tout,yout);
>> legend(yname);
```

Here, additional arguments for the outputs are specified in the `outdata` struct. Specifically, the variable names in `outdata.name` decide which model variables are given as output in `yout`. For a full list of the options available, see the function documentation in MATLAB.

In the above example a bouncing ball model is simulated. The model simulates a ball that is dropped from a height and bounces onto the ground.

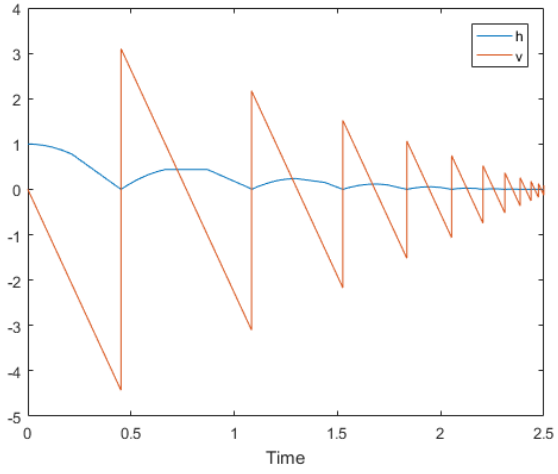


Figure 4.1 Simulation result for the bouncingBall.fmu model. The blue curve corresponds to the height of the ball and the red curve corresponds to the velocity of the ball.

The full call sequence to load and simulate a model includes loading, instantiating, initializing and simulating. For ease of use, instantiation is handled by `loadFMU` and initialization by `simulate` (unless the model is previously

initialized). The extra calls can be used to achieve more fine-grained control over the model setup. For example, assuming that the model is a 1.0 Model Exchange FMU, the previous example may be rewritten as:

```
>> fmu = FMUModelME1('bouncingBall.fmu');
>> fmu.fmiInstantiateModel(); % different calls for 1.0/2.0 and ME/CS
>> fmu.initialize();          % default initialization
>> fmu.simulate([0 3]);
```

4.3. Using the FMU model classes

This section describes in more detail how the FMU model classes work. The examples used for demonstration apply to both the Model Exchange classes and the Co-Simulation classes.

4.3.1. Handle class

The FMU classes are derived from the handle class which means that variables are references (handles) to an object instance rather than complete copies of the object. This makes FMU objects behave simliarly to figures in MATLAB. A consequence of this is that an object cannot be copied in the usual MATLAB manner, only the reference to it. This is demonstrated in the example below, where `fmu` and `fmu_copy` refer to the same object, causing changes to one to affect the other as well.

```
>> fmu = loadFMU('bouncingBall.fmu');
>> fmu.get('h')

ans =

     1

>> fmu_copy = fmu;
>> fmu_copy.set('h', 2)
>> fmu.get('h')

ans =

     2
```

The FMU model class handles the destruction of the model when all its references are cleared, e.g when `clear all` is called or when MATLAB terminates.

4.3.2. Calling functions

The FMU model class' methods can be called in two different ways. There is the one used in the examples above, with the class instance before the function name separated with a dot: `<obj>.<function(...)>`. The other

way to call a class function is to use the function name first, and the class instance as the first input argument: `<function(<obj>, ...)>`.

```
>> fmu = loadFMU('bouncingBall.fmu');
>> fmu.get('h')  % Alternative calling convention 1.
ans =

     1

>> get(fmu, 'h') % Alternative calling convention 2.
ans =

     1
```

The documentation of the functions uses calling convention 2, i.e the first input argument is the class instance, in this case `FMU`:

```
>> help FMUModelME1.get     % or another FMU class
 get returns the values of variables
  VALUES = get(FMU, NAMES) gets VALUES for the variable NAMES. NAMES is a
  ...
```

One advantage of using the function call with dot notation is that auto completion can be used (*if it is enabled in the MATLAB preferences*). By starting to type a class name or function and then press the <**Tab**> button, a list of alternatives is displayed in a tool tip. The up and down arrows on the keyboard can be used to select the right function, and then press enter. *Note, the tab completion may list functions that are not valid to use. Use only the documented functions that are found in fmi_information for that class, see below.*

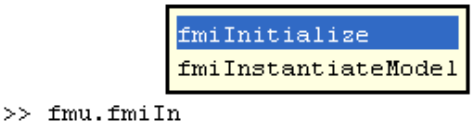


Figure 4.2 Use the Tab button for auto completion.

4.3.3. Help

To get more information about the classes, the MATLAB `help` command can be used.

For each class, `ClassName.fmi_information` lists all available functions, e.g for the FMUModelME1 class, the following functions are listed:

```
>> help FMUModelME1.fmi_information
FMI Model Exchange 1.0 information
  FMI for Model Exchange 1.0 specification
    https://svn.modelica.org/fmi/branches/public/specifications/v1.0/FMI_for_ModelExchange_v1.0.pdf
```

```
  Help functions
    FMUModelME1
    eventUpdate
    getValue
    ...

  All FMI functions are listed here:
    fmiCompletedIntegratorStep
    fmiEventUpdate
    ...


    See also fmi_status, event_info, loadFMU
```

To get help from a specific class function, the class name must be included before the function name and separated with a `.`:

```
>> help FMUModelME1.fmiGetVersion
 fmiGetVersion returns the FMI version for model exchange
  VERSION = fmiGetVersion(FMU) returns the VERSION of the
    FMU´s implemented model exchange interface. FMU is an
    fmu model, see FMUModelME1. This MATLAB class supports
    only FMI version 1.0 for model exchange.


    See also fmi_information
```

4.4. Examples

The following three examples cover how some of the class functions can be used. Special attention is given to the simulate function, describing how different input and output configurations are set up. The simulate function works in the same manner for both Model Exchange and Co-Simulation if nothing else is mentioned.

The `simulate` function takes the time span to simulate and the class instance as mandatory input arguments. All other arguments are optional and are set as name-value pairs:

```
[TOUT, YOUT, YNAME] = simulate(FMU, TSPAN, 'NAME1',VALUE1,...)
```

The properties that can be set and what values they have are described in more detail in the help documentation in MATLAB.

4.4.1. Set start values and parameters

Setting start values to an FMU is done using FMU model class methods. `set` and `get` are two methods that invoke the low level FMI function for setting and getting values. This examples demonstrates how to run multiple

simulations with different parameter values. The FMU is a model of the Van der Pol oscillator, generated from a Modelica compiler using the Modelica code below:

```
model VDP
    // State start values
    parameter Real x1_0 = 0;
    parameter Real x2_0 = 1;
    // The states
    Real x1(start = x1_0);
    Real x2(start = x2_0);
    // The control signal
    input Real u;
equation
    der(x1) = (1 - x2^2) * x1 - x2 + u;
    der(x2) = x1;
end VDP;
```

A pre-compiled FMU for Model Exchange 1.0 is included in the FMI Toolbox distribution. Using this FMU, the following script does a parameter sweep for the initial values of the states `x1` and `x2`:

```

VDP_fmu_path = 'VDP.fmu'; %Path to the FMU file.

%Load the FMU
fmu = FMUModelME1(VDP_fmu_path);
%Define initial conditions
final_time = 20;
N_points = 11;
x1_0 = linspace(-3, 3, N_points);
x2_0 = zeros(size(x1_0));

output.name={'x1', 'x2'};
for k = 1:N_points
    %Set initial conditions in model
    fmu.fmiInstantiateModel;
    fmu.set('x1_0', x1_0(k));
    fmu.set('x2_0', x2_0(k));
    %Simulate
    [tout, yout, yname] = fmu.simulate([0, final_time], 'Output', output);
    %Plot simulation results
    plot(yout(:,1), yout(:,2));
    hold on
end
xlabel(yname(1));
ylabel(yname(2));
```

This script should generate a plot similar to the one below:

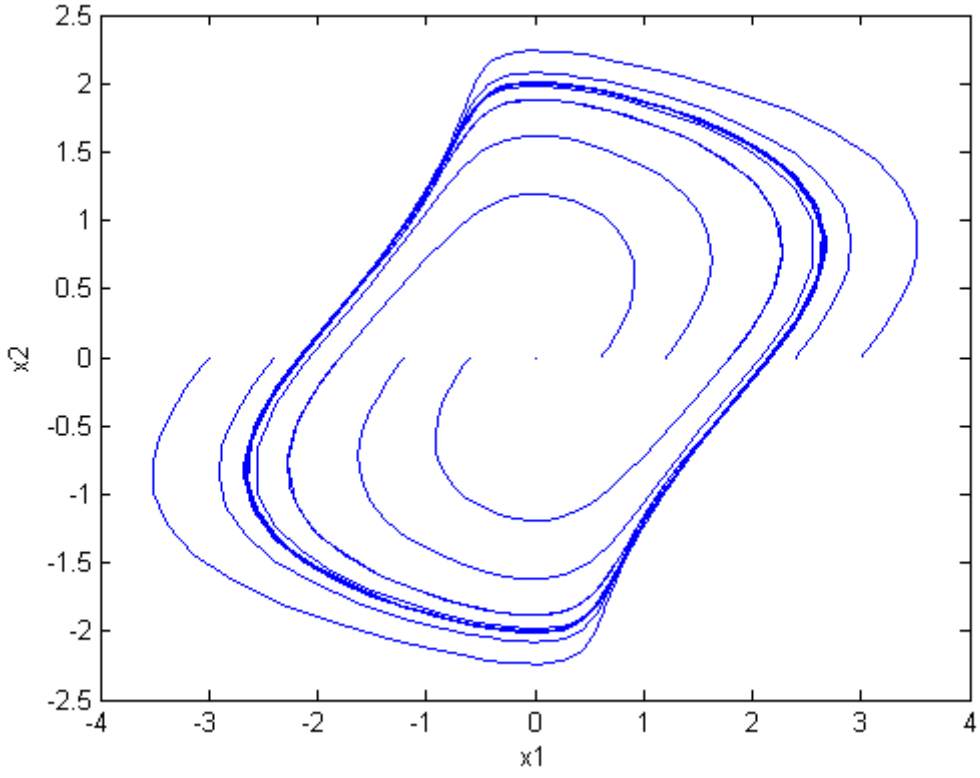


Figure 4.3 Phase plane for the Van der Pol oscillator.

4.4.2. Simulation with inputs

In this example, inputs are used in the simulation using both table data and a MATLAB function handle. The FMU model that is simulated is the first example model in Mechanics.Rotational from the Modelica Standard Library. It has one input variable, `u1`, and three outputs, `w1`, `w2`, `w3`. The model is simulated for 10 seconds and both methods for generating an input signal are described here.

Create a class instance for the FMU model:

```
>> fmu = loadFMU('Mechanics_Rotational.fmu');
```

The property name used to set an input signal is `Input`. The value set for the `Input` property is a cell array with structs where each struct is representing an input signal. Such a struct must have two fields, one with the input variable name and one with the signal data.

The variable name is set in the struct field `name`. The second field, containing the signal data, can have one of two field names: `vec` or `fcn`. `vec` indicates the use of table data, and its use varies slightly between Model Exchange and Co-Simulation models. For Model Exchange, the input is provided as an array with sample times in the first column and signal values in the second:

```
>> inconf{1}.name = 'tau';
>> t = [0:0.1:10]';
>> data = sin(t);
>> u1data = [t', data'];
>> inconf{1}.vec = u1data;
```

For Co-Simulation models, providing the sample times is not necessary as these are determined by the communication points. Thus, only the signal values are provided in this case:

```
>> inconf{1}.name = 'tau';
>> t = [1:0.1:10];
>> inconf{1}.vec = sin(t);
```

The other way of providing input is to use a MATLAB function handle. Then, the struct field `fcn` is used instead:

```
>> inconf{1}.name = 'tau';
>> inconf{1}.fcn = @(t)sin(t);
```

An input function should take a single input argument (the independent variable) and return a scalar value to be used as the current input sample. Since sampling is handled internally in the `simulate` function, there is no difference in input between Model Exchange and Co-Simulation models.

In this example we choose the function handle to generate the input signal for `u1` when simulating. The simulation results for the default output variables `w1`, `w2`, `w3`, corresponding to angular velocities of the inertias in the model, are visualised in the plot below.

```
>> fmu_me = loadFMU('me1/Mechanics_Rotational.fmu');
>> fmu_cs = loadFMU('cs1/Mechanics_Rotational.fmu');
>> inconf{1}.name = 'tau';
>> inconf{1}.fcn = @(t)sin(t);
>> [time_me, yout_me, yname_me] = fmu_me.simulate([0,10], 'Input', inconf);
>> [time_cs, yout_cs, yname_cs] = fmu_cs.simulate(0:0.1:10, 'Input', inconf);
>> subplot(1,2,1); plot(time_me,yout_me); title('Model Exchange'); legend(yname_me);
>> subplot(1,2,2); plot(time_cs,yout_cs); title('Co-simulation');  legend(yname_cs);
```

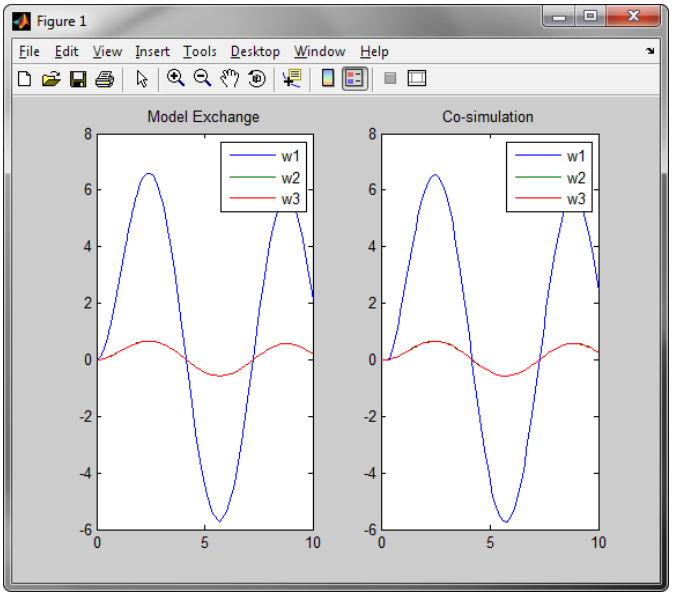


Figure 4.4 Simulation results for angular velocities of inertias in the Mechanics model FMU.

4.4.3. Simulation with configured output

This example shows how the output from the simulation function can be configured. We use the CoupledClutches model, found in the Modelica Standard Library, and simulate it for 10 seconds both as Model Exchange and Co-Simulation. The output is configured to exclude the default output variables, i.e., all top level output variables in the FMU, and instead contain the variables `clutch3.flange_a.phi` and `clutch3.tau`. The output is also configured to create a Dymola-styled result file. After the simulation, the commands for loading and visualising the result from the result file in MATLAB are described and compared with the output result that is created in the MATLAB workspace.

The property name for configuring the output is `Output`. The value is a struct with different fields. The different fields are described below. In this example, we call the struct `outconf`.

To turn off the default output results, the outputs of the FMU model, the struct field `toplevel` is set to false:

```
>> outconf.toplevel = false;
```

Add the names of the variables that should be included in the output by adding the names in a cell array. This cell array is set to the struct field `name`.

```
>> outconf.name = {'coupledClutches.J4.a','coupledClutches.J2.a'};
```

The last output configuration is to set the `writefile` field to true so that a Dymola-styled result file is created.

```
>> outconf.writefile = true;
```

Finally, simulate the model and plot the results:

```
>> fmu = loadFMU('coupledClutches.fmu');
>> [time, yout, yname] = simulate(fmu, [0,10], 'Output', outconf);
>> plot(time,yout(:,1)); title(yname(1));
```

The resulting plot is seen in Figure 4.5.

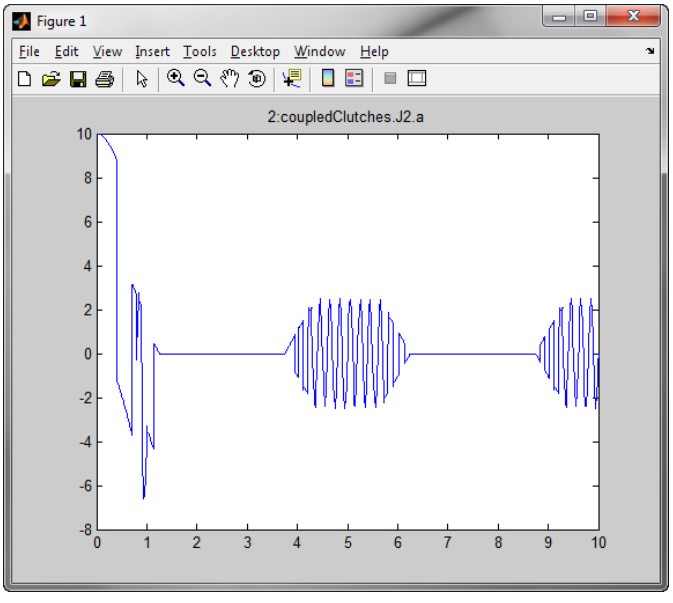


Figure 4.5 Plot using the results created in the Matlab workspace for the J2.a variable.

The simulation also generates a result file that is written to disk. This file contains the results from all the variables and parameters in the model. To load the same variable as above into the MATLAB workspace, the result file must first be loaded:

```
>> resData = loadDSResult('FMUTests.FMUs.CoupledClutches_results.txt');
```

To retreive and plot the variable, use the `getDSVariable` function:

```
>> [T, Y] = getDSVariable(resData, 'coupledClutches.J2.a');
>> plot(T, Y);
```

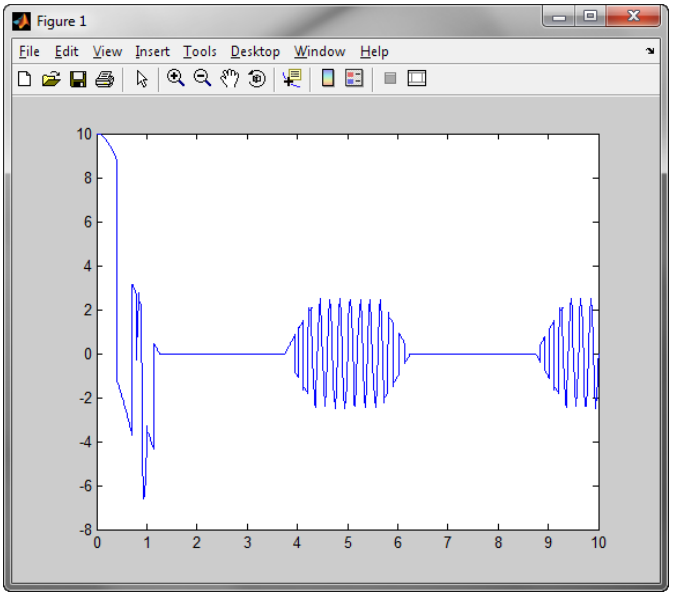


Figure 4.6 Plot using the result from file. The same J2.a variable is plotted.

4.4.3.1. Using custom solver (Model Exchange only)

If the model is instantiated with a model exchange class (`FMUModelME1` or `FMUModelME2`), the simulate function has the property `Solver` and `Options`. These are set to change the MATLAB solver and the options used by this solver. The solver property value is a string with the name of the solver. The options value is created using the `odeset` function.

In the example script below the default solver ode15s is replaced with ode23s. The solver options sets the relative tolerance to 1e-7 and the absolute tolerance to 1e-5.

```
>> fmu = loadFMU('coupledClutches.fmu');
>> odeopt = odeset('RelTol', 1e-7, 'AbsTol', 1e-5);
>> fmu.simulate([0,10], 'Options', odeopt, 'Solver', 'ode23s');
```

4.5. Upgrading to FMI 2.0

4.5.1. Converting from FMI 1.0 to FMI 2.0

When upgrading from FMI 1.0 to FMI 2.0, changes to existing scripts will be necessary. This section is a quick summary for reusing old scripts that are written for FMI 1.0 FMUs.

Changes when upgrading are presented in Table 4.2. Note that while the `initialize` method exists for all model types and versions, it takes different arguments depending on whether the model is an ME 1.0, CS 1.0 or a 2.0 FMU.

Another large change is the re-design of the high level functions for setting and getting values in the FMU. The new methods, `set` and `get`, gives better type check and can be used together with the `ScalarVariable(1/2)` classes. It is recommended to use these new methods, and `setValue/getValue` in the FMI 2.0 interface will give warnings about being deprecated.

Usage of the low-level FMI functions will have to be upgraded according to the Table 4.3. Accessing properties of the variables are now handled through the `ScalarVariable(1/2)` classes, see Table 4.4.

Table 4.2 Initialization and convenience function conversion table

FMI 1.0	FMI 2.0
`fmiInitialize`	`initialize`. Initialization can also be performed automatically by the `simulate` function.
`fmiInitializeSlave`	`initialize`. Initialization can also be performed automatically by the `simulate` function.
`fmiEventUpdate`	`eventUpdate`
`getValue`	`get` (`getValue` exists but is deprecated)
`setValue`	`set` (`setValue` exists but is deprecated)
`modelData`	Removed, use `get` methods for the different attributes. Variable counts should be obtained by counting elements in variable lists.

Table 4.3 Low-level FMI functions conversion table

FMI 1.0	FMI 2.0
`fmiFreeSlaveInstance`	`fmiFreeInstance`
`fmiFreeModelInstance`	`fmiFreeInstance`
`fmiGetModelTypesPlatform`	`fmiGetTypesPlatform`
`fmiGetNominalContinuousStates`	`fmiGetNominalsOfContinuousStates`
`fmiGetStateValueReferences`	`fmu.getStates().valueReference`
`fmiInstantiateSlave`	`fmiInstantiate` (consider using `loadFMU` as it also instantiates the FMU)
`fmiInstantiateModel`	`fmiInstantiate` (consider using `loadFMU` as it also instantiates the FMU)
`fmiResetSlave`	`fmiReset`

Table 4.4 Variable property functions conversion table

FMI 1.0	FMI 2.0
`getVariableAlias`	`getScalarVariable(name).aliasSet`

FMI 1.0	FMI 2.0
`getVariableAliasBase`	`getScalarVariable(name).baseAlias`
`getVariableCausality`	Has a different meaning in the FMI 2.0 standard, see the `ScalarVariable2` class
`getVariableDataType`	`getScalarVariable(name).type`
`getVariableDescription`	`getScalarVariable(name).description`
`getVariableFixed`	Does not exist in the FMI 2.0 standard
`getVariableMax`	`getScalarVariable(name).getMax`
`getVariableMin`	`getScalarVariable(name).getMin`
`getVariableNominal`	`getScalarVariable(name).getNominal`
`getVariableStart`	`getScalarVariable(name).start`
`getVariableValueref`	`getScalarVariable(name).valueReference`
`getVariableVariability`	Has a different meaning in the FMI 2.0 standard, see the `ScalarVariable2` class

4.5.2. Using both FMI 1.0 and FMI 2.0 in scripts

To make it easier to write scripts that can load and use both 1.0 and 2.0 FMUs, the functions `set`, `get`, `eventUpdate`, `initialize`, `aliasSet` and `baseAlias` have also been added to the FMI 1.0 interface. Now only the initialization and FMI specific properties/functions needs to be handled separately for the different FMI versions. These different cases can be constructed using the method `fmiGetVersion`.

Chapter 5. FMU export from Simulink

5.1. Introduction

A Simulink model can be exported as an FMU and imported in an FMI-compliant tool such as SimulationX or Dymola. This section describes how a Simulink model can be exported as an FMU. Code from a Simulink model is generated by Simulink Coder/Real-Time Workshop and is then wrapped in an FMU for FMI version 1.0 or 2.0 and Model Exchange or Co-Simulation .

5.2. Getting started

This tutorial gives a walk-through of the steps required to export an FMU for Model Exchange from Simulink.

1. Configure mex compiler.

   Simulink Coder/Real-Time Workshop selects the C compiler to be used. In order to provide a hint of which compiler to use, configure the mex compiler by executing the following command in MATLAB.

   ```
   >> mex -setup
   ```

   See Table 2.4 for the full list of supported compilers.

2. Create a simple Simulink model.

   Start Simulink to create a simple model.

   ```
   >> simulink
   ```

   The model used in this example integrates an input signal and outputs the integrated value.

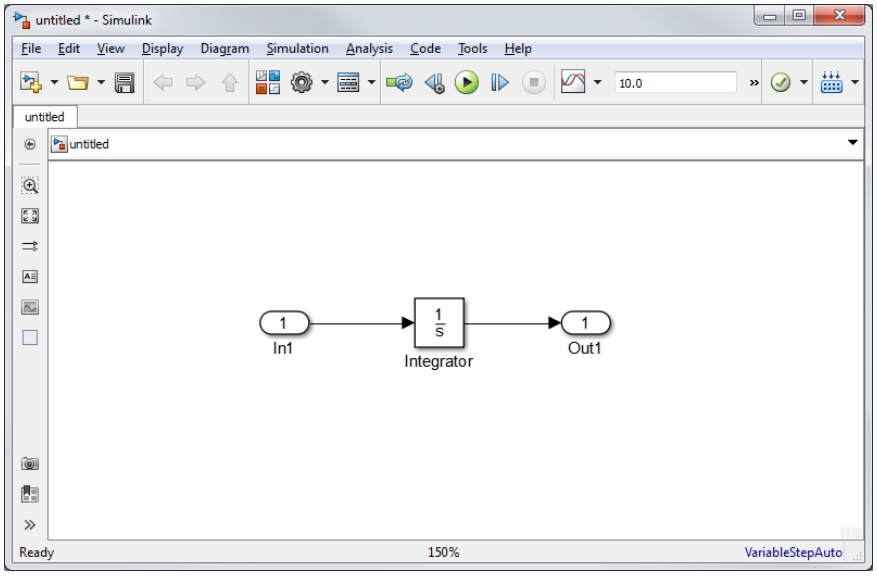


Figure 5.1 Create a new simple Simulink model.

3. Open the **Configuration Parameters** dialog.

   Open the **Configuration Parameters** dialog by clicking **Simulation** -> **Configuration Parameters...**. The dialog may look different depending on which MATLAB version is being used.

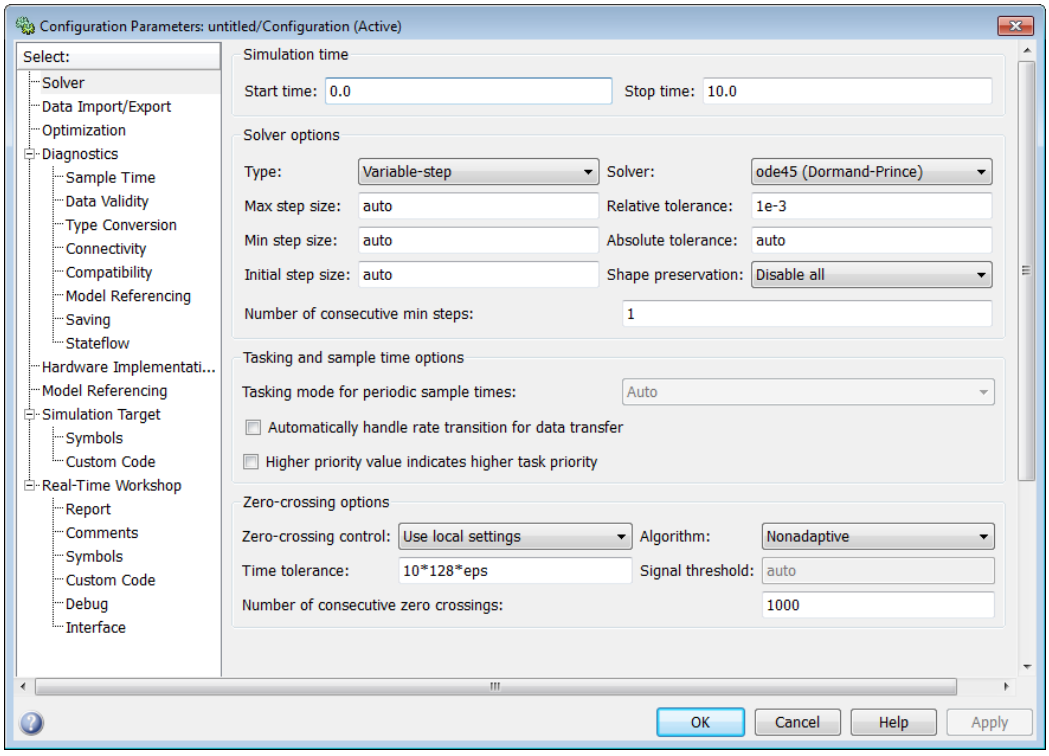


Figure 5.2 Configuration Parameters dialog for the simple Simulink model.

4. Go to **Real-Time Workshop** or **Code Generation**.

   Click on the **Real-Time Workshop** or **Code Generation** node in the tree view to the left. The name depends on the MATLAB version being used.

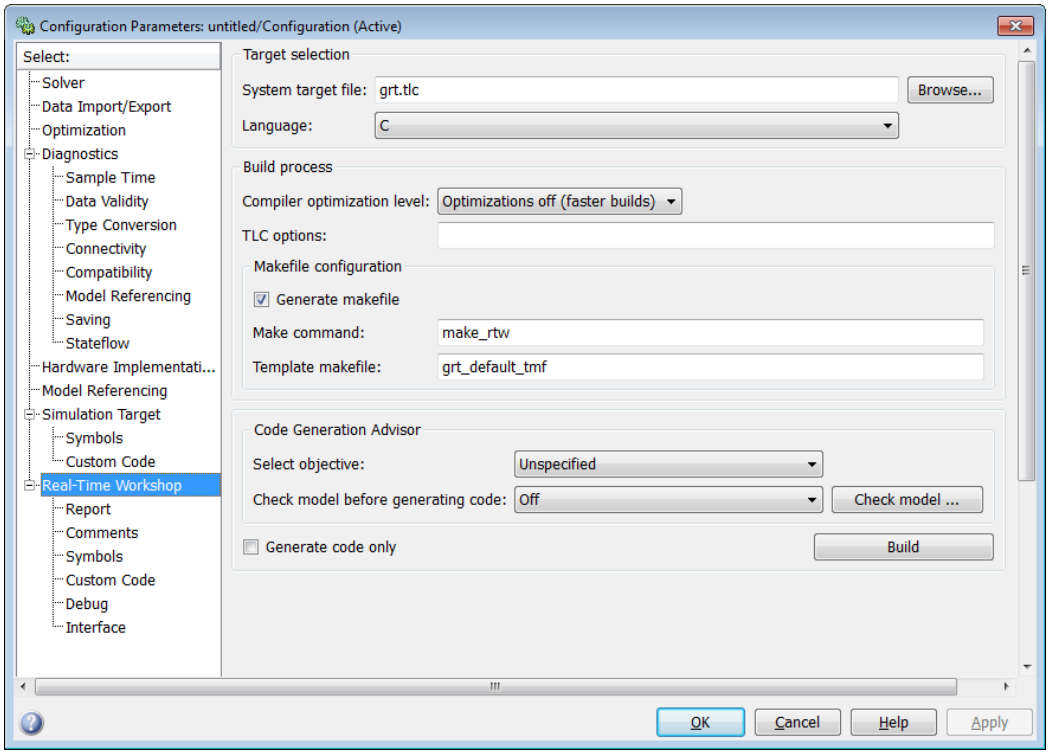


Figure 5.3 Go to the **Real-Time Workshop** or **Code Generation** tab.

5. Select target.

   Select **System target file** by clicking on the **Browse...** button. Select **fmu_me1.tlc** in the dialog that is open and then click OK.

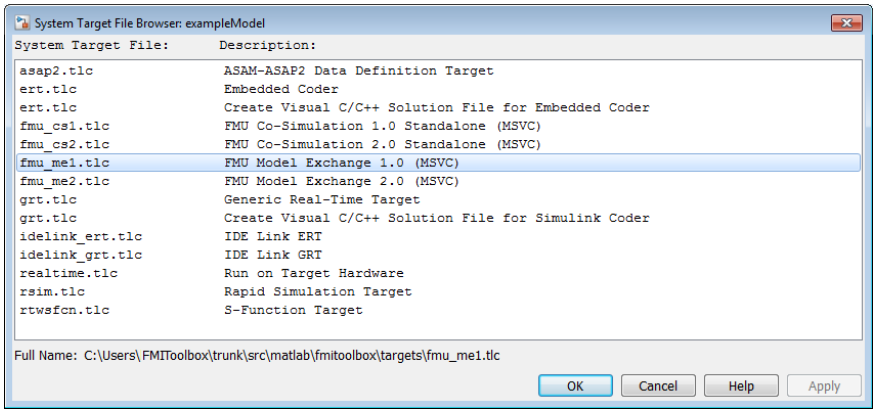


Figure 5.4 Select the system target file **fmu_me1.tlc** in the System Target File Browser.

6. Build target.

   Click **Apply** in lower right corner of the **Configuration Parameters** dialog and then press the **Build Model** button in the Simulink main window. In Figure 5.5 the files generated by the target are listed.

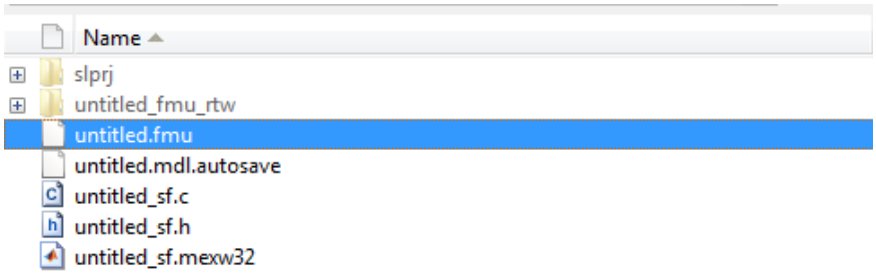


Figure 5.5 Files generated by the target.

7. Test FMU.

   The FMU block in the FMI Toolbox can be used to test the generated FMU. To import the FMU in Simulink see Chapter 3. In Figure 5.6 the created FMU is imported and a sinus signal is connected to the FMU input port and a scope is connected to the output port. The results can be seen in Figure 5.7; an integrated sinus signal with amplitude 1 and frequency 1 rad/s.

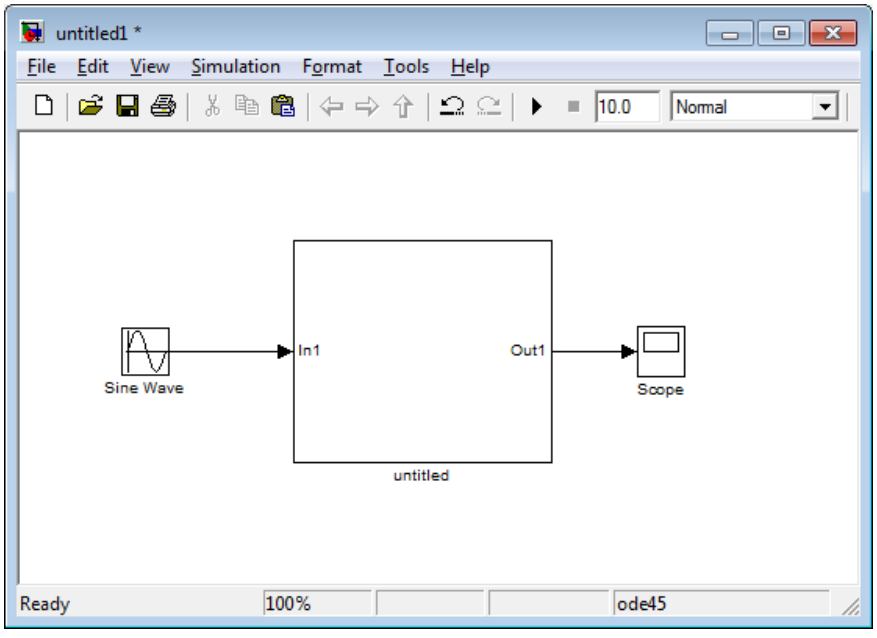


Figure 5.6 Created FMU imported in FMI Toolbox.

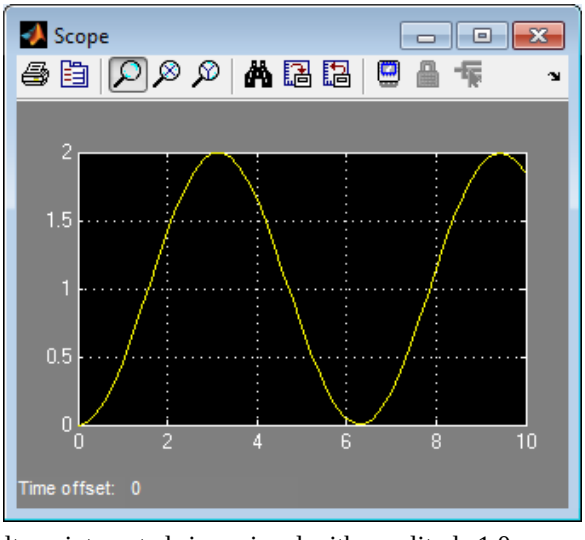


Figure 5.7 Simulation result, an integrated sinus signal with amplitude 1.0.

5.3. Simulink Coder targets for FMU export

To export an FMU from Simulink, a target (System target file) must be selected. The following system target files generates FMUs according to the different FMU types and versions of the FMI standard. See Section 1.2 for a general description of the different options.

Table 5.1 Overview of system target files

Targets (System target files)	FMU type	FMI version
fmu_me1.tlc	Model Exchange	1.0
fmu_me2.tlc	Model Exchange	2.0
fmu_cs1.tlc	Co-Simulation	1.0
fmu_cs2.tlc	Co-Simulation	2.0

The different targets are documented in subsections below. The Simulink model must be configured according to the requirements and limitations of the targets. The target may otherwise be unsuccessfully built.

For details of how to configure a Model Exchange target, see Section 5.6.

For details of how to configure a Co-Simulation target, see Section 5.5.

To select a target in Simulink, open the **Configuration Parameters** dialog and go to the **Real-Time Workshop**/**Code Generation** tab (the name depends on which MATLAB version is being used), see Figure 5.8. Click on the **Browse...** button to selected the **System target file** . In the browser dialog select one of the system target files for FMU export, e.g **fmu_me1.tlc** for exporting as FMI 1.0 Model Exchange, and click OK, see Figure 5.9. This will enable the FMU target in Simulink and a **FMU Export** tab will be visible in the tree view to the left.

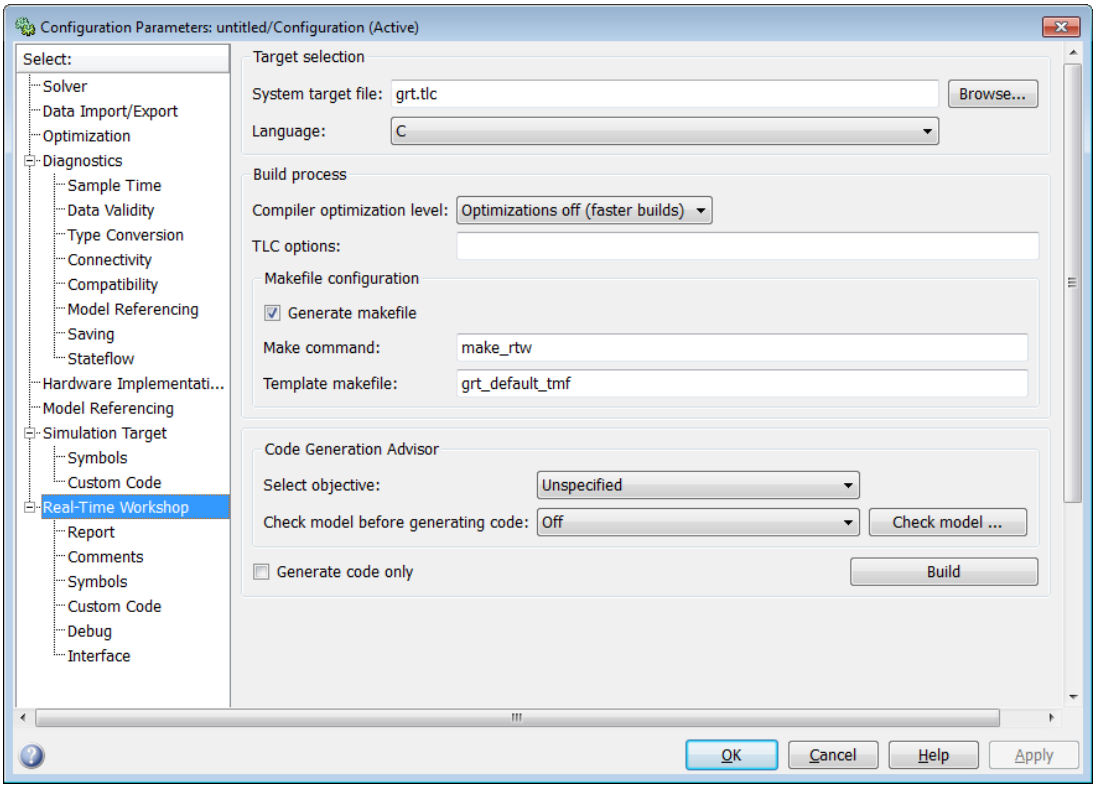


Figure 5.8 Real-Time Workshop or **Code Generation** tab selected in the **Configuration Parameters** dialog. Select one of the FMU export system target files e.g fmu_me1.tlc.

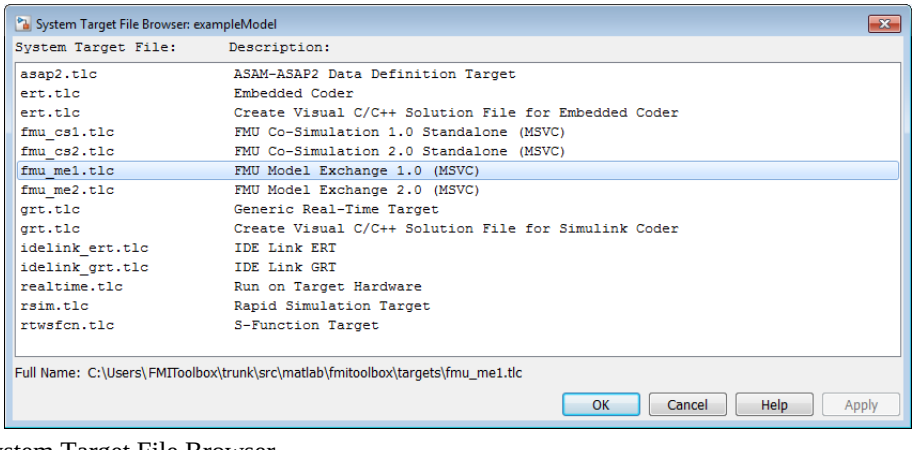


Figure 5.9 System Target File Browser.

All Simulink default settings are valid for building the target. Note that the C compiler must be selected (for some MATLAB versions) before the FMU target can be built, see Section 5.4 for more information.

The model configuration in Simulink is mainly controlled from the **Configuration Parameters** dialog. This documentation only provides a brief description of the settings in the **Configuration Parameters**. For a detailed description see MATLAB´s documentation.

5.4. Selecting MEX C compiler

The FMU export targets relies on the MATLAB built in functions *setup_for_visual.m* and *setup_for_visual_x64.m* to setup the appropriate MSVC C compiler environment. See the MATLAB help for these functions for more information. In order to provide a hint (for some MATLAB versions) of which compiler these function chooses, one can configure the regular mex compiler using the following command in the MATLAB and follow the instructions.

```
>> mex -setup
```

Note that the compiler Simulink Coder/Real-Time Workshop choose to use, must be supported, see Table 2.4.

5.5. Co-Simulation export

Co-Simulation specific information for FMU export is described in these sub sections.

5.5.1. Synchronization of time

The Co-Simulation FMU targets are derived from a **grt** based target that includes the fixed step solver used in the Simulink model. The solver and time synchronization mechanism generated by Simulink Coder is compiled into the FMU. The FMU keeps the internal synchronization mechanism generated by Simulink Coder in sync with external solver/master algorithm during simulation of the FMU. The Simulink Coder synchronization of time is based on an integer clock system which advances in time by a increasing an integer value (**clock tic**). The *real* time in the model is calculated from the **sample time** in the Simulink model and the current **clock tic**.

A Co-Simulation FMU advances in time by calling an FMI do-step[1] function. The do-step function takes real values, current simulation time (**tc**) and the step size (**hc**), as input arguments to calculate the **next simulation time** (**tc** + **hc**). The do-step function increases the clock tic of the model until it is equal to or only less then one **sample time** before the **next simulation time** (in other words, increases **clock tick** as long as **sample time** * **clock tic** <= **tc**+**hc** is valid).

Note that the step size (**hz**) argument to the FMI function should correspond to the sample time used in the Simulink model when the model is exported, otherwise the FMU may have trouble to synchronize the Simulink Coder clock system with the external solver/master algorithm.

[1] do-step corresponds to the FMI 2.0 function *fmi2DoStep* and FMI 1.0 function *fmiDoStep*.

5.5.2. Capability flags

A summary of how the FMI standard *capability* flags are set in the Co-Simulation FMUs are described in Table 5.2 and Table 5.3. For a detailed description of the capability attributes, see FMI standard.

Table 5.2

FMI 1.0 capability flags	Value
canHandleVariableCommunicationStepSize	True
canHandleEvents	False
canRejectSteps	False
canInterpolateInputs	False
maxOutputDerivativeOrder	0
canRunAsynchronuously	False
canSignalEvents	False
canBeInstantiatedOnlyOncePerProcess	False
canNotUseMemoryManagementFunctions	True

Table 5.3

FMI 2.0 capability flags	Value
needsExecutionTool	False
canHandleVariableCommunicationStepSize	True
canInterpolateInputs	False
maxOutputDerivativeOrder	False
canRunAsynchronuously	False
canBeInstantiatedOnlyOncePerProcess	True
canNotUseMemoryManagementFunctions	True
canGetAndSetFMUstate	False
canSerializeFMUstate	False
providesDirectionalDerivative	False

5.5.3. Configuration Parameters

5.5.3.1. Solver

Table 5.4 Configuration Parameters -> Solver tab options.

Option	Comment and value
Type:	Only Fixed-step solvers are supported.
Start time:	Used as value to the FMI attribute **startTime** in the **DefaultExperiment** element.
Stop time:	Used as value to the FMI attribute **stopTime** in the **DefaultExperiment** element.

5.5.3.2. Optimization

Table 5.5 Configuration Parameters -> Optimization tab options.

Option	Comment and value
Inline parameters	Enable/Disabled are valid values.

5.5.3.3. Real-Time Workshop/Code Generation

Table 5.6 Configuration Parameters -> Real-Time Workshop/Code Generation tab options.

Option	Comment and value
System target file:	fmu_cs1.tlc or fmu_cs2.tlc
Language:	C
TLC options:	
Generate makefile	Enabled
Make command:	make_rtw
Template makefile:	fmu_cs1_default_tmf or fmu_cs2_default_tmf
Selected objective:	Unspecified
Check model before generating code:	Off. Other values may also be valid such as *On (proceed with warnings)* and *On (stop for warnings)*.
Generate code only	Disabled

FMU Export

Table 5.7 Configuration Parameters -> Real-Time Workshop/Code Generation -> FMU Export tab options.

Option	Comment and value
Create black box FMU	Enable this option if only the necessary minimum information should be exposed in the FMU. Only inputs and outputs will be exposed for Co-Simulation FMUs and additionally states, derivatives and event indicators with neutral names for Model Exchange FMUs. Default is disabled.
Structured names for parameters	If enabled, the parameter variable names will be structured as specified in the FMI standard. For example, a name could be: *subsystem.block.parameter*. This option is enabled by default.
Include internal signals	If enabled, additional signals will be exposed in the exported FMU. The purpose of this is to make it easier to debug the FMU. The additional variables will have causality local. If **Include internal signals** is enabled, the optimization parameter **Signal storage reuse** needs to be disabled. More information about **Include internal signals** can be found in Section 5.8. **Include internal signals** is disabled by default.
Zip program:	Path to a custom zip tool that should be used for compressing the FMU, only available on Windows. Default value is an empty string which means the FMI-Toolbox uses its own implementation in Java. The tool compresses the FMU files into a single FMU file, *.fmu and must use the deflate algorithm. The program and options are called like: *"<ZipProgram>" <ZipOptions> "<OutputFolder>\<modelName>.fmu" "<FMUFilesDir>*"*. *<ZipProgram>* and *<ZipOptions>* corresponds to the value set in these two fields respectively. If for example 7-Zip is preferred over the Java implementation use: `C:\Program Files\7-Zip\7z.exe`
Zip options:	Options passed to the zip-tool, only available on Windows. This option is ignored if the default Java implementation is used. Default value is an empty string. The compression method used **must be deflate** in accordance with the FMI standard. If for example 7-Zip

Option	Comment and value
	is preferred over the Java implementation use: `a -r -tzip`

Report

Table 5.8 Configuration Parameters -> Real-Time Workshop/Code Generation -> Report tab options.

Option	Comment and value
Create code generation report	Enable/Disable
Launch report automatically	Enable/Disable

Comments

Table 5.9 Configuration Parameters -> Real-Time Workshop/Code Generation -> Comments tab options.

Option	Comment and value
Include comments	Enable/Disable
Simulink block / Stateflow object comments	Enable/Disable
MATLAB source code as comments	Enable/Disable
Show eliminated blocks	Enable/Disable
Verbose comments for SimulinkGlobal storage class	Enable/Disable

Symbols

Table 5.10 Configuration Parameters -> Real-Time Workshop/Code Generation -> Symbols tab options.

Option	Comment and value
Maximum identifier length:	Any value that is valid for Simulink.
Use the same reserved names as Simulation Target	Enable/Disable
Reserved names:	

Custom code

Table 5.11 Configuration Parameters -> Real-Time Workshop/Code Generation -> Custom code tab options.

Option	Comment and value
Use the same custom code settings as Simulation Target	Disable
Include custom C code in generated:	Not supported
Include list of additional:	Used to supply additional resources for User defined S-Functions. See, Section 5.5.4.

Debug

Table 5.12 Configuration Parameters -> Real-Time Workshop/Code Generation -> Debug tab options.

Option	Comment and value
Verbose build	Enable/Disable
Retain .rtw file	Enable/Disable
Profile TLC	Enable/Disable
Start TLC debugger when generating code	Enable/Disable
Start TLC coverage when generating code	Enable/Disable
Enable TLC assertion	Enable/Disable

5.5.4. Support for user defined S-Function blocks

For the Co-Simulation targets (**fmu_cs1.tlc** and **fmu_cs2.tlc**) the source file (*.c) for each S-Function is used. During code generation, Simulink coder also requires a compiled shared library of the S-function, but this libary will not be included in the exported FMU.

Any additional include directories, source files or libraries needed by the S-function must be specified in the **Custom Code** tab, see Figure 5.10.

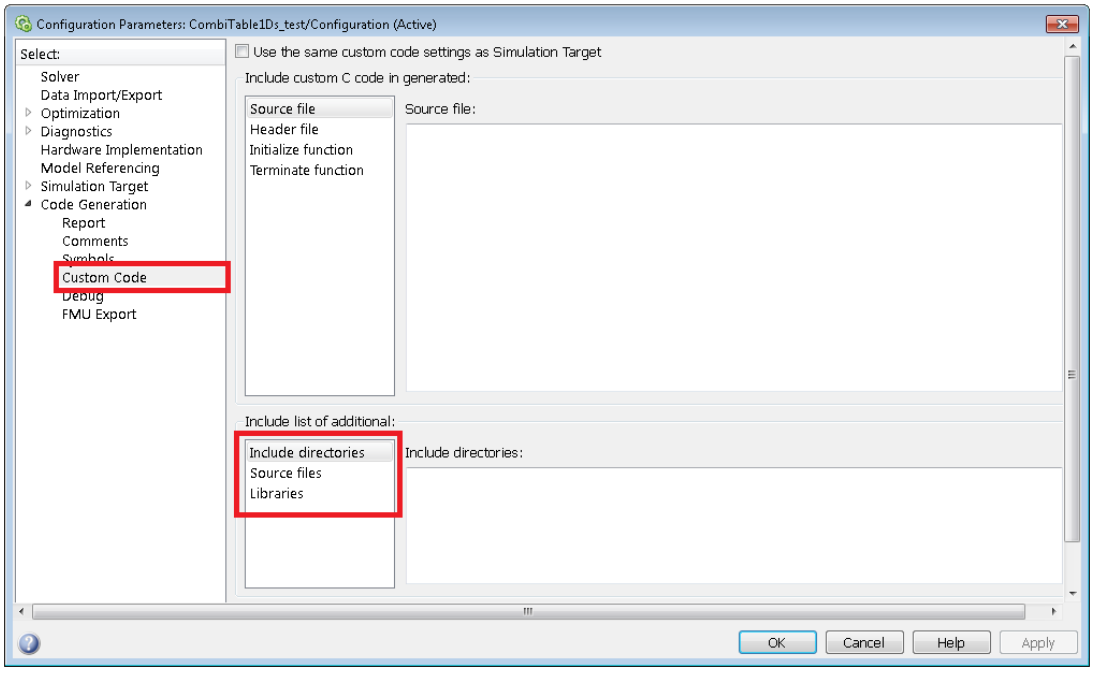


Figure 5.10 Custom Code tab and options for including additional resources.

Additional compiler flags can be added in the **Code Generation** tab after selecting the **fmu_cs1.tlc** or **fmu_cs2.tlc** target by modifying the **Make command**, see Figure 5.11. An example of modification would be to change the **Make command** to

'make_rtw FMIT_INTERMEDIATE_LIB_DIR=DEFAULT FMIT_ADDITIONAL_CFLAGS=-DMY_DEFINE'

this will add -DMY_DEFINE as a compiler flag for the target.

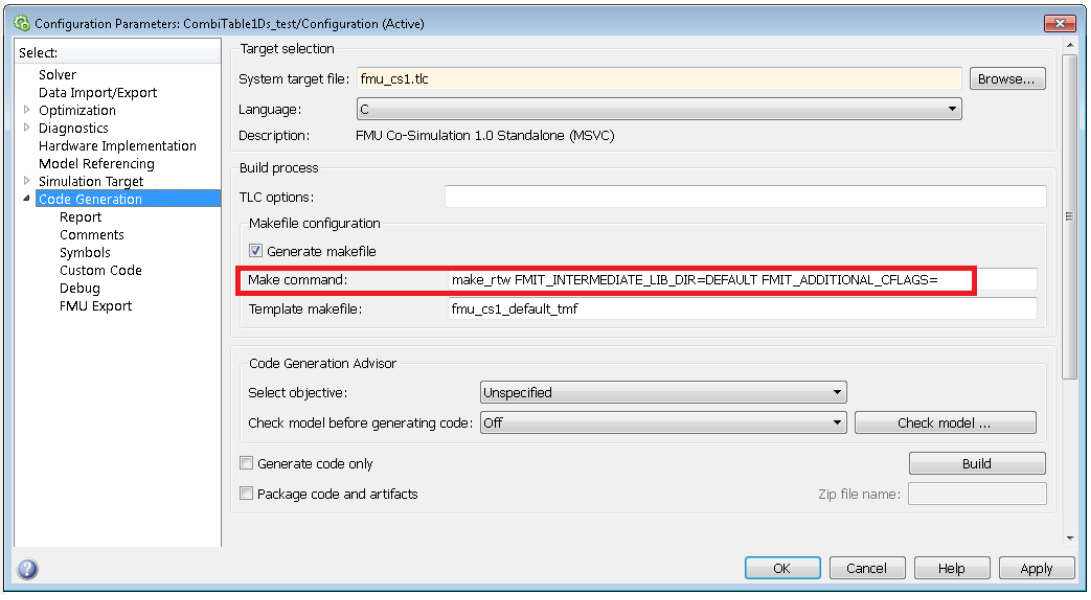


Figure 5.11 Code Generation tab and the field Make command.

It is recommended that the S-function follow the guidelines for writing non-inlined S-functions, see http://www.mathworks.se/help/rtw/ug/s-functions-for-code-generation.html?#f53130 for more information. The S-function may not call into MATLAB, e.g using mexCallMATLAB.

5.6. Model Exchange export

Model Exchange specific information for FMU export is described in this section.

5.6.1. Configuration Parameters

5.6.1.1. Solver

Model Exchange FMUs are exported without an embedded solver, but the code generated by Simulink Coder differs depending on solver type. If a fixed-step solver is selected in the Simulink model when exporting, the solver's step time is set as the sample time of the model, and each sample hit will cause an event even if the underlying dynamics behave in a continuous way. With variable-step solvers, the sampling is replaced by zero-crossing detection, generating events only when required by the model dynamics.

The preferred setting for ME FMU export should be a variable-step solver, as this prevents excessive events caused by sampling embedded into the FMU and allows event detection with greater precision than the selected step size. The exported FMU can still be simulated by both fixed-step and variable-step solvers.

Table 5.13 Configuration Parameters -> Solver tab options.

Option	Comment and value
Type:	All variable-step and fixed-step solvers are supported, but variable-step solvers should be preferred due to differences in the generated code.
Start time:	Used as value to the FMI attribute **startTime** in the **DefaultExperiment** element.
Stop time:	Used as value to the FMI attribute **stopTime** in the **DefaultExperiment** element.
Relative tolerance:	Used as value to the FMI attribute **tolerance** in the **DefaultExperiment** element if a VariableStep solver is selected. It is not set otherwise.

5.6.1.2. Optimization

Table 5.14 Configuration Parameters -> Optimization tab options.

Option	Comment and value
Inline parameters	Enable/Disabled are valid values.

5.6.1.3. Real-Time Workshop/Code Generation

Table 5.15 Configuration Parameters -> Real-Time Workshop/Code Generation tab options.

Option	Comment and value
System target file:	fmu_me1.tlc or fmu_me2.tlc
Language:	C
TLC options:	
Generate makefile	Enabled
Make command:	make_rtw
Template makefile:	fmu_me1_default_tmf or fmu_me2_default_tmf
Selected objective:	Unspecified
Check model before generating code:	Off. Other values may also be valid such as *On (proceed with warnings)* and *On (stop for warnings)*.
Generate code only	Disabled

FMU Export

Table 5.16 Configuration Parameters -> Real-Time Workshop/Code Generation -> FMU Export tab options.

Option	Comment and value
Support precompiled shared library S-functions (e.g *.mexw64)	Build model containing precompiled S-functions. Default value is disabled. See, Section 5.6.2.
Support precompiled object S-functions (e.g *.obj). Takes precedence over precompiled shared library (e.g *.mexw64)	Link with S-function´s object file. Takes precedence over precompiled shared library. Default value is disabled. See, Section 5.6.2.
Compile and link with MATLAB. Must be enabled if any precompiled file depends on MATLAB	Compile and link with MATLAB. Default value is disabled. See, Section 5.6.2.
Create black box FMU	Enable this option if only the necessary minimum information should be exposed in the FMU. Only inputs and outputs will be exposed for Co-Simulation FMUs and additionally states, derivatives and event indicators with neutral names for Model Exchange FMUs. Default is disabled.
Structured names for parameters	If enabled, the parameter variable names will be structured as specified in the FMI standard. For example, a name could be: *subsystem.block.parameter*. This option is enabled by default.
Include internal signals	If enabled, additional signals will be exposed in the exported FMU. The purpose of this is to make it easier to debug the FMU. The additional variables will have causality local. If **Include internal signals** is enabled, the optimization parameter **Signal storage reuse** needs to be disabled. More information about **Include internal signals** can be found in Section 5.8. **Include internal signals** is disabled by default.
Zip program:	Path to a custom zip tool that should be used for compressing the FMU, only available on Windows. Default value is an empty string which means the FMI-Toolbox uses its own implementation in Java. The tool compresses the FMU files into a single FMU file, *.fmu and must use the deflate algorithm. The program and options are called like: *"<ZipProgram>" <ZipOptions> "<OutputFolder>\<modelName>.fmu" "<FMUFilesDir>*"*. *<ZipProgram>* and *<ZipOptions>* corresponds to the value set in these two fields respectively. If for example 7-Zip is preferred over the

Option	Comment and value
	Java implementation use: `C:\Program Files\7-Zip\7z.exe`
Zip options:	Options passed to the zip-tool, only available on Windows. This option is ignored if the default Java implementation is used. Default value is an empty string. The compression method used **must be deflate** in accordance with the FMI standard. If for example 7-Zip is preferred over the Java implementation use: `a -r -tzip`

Report

Table 5.17 Configuration Parameters -> Real-Time Workshop/Code Generation -> Report tab options.

Option	Comment and value
Create code generation report	Enable/Disable
Launch report automatically	Enable/Disable

Comments

Table 5.18 Configuration Parameters -> Real-Time Workshop/Code Generation -> Comments tab options.

Option	Comment and value
Include comments	Enable/Disable
Simulink block / Stateflow object comments	Enable/Disable
MATLAB source code as comments	Enable/Disable
Show eliminated blocks	Enable/Disable
Verbose comments for SimulinkGlobal storage class	Enable/Disable

Symbols

Table 5.19 Configuration Parameters -> Real-Time Workshop/Code Generation -> Symbols tab options.

Option	Comment and value
Maximum identifier length:	Any value that is valid for Simulink.
Use the same reserved names as Simulation Target	Enable/Disable
Reserved names:	

Custom code

Table 5.20 Configuration Parameters -> Real-Time Workshop/Code Generation -> Custom code tab options.

Option	Comment and value
Use the same custom code settings as Simulation Target	Disable
Include custom C code in generated:	Not supported
Include list of additional:	Not supported

Debug

Table 5.21 Configuration Parameters -> Real-Time Workshop/Code Generation -> Debug tab options.

Option	Comment and value
Verbose build	Enable/Disable
Retain .rtw file	Enable/Disable
Profile TLC	Enable/Disable
Start TLC debugger when generating code	Enable/Disable
Start TLC coverage when generating code	Enable/Disable
Enable TLC assertion	Enable/Disable

5.6.2. Support for user defined S-Function blocks

For the Model Exchange targets (**fmu_me1.tlc** and **fmu_me2.tlc**) each S-Function need to be precompiled, either as a shared library S-Function (*.mex32 or *.mex64) or an object S-Function (*.obj). Then the respective options **Support precompiled shared library S-functions** and **Support precompiled object S-functions** can be used and if any of the precompiled files depends on MATLAB the option **Compile and link with MATLAB** must be enabled.

It is recommended that the S-function follow the guidelines for writing non-inlined S-functions, see http://www.mathworks.se/help/rtw/ug/s-functions-for-code-generation.html?#f53130 for more information. The S-function may not call into MATLAB, e.g using mexCallMATLAB.

If an S-function uses parameter values in its mdlStart callback, these will always take on their default values, regardless of parameters set in the exported FMU. They will still be marked as FMU parameters, since Simulink Coder and FMI Toolbox cannot know where in the S-function code they are used. It is recommended that S-functions inlined into ME FMUs initialize their parameters in a later callback, e.g. during the first call to mdlOutputs.

Note: including a compiled S-Function in the FMU will give the FMU the same dependencies as that of the S-Function. In this case, the FMU will typically only run on machines that have MATLAB installed.

5.7. Parameters

This section describes the support for different Simulink parameter settings and how Simulink parameters are exposed as FMI parameters in an FMU.

All Simulink parameters setting are supported for the FMU targets. For a detailed list of parameter settings in Simulink and how they are exposed in the FMU, see Table 5.22. Note that parameters with start values NaN and Inf will be calculated parameters in the FMU.

Table 5.22 Parameter settings for code generation in Simulink.

Parameter settings in Simulink	FMU targets	Supported	FMU parameter
Default settings	fmu_cs1.tlc	Yes	parameter
	fmu_cs2.tlc	Yes	tunable parameter
	fmu_me1.tlc	Yes	parameter
	fmu_me2.tlc	Yes	parameter
Global (tunable) parameters[a] with storage class: Simulink-Global()	fmu_cs1.tlc	Yes	parameter
	fmu_cs2.tlc	Yes	tunable parameter
	fmu_me1.tlc	Yes	parameter
	fmu_me2.tlc	Yes	parameter
Inline parameters[a]	fmu_cs1.tlc	Yes	Not exposed
	fmu_cs2.tlc	Yes	Not exposed
	fmu_me1.tlc	Yes	Not exposed
	fmu_me2.tlc	Yes	Not exposed

[a]Option is set from the **Configuration Parameters** -> **Optimization** tab options

The names of the Global tunable parameters are explicitly specified by the user. For other parameters, their names in the exported FMU will reflect the structure of the model. More precisely, they will have names on the form {"subsystem name".}"block name"."block parameter name". Where the content in {} is repeated zero or more times depending on the number of nested Subsystems the Simulink block is placed in.

5.8. Internal signals

Using the option **Include internal signals** will make additional internal signals available in an exported FMU. The purpose of this is to make it easier to debug the FMU. The option is available in **Simulation > Model Configuration Parameters > Code Generation > FMU Export**. **Include internal signals** will not have any effect if the export option **Create black box FMU** is enabled. If **Include internal signals** is enabled, the optimization parameter **Signal storage reuse** needs to be disabled. **Include internal signals** is disabled by default.

Not all internal signals can be exposed in the generated FMU. Which signals are exposed also depends on whether exporting as a Model Exchange FMU or a Co-Simulation FMU. **Block reduction** may also affect which variables will be visible, see: https://se.mathworks.com/help/simulink/gui/block-reduction.html. To understand which additional signals will be exposed we will be looking at two simple examples.

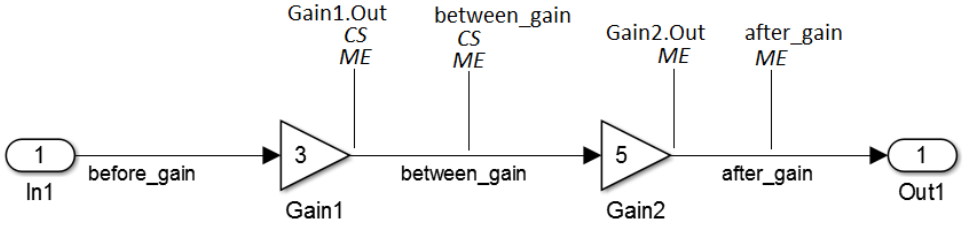


Figure 5.12 A system that when exported as an FMU with internal signals will have the additional shown variables. Which variables are exposed depends on if exporting as an Model Exchange FMU or a Co-Simulation FMU.

First we look at the system found in Figure 5.12. Here we can see that no block inputs are exposed in the resulting FMU in either case. In the Model Exchange case, block outputs and their respective named signals will be exposed. In the Co-Simulation case, block outputs and named signals will only be exposed if they connect to a block and not when they connect to model outputs. The variables will have structured names in the exported FMU, for example **Gain1.Out** as shown in the figure. All internal signals that are exposed as variables in the exported FMU will have the causality local, which are read only variables.

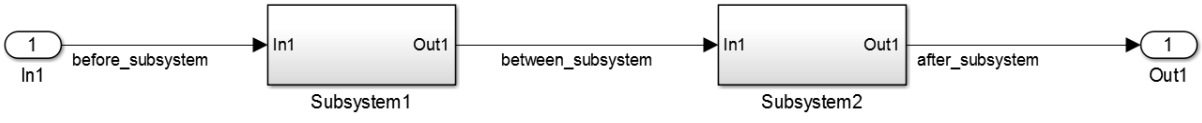


Figure 5.13 A system that when exported as an FMU with internal signals will have no additional variables at this level in the system. See Subsystem1 in Figure 5.14 and Subsystem2 in Figure 5.15.

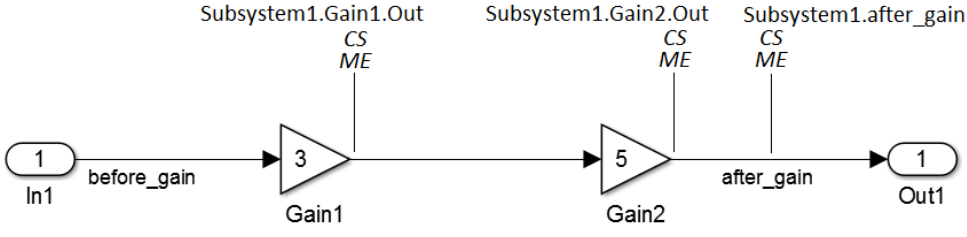


Figure 5.14 A subsystem that when exported as an FMU with internal signals will have the additional shown variables. In this subsystem, the same variables are exposed when exporting as a Model Exchange FMU compared to a Co-Simulation FMU. This is Subsystem1 of the system found in Figure 5.13

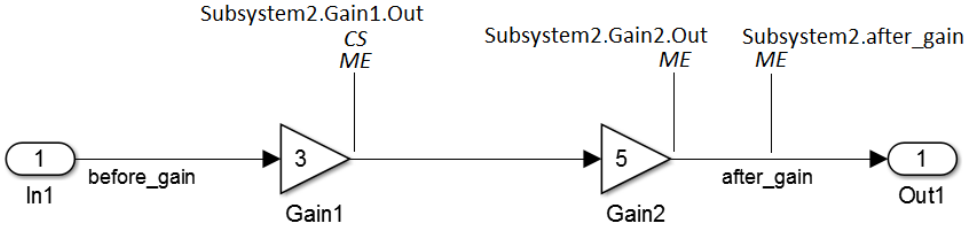


Figure 5.15 A subsystem that when exported as an FMU with internal signals will have the additional shown variables. Which variables are exposed depends on if exporting as a Model Exchange FMU or a Co-Simulation FMU. This is Subsystem2 of the system found in Figure 5.13

Secondly we look at the system found in Figure 5.13. This system is comprised of two subsystems where each subsystem contains two gain blocks, Subsystem1 can be seen in Figure 5.14 and Subsystem2 can be seen in Figure 5.15. In Figure 5.13 we can see that no additional variables will be exposed at the top level when connecting subsystems. Signals that have not been given a name will also not be exposed, as seen in Figure 5.14 and Figure 5.15. In the Model Exchange case, outputs from non-subsystem blocks in subsystems and associated named signals will be visible. In the Co-Simulation case, outputs from non-subsystem blocks in subsystems and associated named signals will be visible if they indirecly or direcly connect to a non-subsystem block and not a top level output.

A limitation of the internal signals is that the structured naming is not merged with the structured naming of the parameters. That is, the parameters for a subsystem may not be displayed together with the internal signal variables in some importing tools.

5.8.1. Test points

It is possible to make more signals and outputs available in exported FMU by using test points. General information about what test points are can be found here: https://se.mathworks.com/help/simulink/ug/working-with-test-points.html. Test points can be used to expose signals between subsystems as variables in the exported FMU. All variables associated with test points in the exported FMU will have a name that starts with **TestPoints** and will have the causality local.

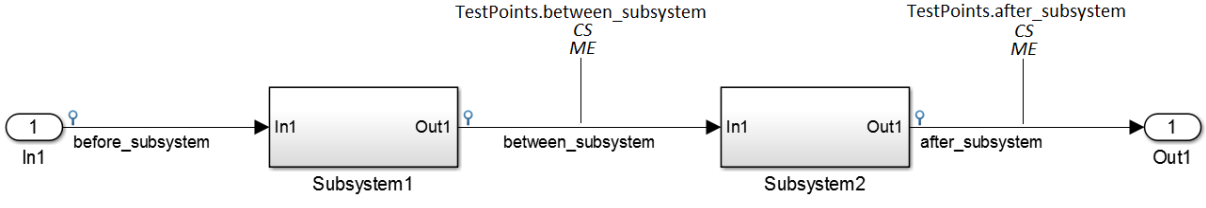


Figure 5.16 A system that is the same as the one found in Figure 5.13 but with added test points. The additional exposed variables are shown, which will be available in both Model Exchange and Co-Simulation FMUs.

In Figure 5.16 test points are added to all three of the named signals. Signals coming from top system inputs will not be exposed even if we add test points to them. Signals coming from subsystem outputs will be exposed if and only if they are named.

5.9. Supported data types

Supported data types in the FMU export are the same as those supported by Simulink, see Table 5.23. Simulink defines a Boolean to have the values 1 for true and 0 for false.

Table 5.23 Supported data types in Simulink.

Name	Description
double	Double-precision floating point
single	Single-precision floating point
int8	Signed 8-bit integer
uint8	Unsigned 8-bit integer
int16	Signed 16-bit integer
uint16	Unsigned 16-bit integer
int32	Signed 32-bit integer
uint32	Unsigned 32-bit integer

To view the data types supported for different blocks, execute the following command in the MATLAB Command Window.

```
>> showblockdatatypetable
```

Simulink data types are mapped to FMI data types according to Table 5.24 below.

Table 5.24 Simulink data types conversion to FMI data types table.

Simulink data type	FMI 1.0 data type	FMI 2.0 data type
double	fmiReal	fmi2Real
single	fmiReal	fmi2Real
int8	fmiInteger	fmi2Integer
uint8	fmiInteger	fmi2Integer
int16	fmiInteger	fmi2Integer
uint16	fmiInteger	fmi2Integer
int32	fmiInteger	fmi2Integer
uint32	fmiInteger	fmi2Integer
Simulink Boolean values	fmiBoolean	fmi2Boolean

Complex input and output ports are **NOT** supported, since there is no corresponding data type in the FMI standard. Complex parameters will not be exposed in the FMU.

Fixed-point input and output ports are **NOT** supported, since there is no corresponding data type in the FMI standard. Fixed-point parameters will not be exposed in the FMU.

For more information about data types in Simulink, go to http://www.mathworks.se/help/simulink/ug/working-with-data-types.html.

5.10. Supported blocks

The tables below lists all blocks that have been tested. If the Comment column is empty, the block is fully supported. If the Comment for a block is not empty, it may indicate that the usage of the block is restricted or not supported.

All blocks are tested using using a variable time step solver for the Model Exchange target and a fixed step solver for the Co-Simulation target. The blocks has not been tested using frame based sampling mode. Frame based sampling mode requires a Signal Processing Blockset license.

Table 5.25 FMI blocks.

Block	Comment
fmu_cs_lib/FMU CS	See table Table 2.6 for supported source-code FMUs. See Section 3.3.7 for using FMUs with shared libraries.
fmu_me_lib/FMU ME	See table Table 2.6 for supported source-code FMUs. See Section 3.3.7 for using FMUs with shared libraries.

Table 5.26 Continuous blocks.

simulink/Continuous/Integrator	
simulink/Continuous/Integrator Limited	
simulink/Continuous/Integrator, Second-Order	
simulink/Continuous/Integrator, Second-Order Limited	
simulink/Continuous/State-Space	
simulink/Continuous/Transfer Fcn	
simulink/Continuous/Zero-Pole	
simulink/Continuous/PID Controller	
simulink/Continuous/PID Controller (2DOF)	
simulink/Continuous/Transport Delay	
simulink/Continuous/Variable Time Delay	Partial supported. Generates different results.

simulink/Continuous/Variable Transport Delay	Partial supported. Generates different results.
simulink/Continuous/Derivative	Partial supported. Derivative approximation is dependent on the length of the integrator step which causes the results to be different.

Table 5.27 Discontinuities blocks.

simulink/Discontinuities/Saturation	
simulink/Discontinuities/Dead Zone	
simulink/Discontinuities/Rate Limiter	
simulink/Discontinuities/Saturation Dynamic	
simulink/Discontinuities/Dead Zone Dynamic	
simulink/Discontinuities/Rate Limiter Dynamic	
simulink/Discontinuities/Backlash	Partial supported. Generates different results.
simulink/Discontinuities/Relay	
simulink/Discontinuities/Quantizer	
simulink/Discontinuities/Hit Crossing	
simulink/Discontinuities/Coulomb & Viscous Friction	
simulink/Discontinuities/Wrap To Zero	

Table 5.28 Discrete blocks.

simulink/Discrete/Unit Delay	
simulink/Discrete/Integer Delay (renamed in 2011b to *Delay*)	
simulink/Discrete/Delay (new since 2011b)	
simulink/Discrete/Tapped Delay	
simulink/Discrete/Discrete-Time Integrator	
simulink/Discrete/Discrete Transfer Fcn	
simulink/Discrete/Discrete Filter	
simulink/Discrete/Discrete Zero-Pole	
simulink/Discrete/Difference	
simulink/Discrete/Discrete Derivative	
simulink/Discrete/Discrete State-Space	
simulink/Discrete/Transfer Fcn First Order	

simulink/Discrete/Transfer Fcn Lead or Lag	
simulink/Discrete/Transfer Fcn Real Zero	
simulink/Discrete/Discrete PID Controller	
simulink/Discrete/Discrete PID Controller (2DOF)	
simulink/Discrete/Discrete FIR Filter	
simulink/Discrete/Memory	
simulink/Discrete/First-Order Hold	
simulink/Discrete/Zero-Order Hold	

Table 5.29 Logic and Bit Operations blocks.

simulink/Logic and Bit Operations/Logical Operator	
simulink/Logic and Bit Operations/Relational Operator	
simulink/Logic and Bit Operations/Interval Test	
simulink/Logic and Bit Operations/Interval Test Dynamic	
simulink/Logic and Bit Operations/Combinatorial Logic	
simulink/Logic and Bit Operations/Compare To Zero	
simulink/Logic and Bit Operations/Compare To Constant	
simulink/Logic and Bit Operations/Bit Set	
simulink/Logic and Bit Operations/Bit Clear	
simulink/Logic and Bit Operations/Bitwise Operator	
simulink/Logic and Bit Operations/Shift Arithmetic	
simulink/Logic and Bit Operations/Extract Bits	
simulink/Logic and Bit Operations/Detect Increase	
simulink/Logic and Bit Operations/Detect Decrease	
simulink/Logic and Bit Operations/Detect Change	
simulink/Logic and Bit Operations/Detect Rise Positive	
simulink/Logic and Bit Operations/Detect Rise Nonnegative	
simulink/Logic and Bit Operations/Detect Fall Negative	
simulink/Logic and Bit Operations/Detect Fall Nonpositive	

Table 5.30 Lookup Tables blocks.

simulink/Lookup Tables/Lookup Table	
simulink/Lookup Tables/Lookup Table (2-D)	
simulink/Lookup Tables/Lookup Table (n-D)	
simulink/Lookup Tables/Prelookup	
simulink/Lookup Tables/Interpolation Using Prelookup	
simulink/Lookup Tables/Direct Lookup Table (n-D)	
simulink/Lookup Tables/Lookup Table Dynamic	
simulink/Lookup Tables/Sine	
simulink/Lookup Tables/Cosine	

Table 5.31 Math Operations blocks.

simulink/Math Operations/Sum	
simulink/Math Operations/Add	
simulink/Math Operations/Subtract	
simulink/Math Operations/Sum of Elements	
simulink/Math Operations/Bias	
simulink/Math Operations/Weighted Sample Time Math	Not supported when continuous sample times are used. See note[a]
simulink/Math Operations/Gain	
simulink/Math Operations/Slider Gain	
simulink/Math Operations/Product	
simulink/Math Operations/Divide	
simulink/Math Operations/Product of Elements	
simulink/Math Operations/Dot Product	
simulink/Math Operations/Sign	
simulink/Math Operations/Abs	
simulink/Math Operations/Unary Minus	
simulink/Math Operations/Math Function	
simulink/Math Operations/Rounding Function	
simulink/Math Operations/Polynomial	
simulink/Math Operations/MinMax	

simulink/Math Operations/MinMax Running Resettable	
simulink/Math Operations/Trigonometric Function	
simulink/Math Operations/Sine Wave Function	
simulink/Math Operations/Algebraic Constraint	Not supported. Algebraic loops are not supported in generated code.
simulink/Math Operations/Sqrt	
simulink/Math Operations/Signed Sqrt	
simulink/Math Operations/Reciprocal Sqrt	
simulink/Math Operations/Assignment	
simulink/Math Operations/Find Nonzero Elements	
simulink/Math Operations/Matrix Concatenate	
simulink/Math Operations/Vector Concatenate	
simulink/Math Operations/Permute Dimensions	
simulink/Math Operations/Reshape	
simulink/Math Operations/Squeeze	
simulink/Math Operations/Complex to Magnitude-Angle	
simulink/Math Operations/Magnitude-Angle to Complex	
simulink/Math Operations/Complex to Real-Imag	
simulink/Math Operations/Real-Imag to Complex	

[a]Not supported by the S-function *CodeFormat* which the the FMU target is derived from.

Table 5.32 Model Verification blocks.

simulink/Model Verification/Check Static Lower Bound	See note [a].
simulink/Model Verification/Check Static Upper Bound	See note [a].
simulink/Model Verification/Check Static Range	See note [a].
simulink/Model Verification/Check Static Gap	See note [a].
simulink/Model Verification/Check Dynamic Lower Bound	See note [a].
simulink/Model Verification/Check Dynamic Upper Bound	See note [a].

simulink/Model Verification/Check Dynamic Range	See note [a].
simulink/Model Verification/Check Dynamic Gap	See note [a].
simulink/Model Verification/Assertion	See note [a].
simulink/Model Verification/Check Discrete Gradient	See note [a]. Requires fixed-step solver, see note [b].
simulink/Model Verification/Check Input Resolution	See note [a].

[a]The block option *Stop simulation when assertion fails* does not affect the FMU simulation. Instead use the *Enable assertion* option to decide if the FMU contains the assertion from the block.

[b]Limited by the S-function *CodeFormat* which the the FMU target is derived from.

Table 5.33 Model-Wide Utilities blocks.

simulink/Model-Wide Utilities/Trigger-Based Linearization	Not supported, see note [a].
simulink/Model-Wide Utilities/Timed-Based Linearization	Not supported, see note [a].
simulink/Model-Wide Utilities/Model Info	
simulink/Model-Wide Utilities/DocBlock	
simulink/Model-Wide Utilities/Block Support Table	

[a]TLC-file for the block is missing. Code for the block cannot be generated.

Table 5.34 Ports & Subsystems blocks.

simulink/Ports & Subsystems/In1	
simulink/Ports & Subsystems/Out1	
simulink/Ports & Subsystems/Trigger	
simulink/Ports & Subsystems/Enable	
simulink/Ports & Subsystems/Function-Call Generator	
simulink/Ports & Subsystems/Function-Call Split	
simulink/Ports & Subsystems/Subsystem	
simulink/Ports & Subsystems/Atomic Subsystem	
simulink/Ports & Subsystems/CodeReuseSubsystem	
simulink/Ports & Subsystems/Model	Not supported. FMU target is not model reference compliant.
simulink/Ports & Subsystems/Model Variants	Not supported. FMU target is not model reference compliant.
simulink/Ports & Subsystems/Function-Call Subsystem	
simulink/Ports & Subsystems/Configurable Subsystem	

simulink/Ports & Subsystems/Variant Subsystem	Not supported. FMU target is not model reference compliant.
simulink/Ports & Subsystems/For Each Subsystem	Not supported for Model Exchange, see note [a].
simulink/Ports & Subsystems/For Iterator Subsystem	
simulink/Ports & Subsystems/While Iterator Subsystem	
simulink/Ports & Subsystems/Triggered Subsystem	
simulink/Ports & Subsystems/Enabled Subsystem	
simulink/Ports & Subsystems/Enabled and Triggered Subsystem	
simulink/Ports & Subsystems/If	
simulink/Ports & Subsystems/If Action Subsystem	
simulink/Ports & Subsystems/Switch Case	
simulink/Ports & Subsystems/Switch Case Action Subsystem	

[a]Block is not supported for generation of a Simulink Coder/Real-Time Workshop target.

Table 5.35 Signal Attributes blocks.

simulink/Signal Attributes/Data Type Conversion	
simulink/Signal Attributes/Data Type Duplicate	
simulink/Signal Attributes/Data Type Propagation	
simulink/Signal Attributes/Data Type Scaling Strip	
simulink/Signal Attributes/Data Type Conversion Inherited	
simulink/Signal Attributes/IC	
simulink/Signal Attributes/Signal Conversion	
simulink/Signal Attributes/Rate Transition	
simulink/Signal Attributes/Signal Specification	
simulink/Signal Attributes/Bus to Vector	
simulink/Signal Attributes/Probe	
simulink/Signal Attributes/Weighted Sample Time	
simulink/Signal Attributes/Width	

Table 5.36 Signal Routing blocks.

simulink/Signal Routing/Bus Creator	

simulink/Signal Routing/Bus Selector	
simulink/Signal Routing/Bus Assignment	
simulink/Signal Routing/Vector Concatenate	
simulink/Signal Routing/Mux	
simulink/Signal Routing/Demux	
simulink/Signal Routing/Selector	
simulink/Signal Routing/Index Vector	
simulink/Signal Routing/Merge	
simulink/Signal Routing/Environment Controller	
simulink/Signal Routing/Manual Switch	
simulink/Signal Routing/Multiport Switch	
simulink/Signal Routing/Switch	
simulink/Signal Routing/From	
simulink/Signal Routing/Goto Tag Visibility	
simulink/Signal Routing/Goto	
simulink/Signal Routing/Data Store Read	
simulink/Signal Routing/Data Store Memory	
simulink/Signal Routing/Data Store Write	
simulink/Signal Routing/State Reader	
simulink/Signal Routing/State Writer	

Table 5.37 Sinks blocks.

simulink/Sinks/Out1	
simulink/Sinks/Terminator	
simulink/Sinks/To File	Supports only "Save format" set to *Array*. *TimeSeries* are not supported by Simulink Coder/Real-Time Workshop generated code.
simulink/Sinks/To Workspace	
simulink/Sinks/Scope	
simulink/Sinks/Floating Scope	
simulink/Sinks/XY Graph	
simulink/Sinks/Display	

simulink/Sinks/Stop Simulation	

Table 5.38 Sources blocks.

simulink/Sources/In1	See note[a].
simulink/Sources/Ground	See note [a].
simulink/Sources/From File	See note [a].
simulink/Sources/From Workspace	See note [a].
simulink/Sources/Constant	See note [a].
simulink/Sources/Enumerated Constant	Not supported. Enumerator is not supported by the target.
simulink/Sources/Signal Builder	See note [a].
simulink/Sources/Ramp	See note [a].
simulink/Sources/Step	See note [a].
simulink/Sources/Sine Wave	See note [a].
simulink/Sources/Signal Generator	See note [a].
simulink/Sources/Chirp Signal	See note [a].
simulink/Sources/Random Number	See note [a].
simulink/Sources/Uniform Random Number	See note [a].
simulink/Sources/Band-Limited White Noise	See note [a].
simulink/Sources/Pulse Generator	Uses a variable sample time, see note [b].
simulink/Sources/Repeating Sequence	See note [a].
simulink/Sources/Repeating Sequence Stair	See note [a].
simulink/Sources/Repeating Sequence Interpolated	See note [a].
simulink/Sources/Clock	See note [a].
simulink/Sources/Digital Clock	See note [a].
simulink/Sources/Counter Free-Running	Due to the nature of the block, the output depends on how many times the FMI functions are called which varies between different FMI import tools and solver settings.
simulink/Sources/Counter Limited	See note [a].

[a]A source block should specify it's sampling time explicitly and not inherit it. The source signal may otherwise depend on the FMI import tool and solver settings.

[b]Not supported by the S-function *CodeFormat* which the the FMU target is derived from.

Table 5.39 User-Defined Functions blocks.

simulink/User-Defined Functions/Fcn	
simulink/User-Defined Functions/MATLAB Fcn (renamed in 2011a, see *Interpreted MATLAB Function*)	Not supported. Not yet supported by Real-Time Workshop/Simulink Coder.
simulink/User-Defined Functions/Interpreted MATLAB Function (new since 2011a)	Not supported. Not yet supported by Real-Time Workshop.
simulink/User-Defined Functions/Embedded MATLAB Function (renamed in 2011a, see *MATLAB Function*)	
simulink/User-Defined Functions/MATLAB Function (new since 2011a)	
simulink/User-Defined Functions/S-Function	
simulink/User-Defined Functions/Level-2 MATLAB S-Function	
simulink/User-Defined Functions/S-Function Builder	Supported if TLC file is generated.
simulink/User-Defined Functions/Matlab System	
simulink/User-Defined Functions/Argument Inport	Not supported. Not supported by the S-Function *CodeFormat* which the Model Exchange targets are based on. The Co-Simulation targets only support reusable code.
simulink/User-Defined Functions/Argument Outport	Not supported. Not supported by the S-Function *CodeFormat* which the Model Exchange targets are based on. The Co-Simulation targets only support reusable code.
simulink/User-Defined Functions/Function Caller	Not supported. Not supported by the S-Function *CodeFormat* which the Model Exchange targets are based on. The Co-Simulation targets only support reusable code.
simulink/User-Defined Functions/Simulink Function	Not supported. Not supported by the S-Function *CodeFormat* which the Model Exchange targets are based on. The Co-Simulation targets only support reusable code.
simulink/User-Defined Functions/Event listener	Not supported for the Model Exchange targets (not supported by the S-Function *CodeFormat*).
simulink/User-Defined Functions/Initialize Function	Not supported for the Model Exchange targets (not supported by the S-Function *CodeFormat*).
simulink/User-Defined Functions/Terminate Function	Not supported. Works in theory for the Co-Simulation targets but need to be used with ert.tlc target to be useful.
simulink/User-Defined Functions/Reset Function	Not supported.

Table 5.40 Additional Math & Discrete/Additional Discrete blocks.

simulink/Additional Math & Discrete/Additional Discrete/Transfer Fcn Direct Form II	Use discrete sample time.
simulink/Additional Math & Discrete/Additional Discrete/Transfer Fcn Direct Form II Time Varying	Use discrete sample time.
simulink/Additional Math & Discrete/Additional Discrete/Fixed-Point State-Space	Use discrete sample time.
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay External IC	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay Resettable	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay Resettable External IC	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay Enabled	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay Enabled Resettable	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay Enabled External IC	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay Enabled Resettable External IC	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay With Preview Resettable	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay With Preview Resettable External RV	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay With Preview Enabled	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay With Preview Enabled Resettable	
simulink/Additional Math & Discrete/Additional Discrete/Unit Delay With Preview Enabled Resettable External RV	

Table 5.41 Additional Math & Discrete/Additional Math blocks.

simulink/Additional Math & Discrete/Additional Math: Increment - Decrement/Increment Real World	

simulink/Additional Math & Discrete/Additional Math: Increment - Decrement/Decrement Real World	
simulink/Additional Math & Discrete/Additional Math: Increment - Decrement/Increment Stored Integer	
simulink/Additional Math & Discrete/Additional Math: Increment - Decrement/Decrement Stored Integer	
simulink/Additional Math & Discrete/Additional Math: Increment - Decrement/Decrement To Zero	
simulink/Additional Math & Discrete/Additional Math: Increment - Decrement/Decrement Time To Zero	

Table 5.42 Control System Toolbox blocks.

cstblocks/LTI System	

5.11. Examples

5.11.1. Using a Simulink model to control a Vehicle model

This example demonstrates how a Simulink controller model for a controlled AWD application can be used in a vehicle model to control the AWD actuator´s torque capacity. This is done in two steps, first by using the FMU target to export the Simulink model and then import it the resulting FMU in an FMI compliant tool that simulates the vehicle model, in this case Dymola 2013 FD01 with Vehicle Dynamics Library. Note that in order to simulate the vehicle model in Dymola, a valid license for VDL is required.

The list of files used in this example are found in Table 5.43.

Table 5.43 Example files.

<installationfolder>/examples/me1/win32/AWDControllerFMITC.mdl	Simulink controller model for a Hang-On to rear AWD.
<installationfolder>/examples/me1/win32/AWDControllerFMITC.fmu	FMU file generated from the Simulink controller. The example demonstrates how this file is generated.
<installationfolder>/examples/me1/win32/ControlledAWD.mo	A Modelica vehicle model that is simulated in Dymola(2013 FD01) using the Simulink controller.

5.11.1.1. Export Simulink model as FMU

1. **Copy example files**

   Copy the example files to a folder with write access, i.e. *C:\Users\<username>\Documents\awdexample*. The Simulink model may not open properly otherwise.

2. **Configure mex compiler**

   Configure the mex compiler in order to help Simulik Coder/Real-Time Workshop selecting an appropriate compiler, see Section 5.4 for more information.

3. **Open Simulink control model**

   Open *AWDControllerFMITC.mdl* in Simulink.

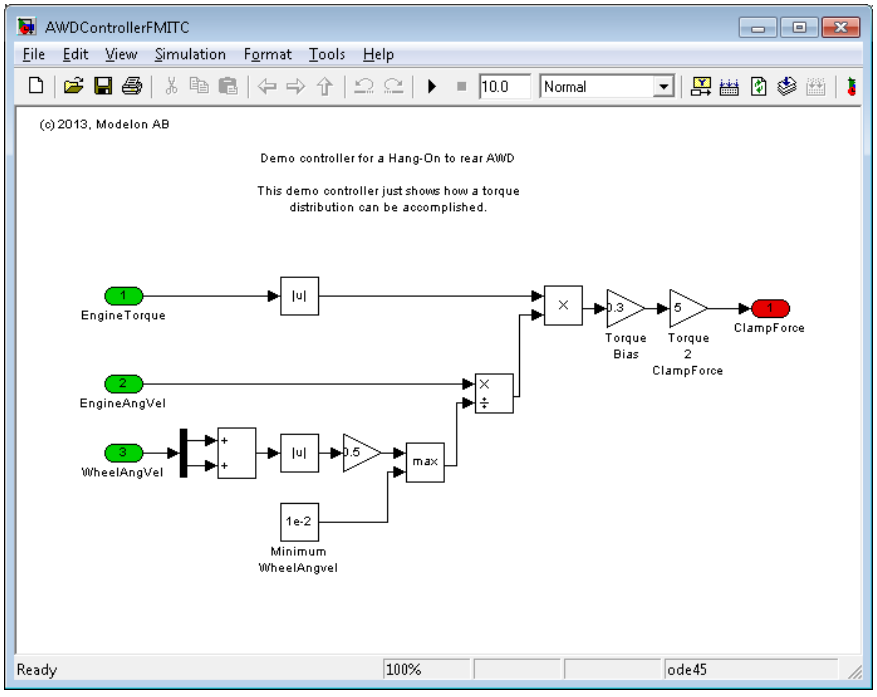


Figure 5.17 Simulink AWD controller model.

4. **Go to the Real-Time Workshop/Coder Generation**

   Open the **Configuration Parameters** dialog and go to the **Real-Time Workshop**/**Coder Generation** (name depends on MATLAB version).

5. **Select target**

   Select **System target file** by clicking on the **Browse...** button. Select **fmu_me1.tlc** in the dialog that opens and then click OK.

6. Build target

Click **Apply** in lower right corner of the **Configuration Parameters** dialog and then press the **Build** button in the **Real-Time Workshop**/**Code Generation** tab. When the build process has finished the *AWDControllerFMITC.fmu* will be located in the current directory.

5.11.1.2. Import FMU in vehicle model and simulate it in Dymola

1. **Open vehicle model**

   Open *ControlledAWD.mo* in Dymola 2013 FD01 and select the *AcceleratingWhileCornering* component, see Figure 5.18.

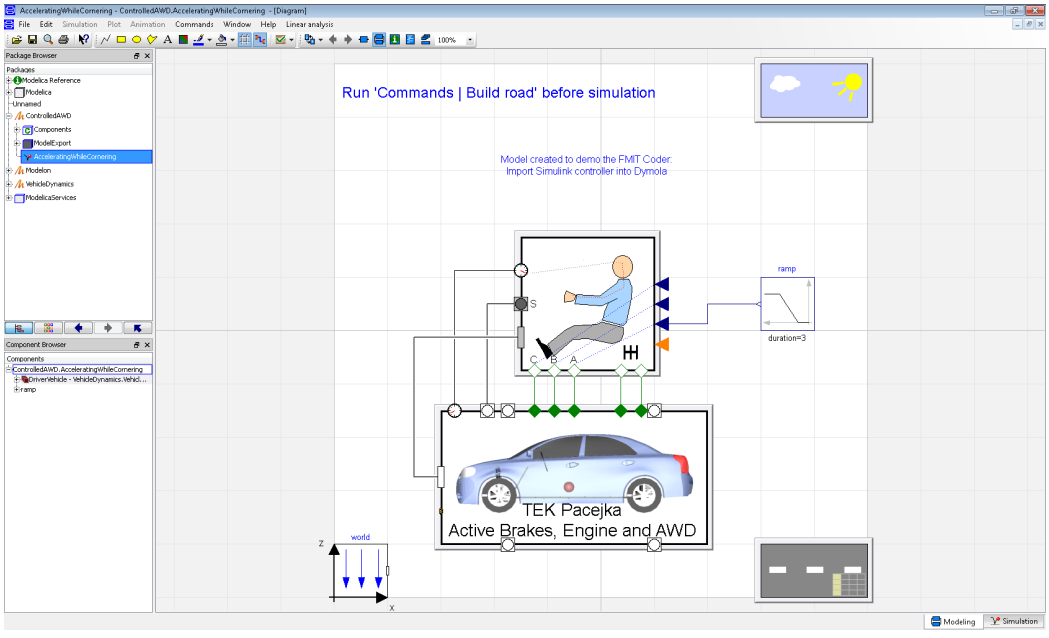


Figure 5.18 Vehicle model ControlledAWD.mo open in Dymola.

2. **Remove placeholder for the FMU controller component**

   Remove the FMU controller component, *ControlledAWD.Components.Controllers.AWDControllerFMITC_fmu*. This is just a placeholder for the FMU that was generated from the Simulink.

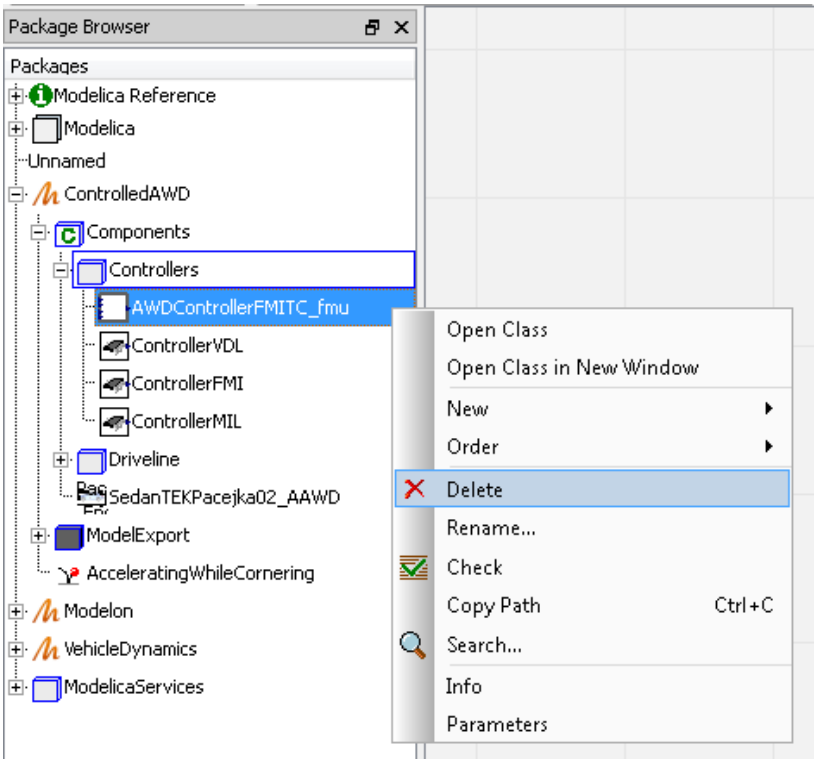


Figure 5.19 Remove the vehicle model's placeholder for the FMU controller.

3. **Import the AWDControllerFMITC FMU**

   In the menu bar, click **File** -> **Import** -> **FMU - All variables...** and select *AWDControllerFMITC.fmu* in the file browser that opens. A new component **AWDControllerFMITC_fmu** is now added in the Package Browser.

4. **Rename the imported FMU**

   Right click on the new **AWDControllerFMITC_fmu** component and select **Rename**. In the **Rename Modelica Class** dialog that opens, select *ControlledAWD.Components.Controllers* in the **Insert in package:** drop down list, see Figure 5.20.

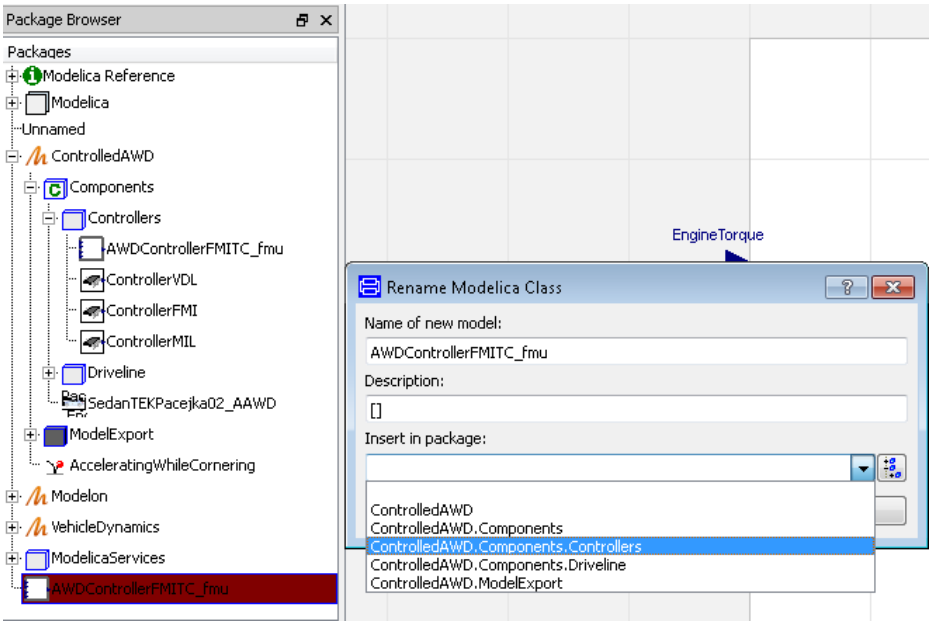


Figure 5.20 Rename AWDControllerFMITC_fmu component.

5. **Build road**

   Select the *AcceleratingWhileCornering* component in the file browser and then in the menu bar, click **Commands** -> **Build road**. Dymola will switch to the **Simulation** tab. Click **Stop** in the dialog that opens and says *Show preview?*.

6. **Select AcceleratingWhileCornering before simulating**

   Go back to the Modeling tab and select the *AcceleratingWhileCornering* to enable the pre configuration simulation settings such as simulation time and tolerances.

7. **Simulate the model**

   Go back to the Simulation tab and click on the Simulation button.

8. **View results and animation**

   Use the results variable browser to view the results or run the animation.

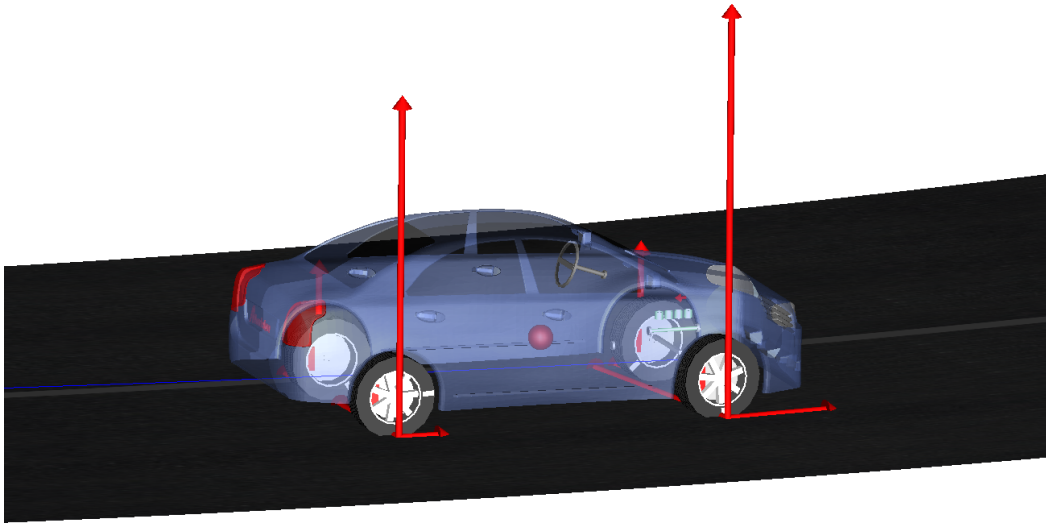


Figure 5.21 Vehicle model simulated using the Simulink controller.

Chapter 6. Design of Experiments

6.1. Introduction

Design of experiments (DoE) is commonly used in the engineering design process t o

- Optimize product design
- Calibrate system
- Verify capability and performance over the entire operating envelope of the process

DoE was originally developed for physical experiments, but is now commonly applied to virtual experiments on detailed simulation models to identify the main relationships between system parameters, operating conditions, and performance.

The DoE tools in the FMI Toolbox for MATLAB support static and dynamic analysis of FMU models in multiple dimensions for Model Exchange 1.0 FMU models. Note that Co-Simulation FMUs are not supported.

The features for dynamic analysis (linearize FMU at each test point, show bode diagram and step responses) require the MATLAB control system toolbox

6.1.1. Concepts

experiment
: Simulation and analysis of FMU at a specified operating/design space point.

factor
: FMU variable that is part of the DoE design, varies between the experiments.

test matrix
: Matrix of operating points where the experiments are run. Each row corresponds to an experiment. Each column corresponds to a DoE factor.

response
: A variable that is influenced by the DoE factors. The response can be an FMU output or state, or some other variable that is computed from the result of the experiments

6.1.2. Workflow

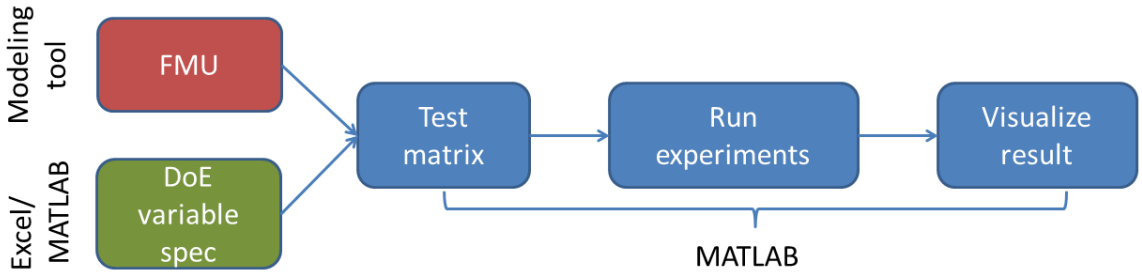


Figure 6.1 FMU DoE analysis workflow

1. Create an FMU model using an FMI compliant tool. Variables to be used as inputs in dynamic analysis must be set as FMU inputs in the FMU. Variables to be stored should be set as FMU outputs.

2. Specify DoE factors in an Excel sheet or in a MATLAB script.

3. Use the MATLAB tools presented in this chapter to create a test matrix and analyze the model at each point in the matrix.

4. Analyze and visualize result.

An important step of the DoE analysis is defining the test matrix. The toolbox supports three types of designs:

- Space-filling quasi-Monte Carlo
- Monte Carlo
- Full factorial (multi-dimensional grid)

Additionally, the tools support a user-supplied test matrix.

6.2. Getting started

In this Getting started tutorial, an analysis of a simulation model will be demonstrated using the DoE tools. The FMU model analyzed is generated from Modelica Standard Library and exported with Dymola as an FMU ME 1.0.

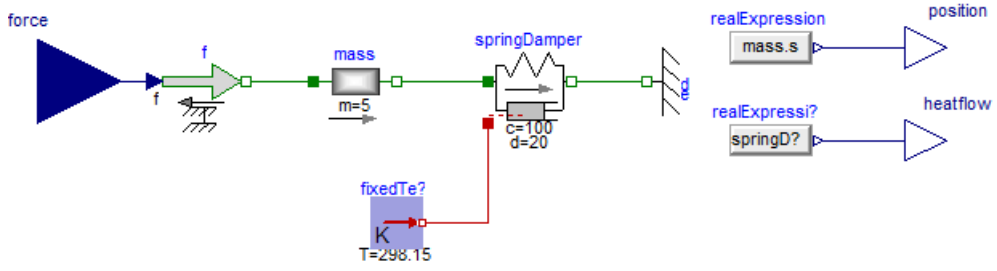


Figure 6.2 MassForce.mo example model in Dymola

The model consists of a simple mechanical system: a force is applied to a body that is connected to a fixed wall through a spring-damper system. The system has one input: the force acting on the body, and two outputs: the position of the body and the heat dissipated in the damper.

Three factors are considered in the DoE analysis:

- force
- spring constant
- mass of body

The DoE factors are defined in an Excel sheet (a template is provided with the toolbox, `DoEParameters.xltx`):

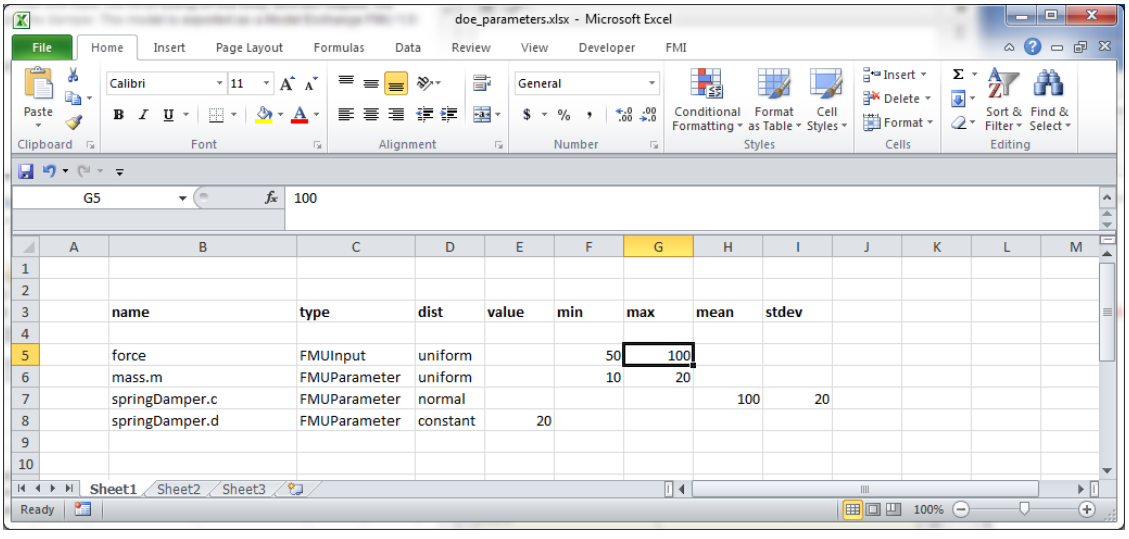


Figure 6.3 DoE parameter definition in Excel

The information on the model and experiment setup is used to create a MATLAB `FMUDoESetup` object:

```
>> doe_setup = FMUDoESetup('MassForce.fmu','doe_parameters.xlsx')
doe_setup =

FMUDoESetup object

    properties:
        fmu_file_name: MassForce.fmu
            exp_setup: cell array with 4 variables specified
              options: struct with 6 fields

    methods: qmc, mc, fullfact, custom
```

The DoE experiments are run by calling a method of the `FMUDoESetup` object. To run a Monte-Carlo analysis that sample each of the DoE factors from the distributions specified in the Excel file, the `mc` method is called:

```
>> nbr_of_experiments = 100;
>> doe_result = doe_setup.mc(nbr_of_experiments);
```

The `FMUDoESetup` method `mc` generates the test matrix and runs the model at each experiment point to find steady state and linearization. An `FMUDoEResult` object is returned:

```
>> doe_result =

FMUDoEResult object

    properties:
        experiment_status: 100 out of 100 experiments successful
          generation_date: 20-Jun-2013 09:53:15
               model_data: struct with 4 fields
                      doe: struct with 6 fields
                constants: struct with 2 fields
             steady_state: struct with 3 fields
                  initial: ---
                   linsys: struct with 5 fields
                  options: struct with 6 fields
                comp_time: 0.54701 s per exp on average

    methods: main_effects, bode, step
```

The steady-state values of model inputs, outputs, and states are stored in `doe_result.steady_state`

```
>> doe_result.steady_state
ans =
    u: [100x1 double]
    y: [100x2 double]
    x: [100x2 double]
```

The method `main_effects` plots a variable against all DoE factors. The mass position (first FMU output) is plotted as

```
>> position = doe_result.steady_state.y(:,1);
>> doe_result.main_effects(position);
```

equivalently, the `main_effects` method may be called with a string that corresponds to the name of an FMU input, output, or DoE factor

```
>> doe_result.main_effects('position');
```

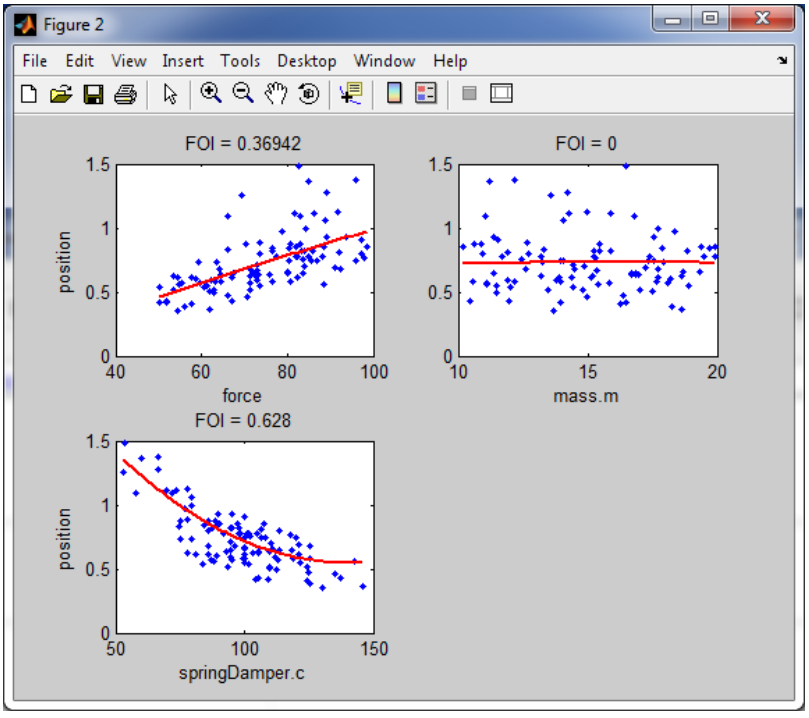


Figure 6.4 Plot of steady-state mass position vs. DoE factors

6.3. Function reference

The DoE tools operate on MATLAB objects of three classes:

FMUModelME1
: FMU model object, as described in Chapter 4.

  Methods:

  - trim
  - linearize

FMUDoESetup
: Stores info on the DoE factors and their distributions, the FMU model, and simulation options.

  Methods:

  - qmc
  - mc
  - fullfact
  - custom

FMUDoEResult
: Stores result from a set of experiments, including input, output, state, and linearization at all test points.

  Methods:

  - main_effects
  - bode (requires Control System Toolbox)
  - step (requires Control System Toolbox)

6.3.1. FMUModelME1

The FMUModelME1 class has several methods and properties, as described in Chapter 4. Methods that are relevant for the DoE features are described here.

6.3.1.1. trim

Find steady-state solution of FMU model

```
[X_SS,U_SS,Y_SS] = fmu_model.trim()
```

Find steady state values of state `X_SS`, input `U_SS`, and output `Y_SS` for FMU model `fmu_model`. The model should be loaded and instantiated before calling `trim`.

```
[X_SS,U_SS,Y_SS] = fmu_model.trim(U)
```

Find steady state for input `U`.

```
[X_SS,U_SS,Y_SS] = fmu_model.trim(U,X_GUESS)
```

Use `X_GUESS` as initial guess for `X_SS`.

```
[X_SS,U_SS,Y_SS] = fmu_model.trim(U,X_GUESS,Y,U_GUESS)
```

Find steady state for a specified value of the output `Y`. The input that matches the output `Y` will be found iteratively. If an output `Y` is specified, the corresponding free input should be set to `NaN`. `U_GUESS` is used as initial guess for `U_SS`.

Example:

For a two-input-two-output system, find the steady-state solution where `u(1) = 2` and `y(2) = 5`. The second input `u(2)` is free to vary to achieve `y(2) = 5`. The arguments should be chosen as `U = [2 NaN]'; Y = [NaN 5]';`

```
[X_SS,U_SS,Y_SS] = fmu_model.trim(U,X_GUESS,Y,U_GUESS,U_MIN,U_MAX)
```

Constrains the solution `U_SS` to lie within `U_MIN` and `U_MAX`.

```
[X_SS,U_SS,Y_SS] = fmu_model.trim(U,X_GUESS,Y,U_GUESS,U_MIN,U_MAX,OPTIONS)
```

Uses a struct `OPTIONS` for simulation options. Use the trimset function to get an `OPTIONS` struct with the default values that can be modified and sent to trim:

```
>> OPTIONS = trimset();
>> OPTIONS.MaxIter = 100; % Default is 50, see 'help trimset'
>> [X_SS,U_SS,Y_SS] = fmu_model.trim(U,X_GUESS,Y,U_GUESS,U_MIN,U_MAX,OPTIONS);
```

```
[X_SS,U_SS,Y_SS,EXITFLAG] = fmu_model.trim(U,X_GUESS,Y,U_GUESS,U_MIN,U_MAX)
```

`EXITFLAG` indicates the success of the iterative algorithm to find input for a given output.

- `EXITFLAG=0` --- Success
- `EXITFLAG=1` --- Could not find inputs that match the specified outputs due to input saturation or local minimum. The solution at the last iteration is returned.
- `EXITFLAG=2` --- Maximum number of iterations reached. The solution at the last iteration is returned.

6.3.1.2. linearize

Linearization of FMU model using finite differences

```
[A,B,C,D,YLIN] = fmu_model.linearize()
```

```
[A,B,C,D,YLIN] = fmu_model.linearize(XLIN,ULIN)
```

Computes linearization of the FMU model object `fmu_model`. The model should be loaded and instantiated before calling `linearize`. The model is linearized with the state `XLIN` and the input `ULIN` or with the current state and input of the FMU if `XLIN` and `ULIN` are not given. Note that linearization is normally done at steady-state. If `XLIN` and `ULIN` do not correspond to a stationary state of the system, unexpected results may be obtained. Returns `A,B,C,D`: the system matrices for the linearized system from all FMU inputs to all FMU outputs, and `YLIN`: the system output at the linearization point.

6.3.2. FMUDoESetup

The FMUDoESetup class is used to store information on the virtual experiment setup. It stores the name of the FMU file, info on parameter ranges and distributions, and general simulation options.

6.3.2.1. Constructor

Constructor

```
DOE_SETUP = FMUDoESetup(FMU_FILE_NAME,EXP_SETUP_FILE)
```

Define the DoE experiment setup. `FMU_FILE_NAME` is the name of the FMU file. `EXP_SETUP_FILE` is the name of an Excel spreadsheet that contains the distribution specification on the DoE factors. A template for the Excel file is provided with the FMI Toolbox, `DoESetup.xlsx`.

The first non-empty row of the Excel sheet should contain column titles *name*, *type*, *dist*, etc. The FMU variables are listed below the column title row. Empty rows and columns are discarded.

Required columns:

name	Name of FMU variable
type	One of *FMUInput*, *FMUOutput*, *FMUParameter*
dist	One of the supported distributions: *constant*, *uniform*, *normal*, *triangle*, *free*

Depending on the choice of *dist*, additional columns are required:

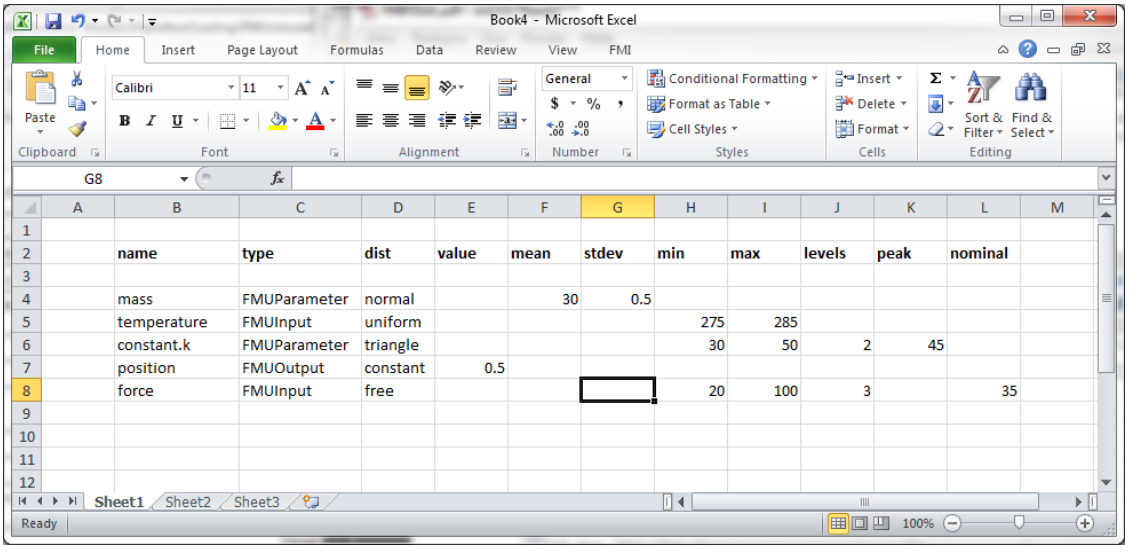
dist	required columns	optional columns
constant	value	
uniform	min, max	
normal	mean, stdev	
triangle	min, max	peak
free	nominal	min, max

The option *dist=free* is only available for FMU inputs, and is used when one or more FMU outputs are specified. The input will be chosen iteratively to match the specified output. The value is constrained to lie in the range specified by the *min* and *max* columns, and the value in column *nominal* is used as initial guess.

Additional columns that specify data used for certain DoE designs can also be added:

column	description
levels	Used for gridding DoE designs to specify the number of grid levels for each variable

Note that each of the DoE methods (`qmc`, `fullfact` etc.) use different algorithms to design a test matrix from the distribution information. All column data are not used for all types of DoE designs. See the documentation on these methods below.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2		**name**	**type**	**dist**	**value**	**mean**	**stdev**	**min**	**max**	**levels**	**peak**	**nominal**	
3													
4		mass	FMUParameter	normal		30	0.5						
5		temperature	FMUInput	uniform				275	285				
6		constant.k	FMUParameter	triangle				30	50	2	45		
7		position	FMUOutput	constant	0.5								
8		force	FMUInput	free				20	100	3		35	
9													
10													
11													
12													

Figure 6.5 Example of experiment setup Excel sheet.

```
DOE_SETUP = FMUDoESetup(FMU_FILE_NAME,EXP_SETUP_FILE,EXP_SETUP_SHEET)
```

`EXP_SETUP_SHEET` is either the name or the index of an Excel sheet in the file `EXP_SETUP_FILE` from which the parameter specification is taken.

```
DOE_SETUP = FMUDoESetup(FMU_FILE_NAME,EXP_SETUP_ARRAY)
```

Alternatively, the experiment setup can be specified as a cell array in MATLAB. The data in the example Excel file above can equivalently be entered as

```
exp_setup = cell(5,1);
```

```
exp_setup{1}.name = 'mass';
exp_setup{1}.type = 'FMUParameter';
exp_setup{1}.dist = 'normal';
exp_setup{1}.mean = 30;
exp_setup{1}.stdev = 0.5;

exp_setup{2}.name = 'temperature';
exp_setup{2}.type = 'FMUInput';
exp_setup{2}.dist = 'uniform';
exp_setup{2}.min = 275;
exp_setup{2}.max = 285;

exp_setup{3}.name = 'constant.k';
exp_setup{3}.type = 'FMUInput';
exp_setup{3}.dist = 'triangle';
exp_setup{3}.min = 30;
exp_setup{3}.max = 50;
exp_setup{3}.peak = 45;

exp_setup{4}.name = 'position';
exp_setup{4}.type = 'FMUOutput';
exp_setup{4}.dist = 'constant';
exp_setup{4}.value = 0.5;

exp_setup{5}.name = 'force';
exp_setup{5}.type = 'FMUInput';
exp_setup{5}.dist = 'free';
exp_setup{5}.min = 20;
exp_setup{5}.max = 100;
exp_setup{5}.peak = 35;

doe_setup = FMUDoESetup('model.fmu',exp_setup);
```

```
FMUDoESetup(FMU_FILE_NAME,EXP_SETUP_FILE,EXP_SETUP_SHEET,OPTIONS)
```

Sets the `FMUDoESetup` property `options` to the `OPTIONS` struct. The default struct can be accessed through the function `fmu_doe_options`.

```
OPTIONS = fmu_doe_options()
```

The fields in the struct `OPTIONS` may then be modified by the user before using it in the `FMUDoESetup` constructor.

field	allowed values (default first)	description
`mode`	`steady_state`	perform DoE analysis at system steady-state
	`initial`	perform DoE analysis after initialization

field	allowed values (default first)	description
`Tmax`	positive scalar (default `1e6`)	maximum simulation horizon before reaching steady-state
`linearize`	`on`	compute linearization at all test points (only available if MATLAB Control System Toolbox is installed)
	`off`	do not compute linearization (automatically selected if the MATLAB Control System Toolbox is not available)
`minreal`	`on`	return minimal realization of linearized system between selected inputs and outputs
	`off`	return full linearized system
`input_index`	`all`	linearization computed from all FMU inputs
	index vector	linearization only computed from input indices specified in vector
`output_index`	`all`	linearization computed to all FMU outputs
	index vector	linearization only computed to output indices specified in vector
`solver`	`ode15s`	
	Name of MATLAB ode solver	
`solver_settings`	solver settings struct	see MATLAB command `odeset`

6.3.2.2. DoE methods

Four DoE designs are supported

`qmc`	Space-filling quasi-Monte-Carlo design in a hypercube
`mc`	Monte-Carlo sampling
`fullfact`	Full-factorial multi-dimensional grid
`custom`	User-defined test matrix

All DoE methods perform the same analysis steps:

- Generate a test matrix according to the experiment setup specification
- Simulate the FMU at all points in the test matrix to extract the state input and output, either at steady-state (default) or after initialization depending on the `mode` setting in the options struct
- Linearize the system at each point in the test matrix
- Return an `FMUDoEResult` object (see next section) with the analysis result

The difference between the functions is how the test matrix is computed.

To illustrate the difference, consider a DoE run with two factors, *x1*and *x2*. Both factors are specified to have distribution *normal*, *mean = 0*, and *stdev = 1*. Examples of test matrices that can be constructed by the four functions are shown below:

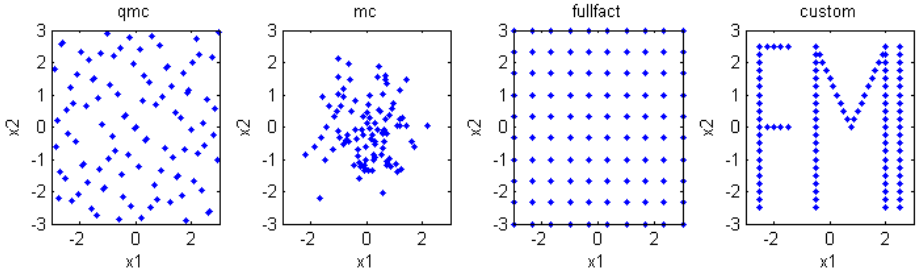


Figure 6.6 Example of 2D test matrices

The choice of design should be based on the type of questions to be answered by the analysis. The space-filling QMC design distributes the points in the test matrix evenly in a hypercube. This is useful for investigating the achievable capacity of the system over a design space, e.g., for process optimization. It is also applicable to find the worst-case scenario over a range of process operating conditions.

For statistical analysis, e.g., determine the distribution of some performance value given statistical distributions on component parameters, the Monte Carlo design should be used.

Full factorial design can be used as an alternative to the QMC design if the number of DoE factors are small or if a grid design is specifically requested. In general, the QMC design is more efficient for investigating the system in a hypercube.

qmc - Quasi-Monte Carlo analysis

Space-filling quasi-Monte-Carlo DoE design.

```
DOE_RESULT = doe_setup.qmc(NBR_OF_EXPERIMENTS)
```

The test matrix is generated using a Sobol sequence, which is a quasi-random sequence that aims to distribute the test points evenly in a hypercube. For DoE factors with uniform or triangular distribution, the edges of the

hypercube are taken from the `min` and `max` values. For DoE variables with a normal distribution, the hypercube edges are taken at *mean* ± *3*stdev*. The number of experiments are given by `NBR_OF_EXPERIMENTS`. Note that `qmc` does *not* sample according to the specified distribution. Rather, it is a tool to explore the model within a specified parameter space.

mc - Monte Carlo analysis

Monte-Carlo DoE design.

```
DOE_RESULT = doe_setup.mc(NBR_OF_EXPERIMENTS)
```

The test matrix is generated by sampling independently from the distributions that are specified for each DoE factor. The number of experiments are given by `NBR_OF_EXPERIMENTS`.

fullfact - Full factorial analysis

Full factorial DoE design.

```
DOE_RESULT = doe_setup.fullfact()
```

The test matrix is generated as a multi-dimensional grid. For DoE factors with uniform or triangular distribution, the edges of the grid are taken from the `min` and `max` values. For DoE variables with a normal distribution, the grid edges are taken at *mean* ± *3*stdev*. The number of levels for each factor is taken from the `levels` field. For factors where no value is given for levels, three levels are used as default.

```
DOE_RESULT = doe_setup.fullfact(DEFAULT_LEVELS)
```

Changes the default number of levels to `DEFAULT_LEVELS`. The value in `DEFAULT_LEVELS` is overridden for the factors where the field `levels` is specified in the experiment setup.

custom - User-defined test matrix

DoE analysis with user-provided test matrix.

```
DOE_RESULT = doe_setup.custom(TEST_MATRIX,VARIABLE_NAMES,VARIABLE_TYPES)
```

Runs the DoE experiments at operating points in a user-provided test matrix. Each row in `TEST_MATRIX` corresponds to an experiment, and each column to a variable. The names of the variables corresponding to the test matrix columns are supplied in the cell array `VARIABLE_NAMES`, and their types (*FMUInput*, *FMUParameter*, *FMUOutput*) are supplied in the cell array `VARIABLE_TYPES`. Variable specifications that are provided in the `doe_setup` object are applied if their `dist` value is either `constant` or `free`, otherwise they are ignored. .

Example

```
test_matrix = [1 0 5
               1 1 5
               0 1 5
               0 0 5];
var_names = {'u1','u2','some_parameter'};
var_types = {'FMUInput','FMUInput','FMUParameter'};
result = doe_setup.custom(test_matrix, var_names,var_types);
```

6.3.3. FMUDoEResult

The DoE methods return an `FMUDoEResult` object. The information from the experiment can be accessed as properties on the object. Three methods are provided to visualize the result:

main_effects
: Plots a response variable against each of the DoE factors

bode
: Shows the Bode diagram of the ensemble of linear systems at all test points

step
: Shows the step response of the ensemble of linear systems at all test points

6.3.3.1. properties

field	subfields	description
`generation_date`		time and date when the `FMUDoEResult` object was generated
`model_data`	`name`	name of FMU
	`input_names`	names of all FMU inputs
	`output_names`	names of all FMU outputs
	`generation_date`	time and date when the FMU was generated
`doe`	`nbr_of_experiments`	number of experiments in DoE design
	`ndim`	number of DoE factors
	`factor_names`	names of DoE factors
	`factor_types`	types of DoE factors (FMUInput, FMUOutput, FMUParameter)
	`test_matrix`	test matrix
	`generating_function`	the name of the `FMUDoESetup` method that generated the result
`constants`	`names`	names of all variables that were constant in all the experiments, but changed from the default FMU values

field	subfields	description
	values	values for these constants
experiment_status		vector of size nbr_of_experiments x 1 that denote the status of the FMU analysis at each test point
		0 | Successful
		-1 | Could not find inputs to match specified outputs because of input saturation or a local minimum. If input was saturated, return values for steady-state solutions and linearized model are taken slightly away from the saturation border. If a local minimum was found, return values are taken at the last iteration point.
		-2 | The algorithm to find inputs to match specified outputs did not converge. Return values correspond to last iteration point.
		-99 | Error in simulating the FMU with the given settings, no values are returned at this test point.
steady_state	u	steady-state input u
	y	steady-state output y
	x	steady-state state x
linsys	sys	cell array where each item corresponds to the system linearization at one of the test points
	u_index	the indices of FMU inputs that are inputs to the linear systems (the default is all inputs, see Section 6.3.2.1)
	y_index	the indices of FMU outputs that are outputs to the linear systems
	u_names	the names of the linear system inputs
	y_names	the names of the linear system outputs
options		the options struct that were used when generating the result struct
comp_time		vector of size nbr_of_experiments x 1 with the computational time in seconds for each experiment

6.3.3.2. main_effects

Visualization of the main effects of the DoE factors on a response variable.

```
[FOI,TS] = doe_result.main_effects(RESPONSE)
```

RESPONSE is either a vector of length doe_result.doe.nbr_of_experiments, or the name of an FMU input or output variable. The function generates a series of subplots where the RESPONSE variable is plotted against each of the DoE factors. A second order polynomial is fitted to the data and also shown in the plots.

FOI is a vector of first-order indices: the ratio of variance explained by the second-order polynomial approximation to the total variance in the response. TS (total score) is the ratio of variance described by a second-order polynomial fit in all of the variables (without interaction terms), to the total variance in the response.

```
[FOI,TS] = doe_result.main_effects(RESPONSE,RESPONSE_LABEL)
```

Uses the string RESPONSE_LABEL for the y axis label.

```
[FOI,TS] = doe_result.main_effects(RESPONSE,RESPONSE_LABEL,CLASSES)
```

Color codes the dots according to the array CLASSES. CLASSES should be a vector of integers or an array of strings, each unique element in CLASSES will be assigned to a different color.

```
[FOI,TS] = doe_result.main_effects(RESPONSE,RESPONSE_LABEL,CLASSES,FIG_NR)
```

Generates the plot in figure FIG_NR.

Caution:

If the factors are not independent (e.g., if a custom test matrix was used or the number of experiments is small compared to the number of factors), false correlations between factors and response may appear. The first-order indices FOI and total score TS values should be interpreted with care.

Example:

A vector as response variable

```
y1 = result.steady_state.y(:,1);
y2 = result.steady_state.y(:,2);
doe_result.main_effects(y2-y1);
```

An FMU output variable name as response variable

```
doe_result.main_effects('y2');
```

Using a CLASSES vector

```
indicator_array = cell(doe_result.doe.nbr_of_experiments,1);
u2 = doe_result.steady_state.u(:,2);
for k=1:1:doe_result.doe.nbr_of_experiments
    if u2 < 2
```

```
        indicator_array{k} = 'low';
    elseif u2 > 5
        indicator_array{k} = 'high';
    else
        indicator_array{k} = 'medium';
    end
end
y1 = doe_result.steady_state.y(:,1);
doe_result.main_effects(y1,'first output',indicator_array);
```

6.3.3.3. bode

Visualize Bode plot variability for the system linearizations computed at the test points.

```
PLOT_SUBSET = doe_result.bode()
```

Computes the magnitude and phase for each of the linear systems in `doe_result.linsys.sys`. To generate the plot efficiently, only a subset of the systems are shown in the plot. By default, the systems with the highest and lowest phase and magnitude at each frequency are determined. The union of these systems over all frequencies are then plotted. This means that the systems that are excluded have a frequency response that lie within the range of the ones that are shown. For MIMO systems, the selection of systems to show are done individually for each pair of inputs and outputs. See the `PLOTMODE` argument below for alternative methods to select a subset of systems to show. Returns the subset of systems that are shown in the plot in the cell array `PLOT_SUBSET`.

```
PLOT_SUBSET = doe_result.bode(INPUT_INDEX,OUTPUT_INDEX)
```

For MIMO systems, only shows the Bode plot for the systems between the inputs and outputs specified in `INPUT_INDEX` and `OUTPUT_INDEX`.

```
PLOT_SUBSET = doe_result.bode(INPUT_INDEX,OUTPUT_INDEX,W_RANGE)
```

Selects the subset of systems and shows the Bode plot in the frequency range specified by `W_RANGE = {w_min,w_max}`.

```
PLOT_SUBSET = doe_result.bode(INPUT_INDEX,OUTPUT_INDEX,W_RANGE,PLOTMODE)
```

`PLOTMODE` determines the subset of systems that are shown. The default, where all systems that span the envelope of the magnitude and phase are shown, corresponds to `PLOTMODE = 'envelope'`. `PLOTMODE = 'all'` shows all systems (max number of systems is 100). `PLOTMODE = NBR` where `NBR` is a positive integer selects `NBR` systems at random. For this option the same selection is applied to all input-output pairs.

```
PLOT_SUBSET = doe_result.bode(INPUT_INDEX,OUTPUT_INDEX,W_RANGE,PLOTMODE,FIG_NR)
```

Generates the plot in figure `FIG_NR`.

6.3.3.4. step

Visualize step response variability for the system linearizations computed at the test points.

```
PLOT_SUBSET = doe_result.step()
```

Computes the step response for each of the linear systems in `doe_result.linsys.sys`. To generate the plot efficiently, only a subset of the systems are shown in the plot. By default, the systems that span the envelope of the step response trajectories are plotted. This means that the systems that are excluded have a step response that lie within the range of the ones that are shown. For MIMO systems, the selection of systems to show are done individually for each pair of inputs and outputs. See the `PLOTMODE` argument below for alternative methods to select a subset of systems to show. Returns the subset of systems that are shown in the plot in the cell array `PLOT_SUBSET`.

```
PLOT_SUBSET = doe_result.step(INPUT_INDEX,OUTPUT_INDEX)
```

For MIMO systems, only shows the step responses between the inputs and outputs specified in `INPUT_INDEX` and `OUTPUT_INDEX`.

```
PLOT_SUBSET = doe_result.step(INPUT_INDEX,OUTPUT_INDEX,T_MAX
```

Selects the subset of systems and shows the step response in the time range `[0,T_MAX]`.

```
PLOT_SUBSET = doe_result.step(INPUT_INDEX,OUTPUT_INDEX,T_MAX,PLOTMODE)
```

`PLOTMODE` determines the subset of systems that are shown. The default, where all systems that span the envelope of the step response trajectories are shown, corresponds to `PLOTMODE = 'envelope'`. `PLOTMODE = 'all'` shows all systems (max number of systems is 100). `PLOTMODE = NBR` where `NBR` is a positive integer selects `NBR` systems at random. For this option the same selection is applied to all input-output pairs.

```
PLOT_SUBSET = doe_result.step(INPUT_INDEX,OUTPUT_INDEX,T_MAX,PLOTMODE,FIG_NR)
```

Generates the plot in figure `FIG_NR`.

6.4. Examples

6.4.1. Mass-Spring system

The mass-spring model was introduced in Section 6.2.

6.4.1.1. Define the Experiment Setup

To load the experiment setup data from the second sheet in the Excel file `doe_parameters.xlsx`, the DoE setup constructor is called with the optional sheet name argument:

```
doe_setup = FMUDoESetup('MassForce.fmu','doe_parameters.xlsx','Sheet2')
```

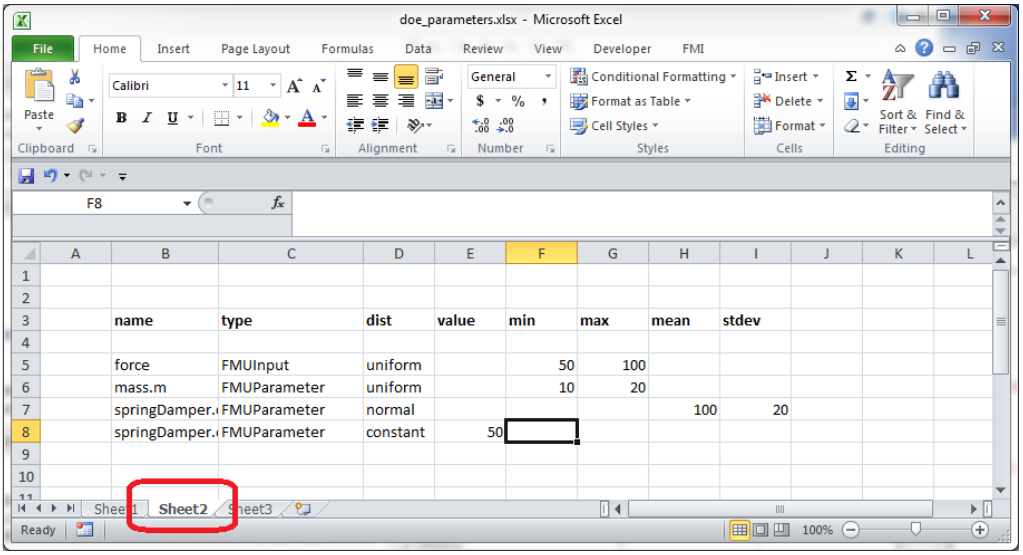


Figure 6.7 DoE parameter definition in Excel

The DoE design spans three dimensions: the force input *force*, the mass parameter *mass.m*, and the spring constant parameter *springDamper.c*. A constant is also defined in the Excel sheet, the damping constant *springDamper.d*. All FMU parameters that are not specified in the Excel sheet will be set to their default values in the FMU.

The `doe_setup` object stores information on the FMU file name and the parameter specification loaded from Excel

```
>> doe_setup
doe_setup =

FMUDoESetup object

    properties:
        fmu_file_name: MassForce.fmu
            exp_setup: cell array with 4 variables specified
              options: struct with 6 fields

    methods: qmc, mc, fullfact, custom
```

6.4.1.2. Run DoE experiments

The `qmc` DoE method is an efficient algorithm to spread test points approximately evenly in a hypercube.

To run the `qmc` method with 100 test points, call

```
>> nbr_of_experiments = 100;
```

```
>> doe_result = doe_setup.qmc(nbr_of_experiments);
```

The test matrix from a DoE run can be accessed in `doe_result.doe.test_matrix`. According to the specification of the `qmc` method, the test points should be distributed evenly between the *min* and *max* values for the parameters with *dist=uniform*, and between *mean+-3*stdev* for variables with *dist=normal*. The following code illustrates the test design in 3D and projected in 2D

```
test_matrix = doe_result.doe.test_matrix;
factor_names = doe_result.doe.factor_names;
%----------------
% 3D-plot
%----------------
subplot(4,1,1); plot3(test_matrix(:,1),test_matrix(:,2),test_matrix(:,3),'o'); grid on;
xlabel(factor_names{1}); ylabel(factor_names{2}); zlabel(factor_names{3});
%----------------
% Projected to 2D
%----------------
subplot(4,1,2); plot(test_matrix(:,1),test_matrix(:,2),'o');
xlabel(factor_names{1}); ylabel(factor_names{2});
subplot(4,1,3); plot(test_matrix(:,2),test_matrix(:,3),'o');
xlabel(factor_names{2}); ylabel(factor_names{3});
subplot(4,1,4); plot(test_matrix(:,3),test_matrix(:,1),'o');
xlabel(factor_names{3}); ylabel(factor_names{1});
```

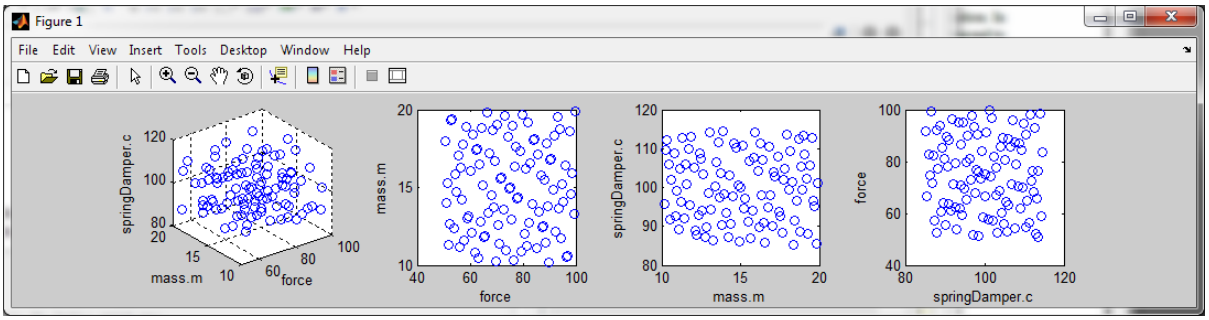


Figure 6.8 DoE test matrix plot

The test points are approximately uniformly distributed in the cube spanned by the three DoE factors.

6.4.1.3. Analyze results

Steady-state

The steady-state values of model inputs, outputs, and states are available in `doe_result.steady_state`

```
>> doe_result.steady_state
ans =
    u: [100x1 double]
    y: [100x2 double]
```

```
    x: [100x2 double]
```

The method `main_effects` plots a response against all DoE factors. The mass position (first FMU output) is plotted as

```
>> position = doe_result.steady_state.y(:,1);
>> doe_result.main_effects(position);
```

equivalently, the `main_effects` method may be called with a string that corresponds to the name of an FMU input, output, or DoE factor

```
>> doe_result.main_effects('position');
```

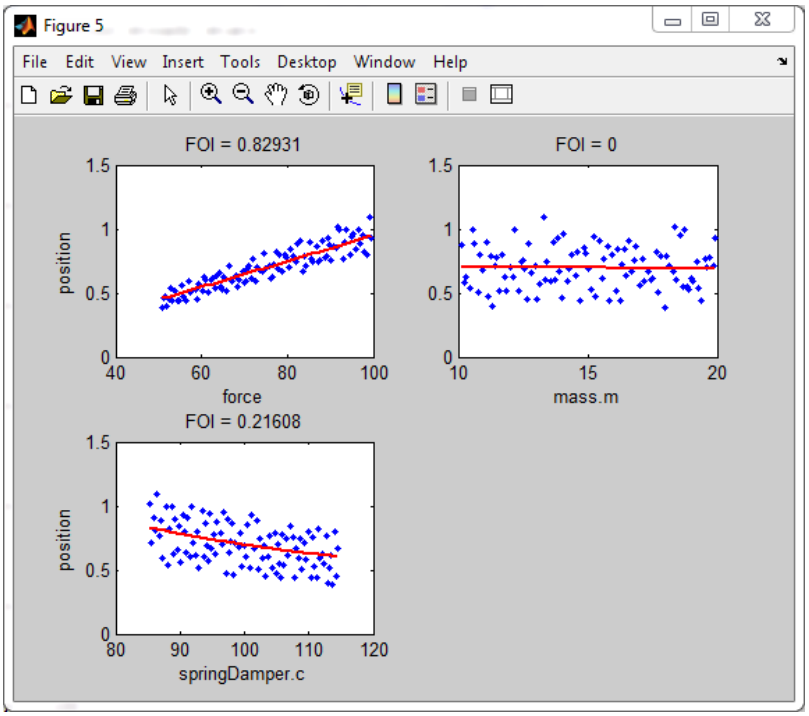


Figure 6.9 Plot of steady-state mass position vs. DoE factors

In the range examined in the DoE design, the main influencing factor for the steady-state mass position is the applied force. The spring constant also influences the steady-state position but to a smaller extent, and the mass has no influence at all.

The value on top of each subplot is the first-order-index; it tells the percentage of the variation in the plotted response variable that is explained by a second-order polynomial in the corresponding DoE factor. Note that the FOI indices may sum to more than 1 if the factors are not perfectly uncorrelated.

The dots in the scatter plots may be color coded by using an integer vector that represent different classes as input to the `main_effects` method. Each unique integer in the vector is assigned to a different color.

```
>> force = doe_result.doe.test_matrix(:,1);
>> indicator_vector = zeros(size(force));
>> indicator_vector(force>90) = 1;
>> indicator_vector(force<60) = -1;
>> doe_result.main_effects('position','mass position',indicator_vector);
```

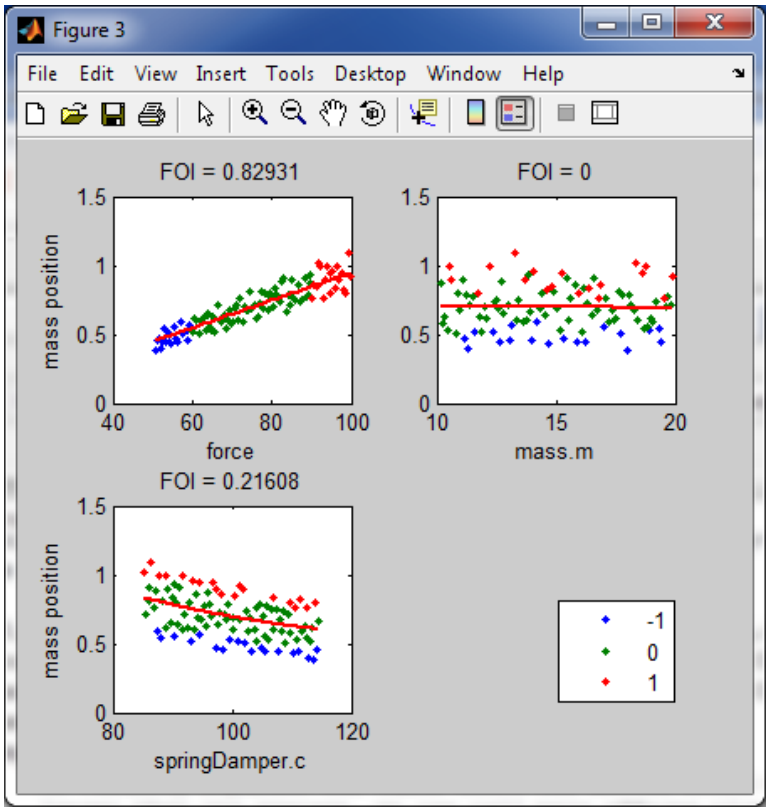


Figure 6.10 Plot of steady-state mass position vs. DoE factors with color coding.

The influence of force (high = red, medium = green, low = blue) is now visible in all DoE factor subplots.

Dynamic analysis

The linearization of the system at the points in the test matrix is available in `doe_result.linsys`

```
>> doe_result.linsys
```

```
ans =
        sys: {100x1 cell}
    u_index: 1
    y_index: [1 2]
    u_names: {'force'}
    y_names: {'position'  'heatflow'}
```

`doe_result.linsys.sys` is a cell array with a state-space model for each point in the test matrix. The method `bode` plots the Bode diagrams for the set of linear systems.

```
>> doe_result.bode(1,1,[],'all');
```

plots the Bode diagram for all linear systems in the `doe_result` struct from the first FMU input (here force) to the first FMU output (here mass position).

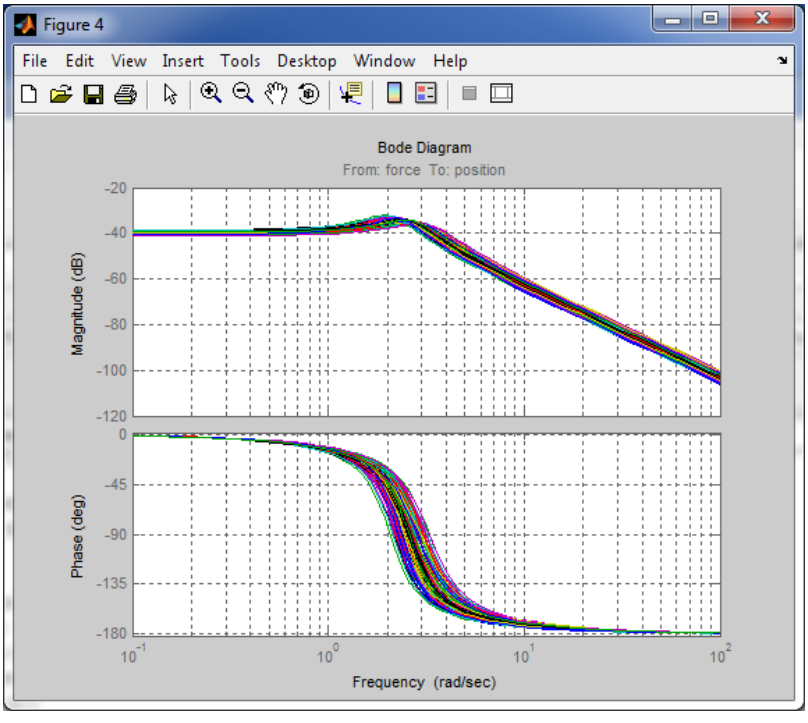


Figure 6.11 Bode diagram for the linearized systems at all test points from input force to mass position.

The corresponding step responses can be plotted using `step`

```
>> doe_result.step(1,1,[],'all');
```

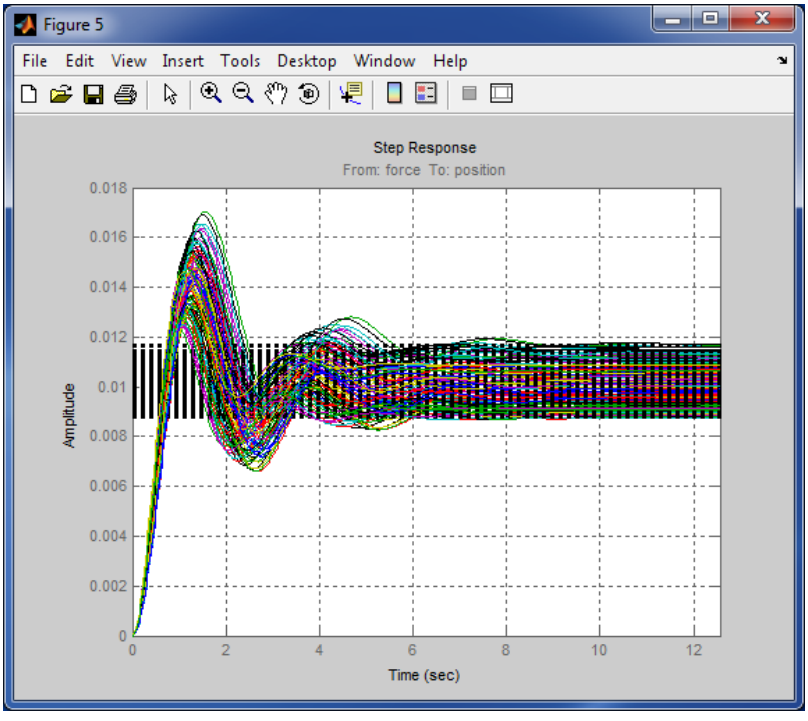


Figure 6.12 Step response for the linearized systems at all test points from input force to mass position.

Steady-state values for each of the step responses are shown as dashed black lines.

If the number of experiments is large, the Bode plots and step responses may be slow to generate and visually cluttered if all systems are shown. The `bode` and `step` methods have an option to plot the envelope of step responses and Bode diagrams. For this case, the curves are computed for all systems but only the ones that have the maximum or minimum value for some time/frequency are shown in the plot. For example,

```
>> doe_result.step(1,1,[],'envelope');
```

generates the plot

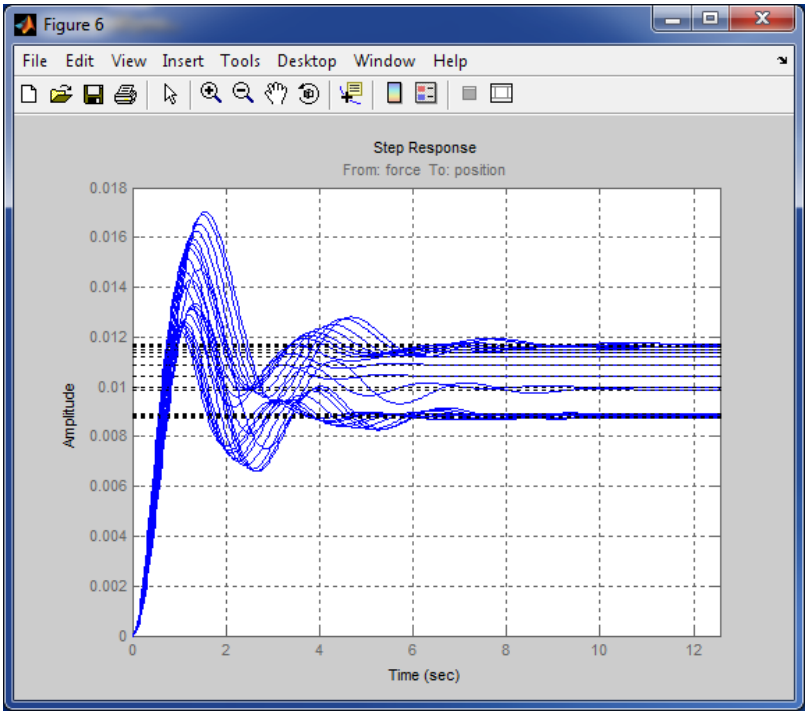


Figure 6.13 Envelope of step responses for the linearized systems.

All step responses that are not shown lie entirely within the ones that are shown in the plot.

With only a few lines of code, it is possible to correlate some feature of the step response or Bode diagram to DoE factors. The following code extracts the peak frequency for each Bode plot and correlates it to the DoE factors in a main effects plot

```
peak_frequency = zeros(doe_result.doe.nbr_of_experiments,1);
freq_vec = logspace(0,1,100);                                %  search for the peak in
                                                             %     the frequency
                                                             %     range [10^0,10^1]
for k=1:1:doe_result.doe.nbr_of_experiments
    [mag,phase] = bode(doe_result.linsys.sys{k}(1,1),freq_vec); % compute mag and phase
                                                             %     at these frequencies
    [maxgain,indmaxfreq] = max(squeeze(mag));                % extract max gain and
                                                             %     corresponding freq_vec
                                                             %     index
    peak_frequency(k) = freq_vec(indmaxfreq);                % store peak frequency
end
doe_result.main_effects(peak_frequency,'peak freq');       % generate plot
```

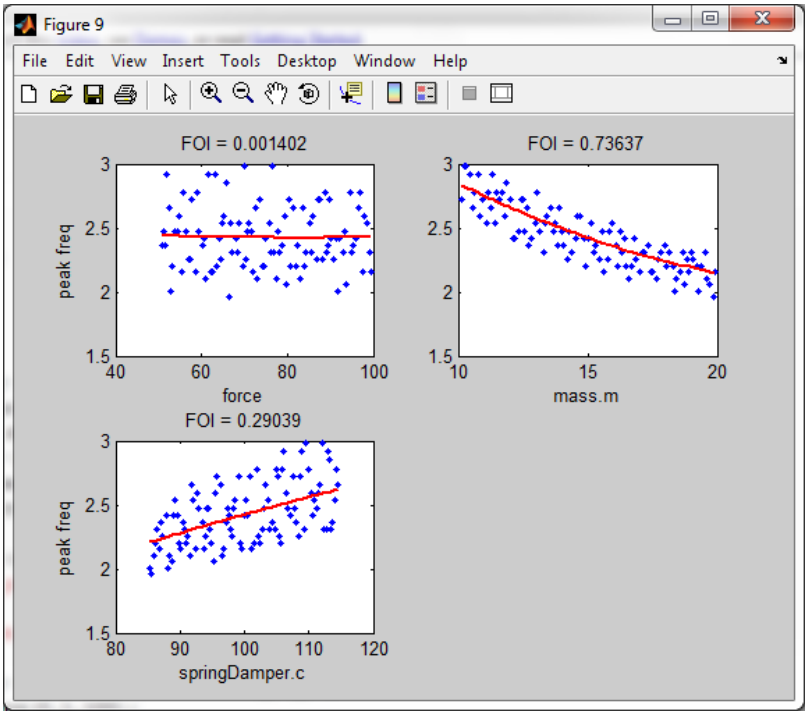


Figure 6.14 Maximum-gain frequency plotted vs. DoE factors.

Chapter 7. Tutorial examples

7.1. Stabilization of a Furuta pendulum system

In this tutorial, you will go through the following steps using a tool for generating an FMU from modelica code and FMI Toolbox. *If you do not have an FMU generating tool from modelica code, skip step 3.* A pre-compiled FMU is included for convenience.

- Compile a binary model from Modelica code.
- Import the model in Simulink.
- Simulate a Modelica model in Simulink with a simple control system.

For the tutorial, we will use a mechanical system called a Furuta pendulum, see [Jak2003]. The system is shown in Figure 7.1.

Figure 7.1 The Furuta pendulum.

The angle of the pendulum, theta is defined to be zero when in upright position and positive when the pendulum is moving clockwise. The angle of the arm, phi, is positive when the arm is moving in counter clockwise direction. Further, the central vertical axis is connected to a DC motor which applies a torque proportional to the control signal u. The Modelica code for the Furuta pendulum model is given by:

```
model Furuta
  import SI = Modelica.SIunits;
  parameter SI.MomentOfInertia Jp = (m_pa/3 + M)*lp^2;
  parameter SI.MomentOfInertia Ja = 0.00144;
  parameter SI.Length lp = 0.421;
  parameter SI.Length l = (m_pa/2 + M)/(m_pa + M)*lp;
  parameter SI.Mass M = 0.015;
  parameter SI.Length r = 0.245;
  parameter SI.Mass m_pa = 0.02;
  parameter SI.Acceleration g = 9.81;

  parameter SI.Angle theta_0 = 0.1;
  parameter SI.AngularVelocity dtheta_0 = 0;
  parameter SI.Angle phi_0 = 0;
  parameter SI.AngularVelocity dphi_0 = 0;

  output SI.Angle theta(start=theta_0);
  output SI.AngularVelocity dtheta(start=dtheta_0);
  output SI.Angle phi(start=phi_0);
  output SI.AngularVelocity dphi(start=dphi_0);

  input SI.Torque u;

protected
  parameter Real a = Ja + (m_pa + M)*r^2;
  parameter Real b = Jp;
  parameter Real c = (m_pa + M)*r*l;
  parameter Real d = (m_pa + M)*g*l;

equation
  der(theta) = dtheta;
  der(phi) = dphi;
  c*der(dphi)*cos(theta) - b*dphi^2*sin(theta)*cos(theta) +
    b*der(dtheta) - d*sin(theta) = 0;
  c*der(dtheta)*cos(theta) - c*dtheta^2*sin(theta) +
    2*b*dtheta*dphi*sin(theta)*cos(theta) +
    (a + b*sin(theta)^2)*der(dphi) = u;
end Furuta;
```

Note that the model is written on implicit form, i.e., the derivatives der(dphi) and der(dtheta) are given by a system of two equations.

7.1.1. Tutorial

1. Create a new folder on your hard drive, e.g., `C:\Furuta\`

2. Copy the example files to the directory you just created. The example files are included in FMI Toolbox, in the directory `examples\me1\Furuta` under the installation directory. FMI Toolbox is typically located at `C:\Program Files\Modelon\FMI Toolbox 1.3.1`. The following files are needed:

- `furuta.m`
- `Furuta.mo`
- `Furuta.png`
- `Furuta_open_loop.mdl`
- `Furuta_linearization.mdl`
- `Furuta_state_feedback.mdl`

3. In order to use the Furuta model in Simulink, the model Furuta.mo, has to be compiled. Please generate an FMU for Model Exchange version 1.0 with your FMU generating tool from modelica code. Make sure that the FMU is located in `C:\Furuta\` before you continue to the next step. Note that a pre-compiled FMU is included in FMI Toolbox for convenience.

4. Start MATLAB, and make sure that the FMI Toolbox is properly installed by adding the installation directory to MATLAB's paths, see the Installation section for details. Next, type the command:

```
>> simulink
```

Open the file `Furuta_open_loop.mdl`

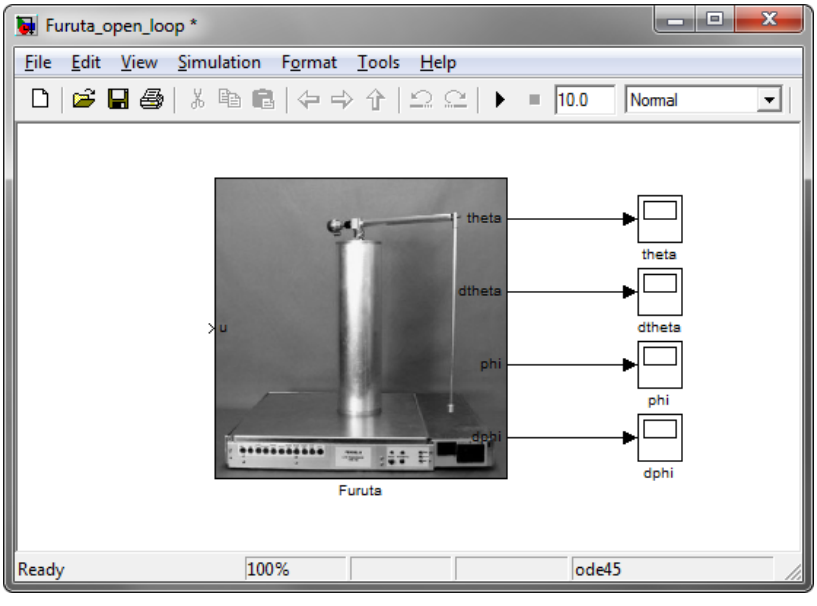


Figure 7.2 A Simulink diagram with the Furuta pendulum.

The Furuta pendulum model is represented by an FMU block in the Simulink diagram, and Simulink Scopes have been connected to the outputs. Press the simulation button and then open the scopes `theta` and `phi`. You should now see plots like the ones below.

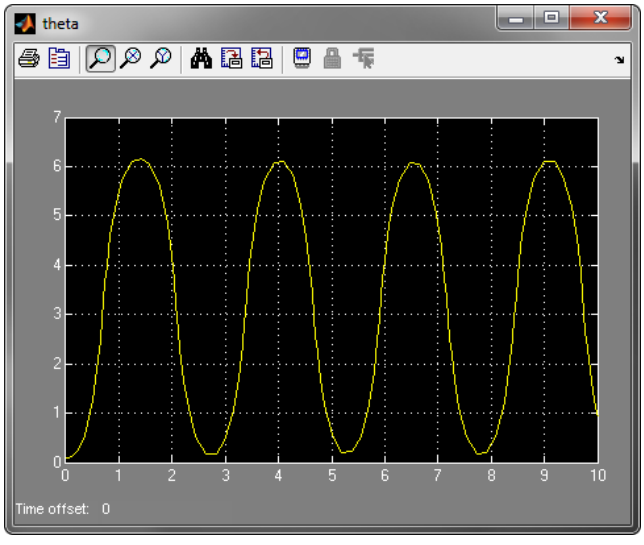


Figure 7.3 Simulation result for the theta angle [rad].

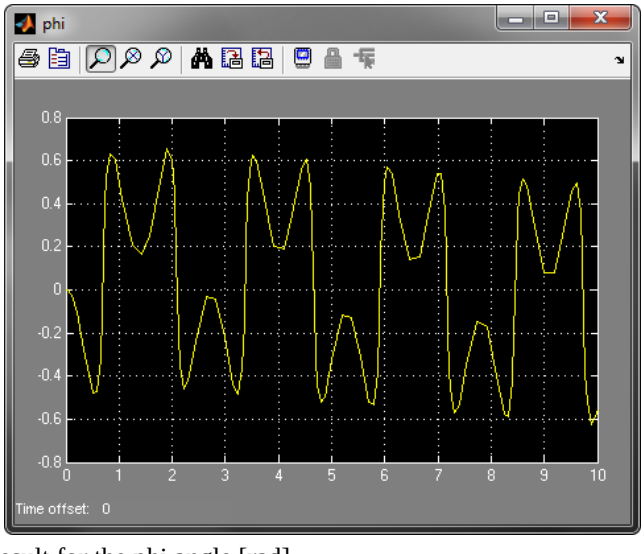


Figure 7.4 Simulation result for the phi angle [rad].

Next, open the FMU dialog by double clicking the FMU block. In the **Parameters & Start values** tab, change the value of the parameter `theta_0` from 0.1 to 1.3. This change corresponds to altering the start value of the pendulum angle from almost upright position to almost aligned to the horizontal plane.

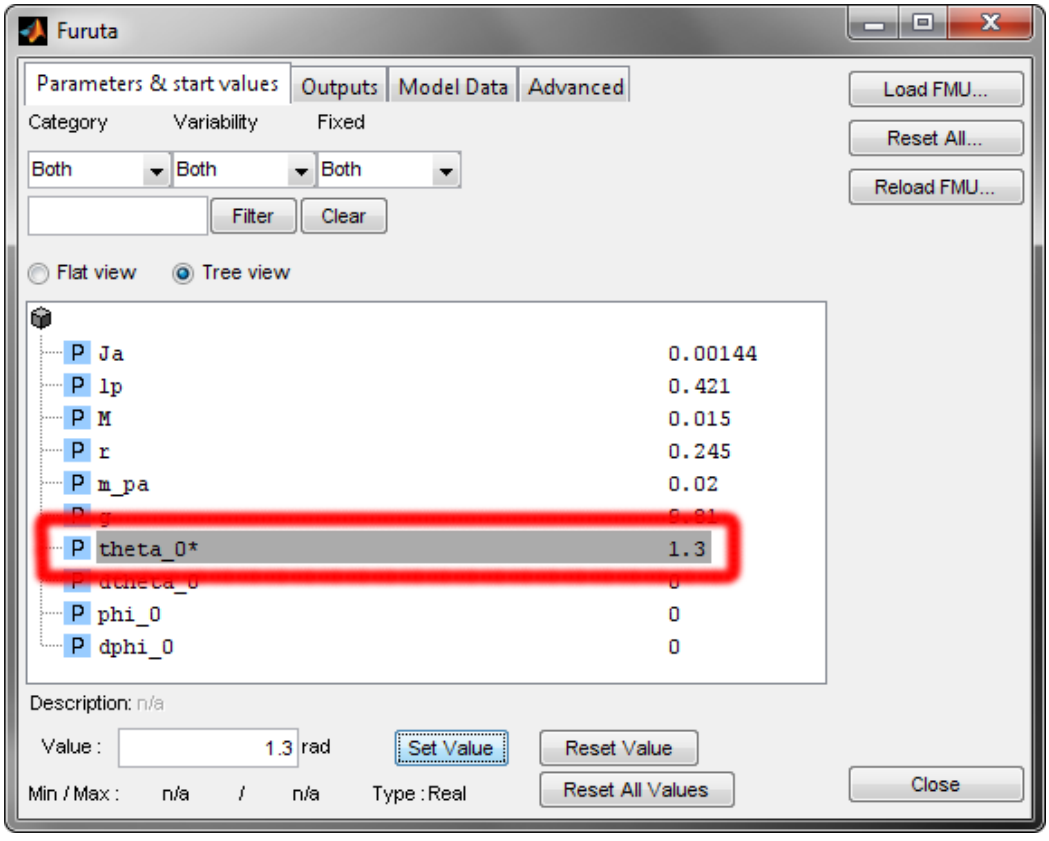


Figure 7.5 Changing of parameter values of the Furuta FMU block.

Simulating the Simulink model again should give a result like the one shown below.

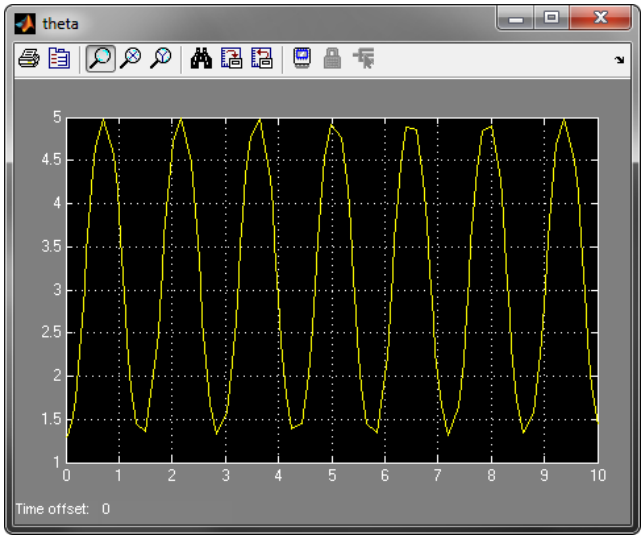


Figure 7.6 Simulation result for the theta angle [rad].

Figure 7.7 Simulation result for the phi angle [rad].

5. Next, we will linearize the pendulum in its upright position, in order to obtain a linear model to use for control design. Open the model `Furuta_linearization.mdl`.

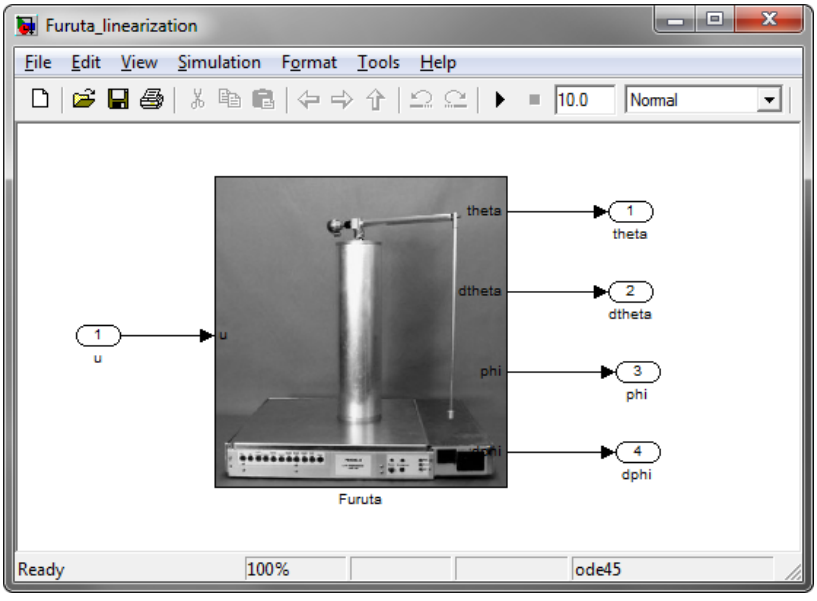


Figure 7.8 The Simulink model used for linearizion of the Furuta model.

The linearizion commands below is found in the `furuta.m` script. In order to linearize the model, we use the MATLAB command `linmod`:

```
[A,B,C,D] = linmod('Furuta_linearization',[0 0 0 0], [0])
```

The A, B, C and D matrices represent a linear state space model for the pendulum in its upright position. In order to simplify the computations, we transform the model so that the ordering of the outputs correspond to the ordering of the Furuta FMU block:

```
% Transform state vector to correspond to output ordering
A = C*A*inv(C)
B = C*B
C = C*inv(C)
```

Finally, we design a linear quadratic state feedback controller using the `lqr` command from the Control Systems Toolbox. If Control Systems Toolbox is not installed on your system, comment the line with the `lqr` command and uncomment the row below.

```
% Compute a state feedback control law
Q = diag([100 10 1 0.25])
R = 100
L = lqr(A,B,Q,R)
%L = [-2.4263   -0.5189   -0.1000   -0.1179]
```

In order to run the script, type the following commands into your MATLAB shell:

```
>> cd C:\Furuta
>> furuta
```

The state feedback control law has now been computed. Next, open the model `Furuta_state_feedback.mdl`.

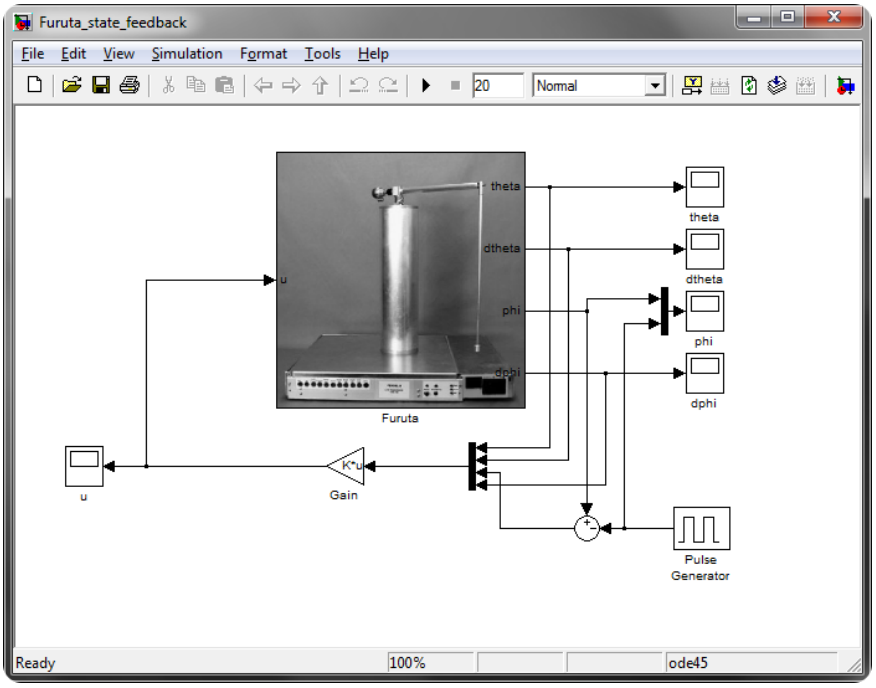


Figure 7.9 The Furuta state feedback Simulink model.

The model contains, apart from the Furuta FMU block, a state feedback control system where the reference value for the arm `angle`, `phi`, is a square wave. Set the simulation time to 20s and simulate the model. Opening of the scopes `theta` and `phi` should give the following plots:

Figure 7.10 State feedback trajectory for the theta angle [rad].

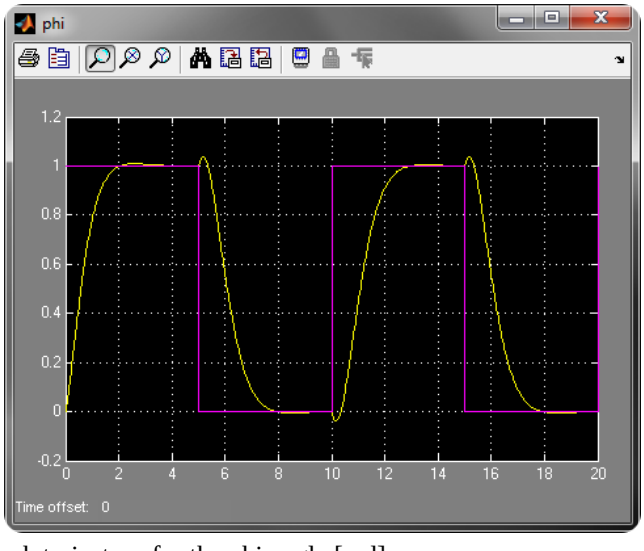


Figure 7.11 State feedback trajectory for the phi angle [rad].

7.1.1.1. Simulate Furuta model with co-simulation block

The Furuta pendulum can also be simulated with a Co-Simulation model. To to this, redo the tutorial above but in step 2, copy the files found from the directory `examples\cs1\Furuta` instead of `examples\me1\Furuta`. In step

3, generate a Co-Simulation 1.0 FMU instead of Model Exchange 1.0. Note that these Simulink models uses the `FMU CS 1.0` block instead of the `FMU ME 1.0` block.

The simulation results from `Furuta_state_feedback.mdl` model is given here. The scopes `theta` and `phi` should give the following plots:

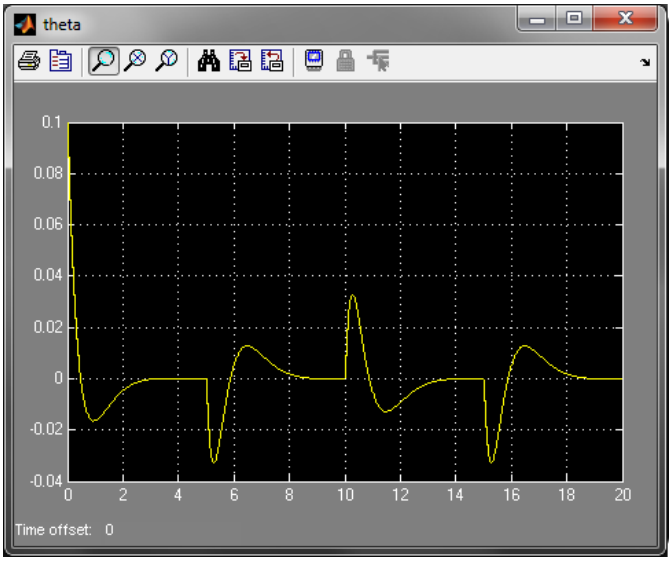


Figure 7.12 State feedback trajectory for the theta angle [rad].

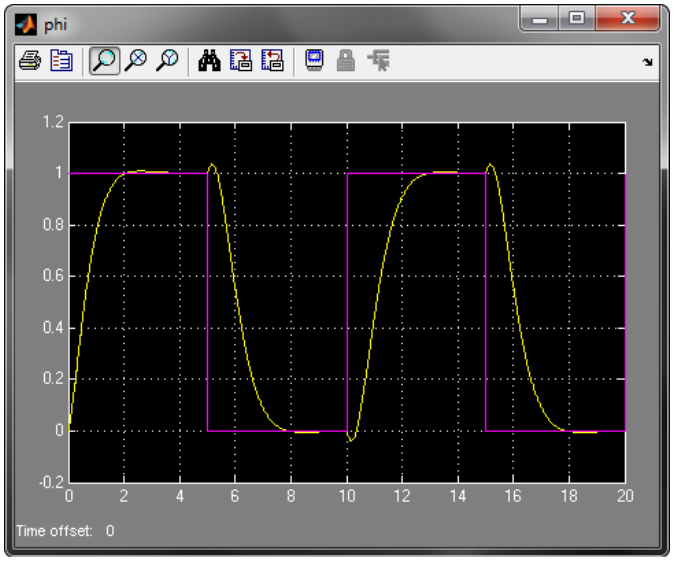


Figure 7.13 State feedback trajectory for the phi angle [rad].

7.2. Vehicle dynamics model simulated in Simulink with a driver

In this tutorial, you will go through the following steps using a tool for generating an FMU from modelica code and FMI Toolbox. *If you do not have an FMU generating tool from modelica code, skip step 2.* A pre-compiled FMU is included for convenience.

- Compile a linear single-track vehicle model into an FMU
- Simulate the vehicle model in an open-loop experiment
- Use a simple driver model to drive the vehicle model around a predefined path

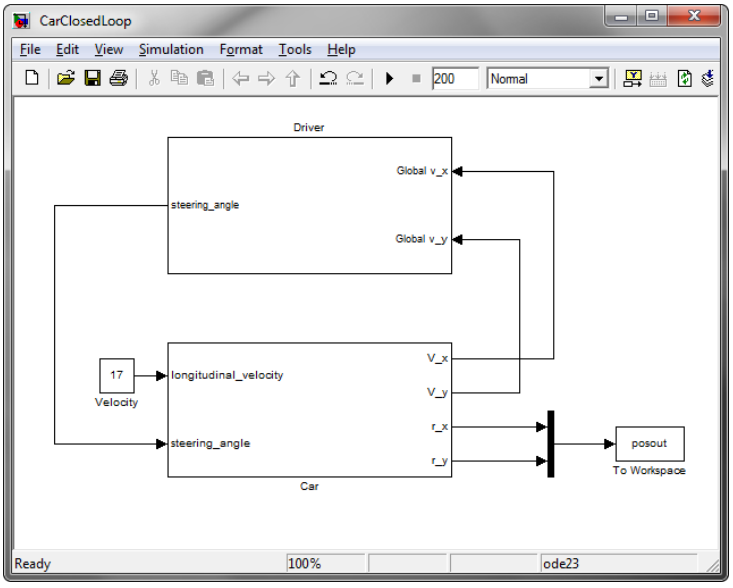


Figure 7.14 Simulating the FMU Car model in Simulink connected to a Driver block created in Simulink.

7.2.1. Tutorial

1. Setting up project

   - To start, create a folder somewhere on your hard drive, e.g. `C:\VehicleDynamics`
   - Copy the example files to the directory you just created. The example files are located in the directory `examples\me1\VehicleDynamics` under the installation directory. The following files are needed:
     - `Car.mo`
     - `CarOpenLoop.mdl`
     - `CarClosedLoop.mdl`

     If you do not have an FMU generating tool from modelica code, then also copy `Car.fmu` and go to step 3.

2. In order to use the Car model in Simulink, the model Car.mo, has to be compiled. Please generate an FMU for Model Exchange version 1.0 with your FMU generation tool for modelica code. Make sure that the FMU is located in `C:\VehicleDynamics\` before you continue to the next step. Note that a pre-compiled FMU is included in FMI Toolbox for convenience. The Modelica code for the Car model is given by:

```
model Car "Linear single-track vehicle model"
parameter Modelica.SIunits.Length l_f=1 "Distance from front axle to c.o.g.";
```

```
parameter Modelica.SIunits.Length l_r=1 "Distance from rear axle to c.o.g.";
parameter Modelica.SIunits.Mass m=1000 "Mass";
parameter Modelica.SIunits.Inertia i_zz=2500
         "Moment of inertia around vertical axis";
parameter Real C_f=100000 "Front axle cornering stiffness";
parameter Real C_r=100000 "Rear axle cornering stiffness";
parameter Real k_sw=1 "Steering gain";

input Modelica.Blocks.Interfaces.RealInput steering_angle
         "Steering wheel angle input";
input Modelica.Blocks.Interfaces.RealInput longitudinal_velocity
         "Longitudinal velocity input";

Modelica.SIunits.Angle alpha_f "Front slip angle";
Modelica.SIunits.Angle alpha_r "Rear slip angle";
Modelica.SIunits.Force f_y_f "Front axle lateral force";
Modelica.SIunits.Force f_y_r "Rear axle lateral force";
Modelica.SIunits.AngularVelocity w_z "Yaw rate";
Modelica.SIunits.Position p_z "Yaw angle";
Modelica.SIunits.Velocity v_y "Lateral velocity";
Modelica.SIunits.Velocity v_x "Longitudinal velocity";
Modelica.SIunits.Acceleration a_y "Lateral acceleration";

Modelica.SIunits.Angle delta=k_sw*steering_angle "Steering angle at wheels";

output Modelica.SIunits.Position r_x "Global X position";
output Modelica.SIunits.Position r_y "Global Y position";

output Modelica.SIunits.Velocity V_x "Global X velocity";
output Modelica.SIunits.Velocity V_y "Global Y velocity";


equation
v_x = max(0.1,longitudinal_velocity) "Avoid division by zero for low speeds";

alpha_f = (-v_y-l_f*w_z)/v_x+delta "Front axle slip angle (assuming small angles)";
alpha_r = (-v_y+l_r*w_z)/v_x "Rear axle slip angle (assuming small angles)";

f_y_f = C_f*alpha_f "Front axle lateral force";
f_y_r = C_r*alpha_r "Rear axle lateral force";

a_y = der(v_y)+v_x*w_z "Lateral acceleration";

m*a_y = f_y_f+f_y_r "Lateral force balance";
i_zz*der(w_z) = l_f*f_y_f-l_r*f_y_r "Torque balance around vertical axis";

der(p_z) = w_z "Yaw angle output";

V_x=v_x*cos(p_z)-v_y*sin(p_z) "Global X velocity output";
V_y=v_x*sin(p_z)+v_y*cos(p_z) "Global Y velocity output";
der(r_x)=V_x "Global X position output";
```

```
der(r_y)=V_y "Global Y position output";
end Car;
```

3. Simulate the model in Simulink

   - In Simulink, open the model `CarOpenLoop.mdl` from your project directory.

   - A step steer maneuver is prepared with a constant velocity input and some outputs have been selected and routed into a scope block. Simulate the model and you should see the following plots in the scope:

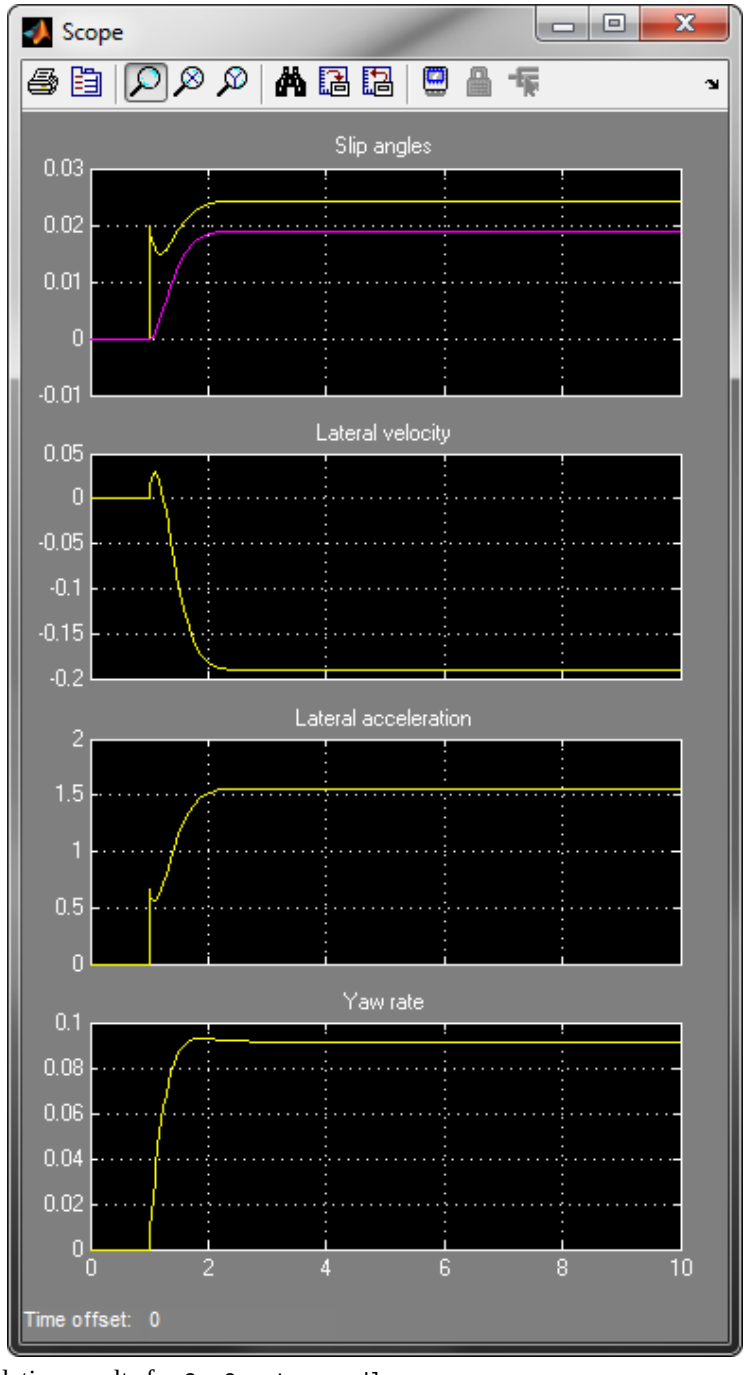


Figure 7.15 Simulation results for `CarOpenLoop.mdl`

4. Closed loop simulation in Simulink

   - Now open `CarClosedLoop.mdl` instead. Here the vehicle model is connected to a driver model with a simple path tracking controller. The path to follow is defined as a distance-curvature interpolation table:

     Table 7.1 Distance-curvature interpolation table of the path for the driver to follow

Distance along path	Curvature
0	0
50	0
50+50	1/200
50+2*pi*200	1/200
50+2*pi*200+50	-1/100
50+2*pi*200+2*pi*100	-1/100

   - In the experiment, the path is defined as two full circles with two different curve radii. First a left turn with r=200m then a right turn with r=100m. A 50m transition distance is used when changing curvature.

   - Simulate the model.

   - To plot the position of the vehicle the following commands can be used:

```
figure;
plot(posout.signals.values(:,1),posout.signals.values(:,2));
axis equal;
```

     The following plot should appear:

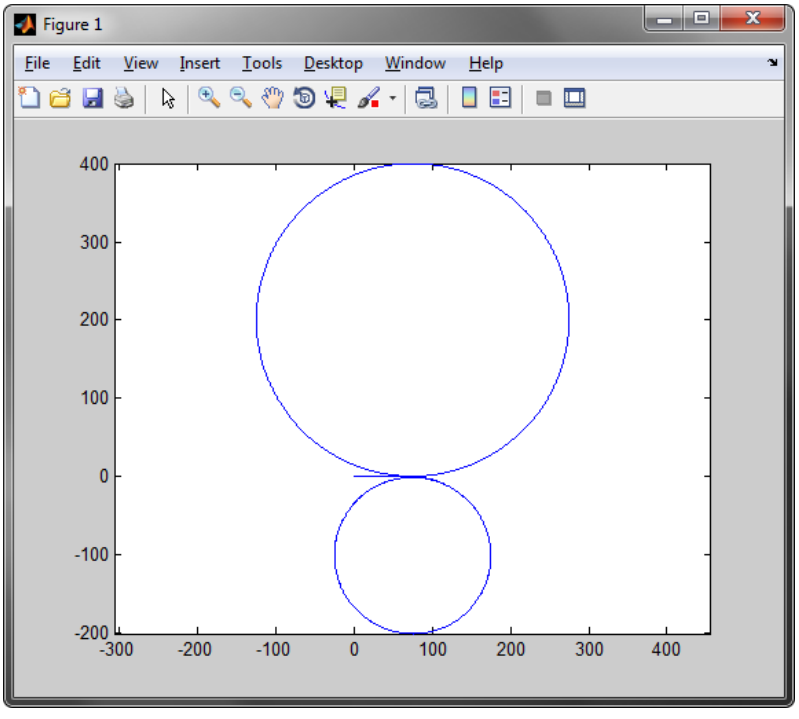


Figure 7.16 Vehicle position

- In the parameter dialog of the FMU block, add outputs for lateral acceleration (`a_y`) and yaw rate (`w_z`) and add scopes to view the output. After simulating, you should see the following plots:

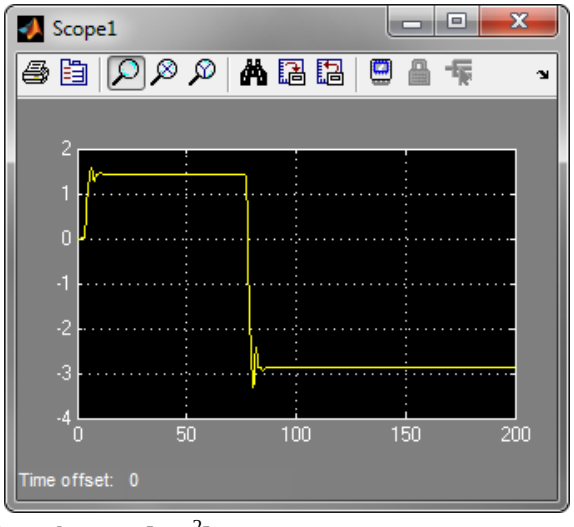


Figure 7.17 Plot of lateral acceleration [m/s^2]

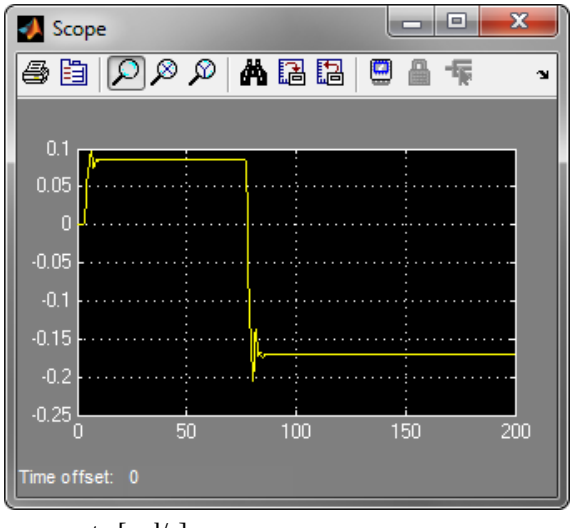


Figure 7.18 Plot of vehicle yaw rate [rad/s]

Chapter 8. Limitations

This page lists the current limitations of the FMI Toolbox.

8.1. Simulink FMU block

- The input and output port does not support strings.
- For large models (above 30 thousand variables, depending on the computer), the tree views in the GUI may take long time to draw. It is recommended to use the structured tree view and not the flat view.
- Co-Simulation FMUs with modelDescription attribute canRunAsynchronuously set to true are not supported.
- When Simulink Coder/Real-Time Workshop builds a model containing multiple FMU blocks, interference may occur due multiple source code FMUs may include different files with the same name using the *#include <...>* include directive.
- Simulating with Rapid Accelerator Mode is not supported when using FMU blocks.

8.2. MATLAB FMU Classes

- DOE analysis is not supported for Model Exchange 2.0 FMUs.
- No analytical Jacobain will be used when simulating Model Exchange 2.0 FMUs.
- The FMI functions fmi2GetFMUstate, fmi2SetFMUstate, fmi2FreeFMUstate, fmi2SerializedFMUstateSize, fmi2SerilizeFMUstate and fmi2DeSerializeFMUstate are not implemented.
- It is not possible to access the dependency information of variables present in the ModelStructure tag in the XML for 2.0 FMUs.

8.3. FMU Export

8.3.1. Common target

- Complex input and output ports are not supported. There is no corresponding data type in the FMI standard. Complex parameters will not be exposed in the FMU.
- Fixed-point input and output ports are not supported. There is no corresponding data type in the FMI standard. Fixed-point parameters will not be exposed in the FMU.

- Discrete variables (variability attribute set to discrete) may change value at instants other than during initialization or at event instants.
- Start values NaN and Inf are not supported for exposed parameters.

In Table 8.1 unsupported blocks or blocks with restricted usage are listed. For a full list with blocks that have been tested, see Section 5.10.

Table 8.1 Unsupported or restrictions on blocks

Block	Comment
simulink/Continuous/Variable Time Delay	Partial supported. Generates different results.
simulink/Continuous/Variable Transport Delay	Partial supported. Generates different results.
simulink/Continuous/Derivative	Partial supported. Derivative approximation is dependent on the length of the integrator step which causes the results to be different.
simulink/Discontinuities/Backlash	Partial supported. Generates different results.
simulink/Math Operations/Weighted Sample Time Math	Not supported when continuous sample times are used. See note[a]
simulink/Math Operations/Algebraic Constraint	Not supported. Algebraic loops are not supported in generated code.
simulink/Model Verification/Check Discrete Gradient	See note [a]. Requires fixed-step solver, see note [b].
simulink/Model-Wide Utilities/Trigger-Based Linearization	Not supported, see note [c].
simulink/Model-Wide Utilities/Timed-Based Linearization	Not supported, see note [c].
simulink/Ports & Subsystems/Model	Not supported. FMU target is not model reference compliant.
simulink/Ports & Subsystems/Model Variants	Not supported. FMU target is not model reference compliant.
simulink/Ports & Subsystems/Variant Subsystem	Not supported. FMU target is not model reference compliant.
simulink/Ports & Subsystems/For Each Subsystem	Not supported for Model Exchange, see note [d].
simulink/Sinks/To File	Supports only "Save format" set to *Array*. *TimeSeries* are not supported by Simulink Coder/Real-Time Workshop generated code.
simulink/Sources/Enumerated Constant	Not supported. Enumerator is not supported by the target.

Block	Comment
simulink/Sources/Pulse Generator	Uses a variable sample time, see note [e].
simulink/Sources/Counter Free-Running	Due to the nature of the block, the output depends on how many times the FMI functions are called which varies between different FMI import tools and solver settings.
simulink/User-Defined Functions/MATLAB Fcn (renamed in 2011a, see *Interpreted MATLAB Function*)	Not supported. Not yet supported by Real-Time Workshop/Simulink Coder.
simulink/User-Defined Functions/Interpreted MATLAB Function (new since 2011a)	Not supported. Not yet supported by Real-Time Workshop.
simulink/User-Defined Functions/S-Function Builder	Supported if TLC file is generated.
simulink/Additional Math & Discrete/Additional Discrete/Transfer Fcn Direct Form II	Use discrete sample time.
simulink/Additional Math & Discrete/Additional Discrete/Transfer Fcn Direct Form II Time Varying	Use discrete sample time.
simulink/Additional Math & Discrete/Additional Discrete/Fixed-Point State-Space	Use discrete sample time.

[a]Not supported by the S-function *CodeFormat* which the the FMU target is derived from.

[b]Limited by the S-function *CodeFormat* which the the FMU target is derived from.

[c]TLC-file for the block is missing. Code for the block cannot be generated.

[d]Block is not supported for generation of a Simulink Coder/Real-Time Workshop target.

[e]Not supported by the S-function *CodeFormat* which the the FMU target is derived from.

8.3.2. Model Export target

- The Model Exchange target uses the code format *S-function* and target type *non real time*. This means in general that the same limitations of Simulink Coder´s native S-function target, *rtwsfcn* is applied to the FMU target. For more information about S-function generation limitations, go to http://www.mathworks.se/help/rtw/ug/generated-s-function-block-deployment.html.

8.3.3. Co-Simulation target

- Only Fixed-step solvers are supported.
- *Support for precompiled S-functions* is only supported for export of Model Exchange FMUs.

Chapter 9. License installation

9.1. Retrieving a license file

There are different types of license models that can be used with Modelon products.

- *Node-locked* (No license server required)

  This license enables use on a single computer. The license cannot be moved from one computer to another. The license is locked for use on a computer with a specific MAC address.

- *Server* (Requires a license server)

  This licensing model represents a classic network configuration with a server and users. The server grants or denies requests from computers in the network to use a program or feature. The license file specifies the maximum number of concurrent users for a program or feature. There is no restriction for which computer is using the program or feature, only in the number of programs and features that can be used simultaneously.

  The computer on which the server is running cannot be changed. The server computer's MAC address must be provided to Modelon to generate the license file.

- Evaluation license (*Node Locked*)

  This license enables a program or feature for a limited amount of time and is the same as a node-locked license.

Please contact the Modelon sales department at `<sales@modelon.com>` to purchase a license or to get an evaluation license. In order to obtain a license file for a node-locked license, you must provide the MAC address of your computer. If you are using a license server, you must provide the MAC address of the server. In Section 9.1.1 below, you will find instructions for how to retrieve the MAC address of a computer.

9.1.1. Get MAC address

Modelon uses the Ethernet address (MAC address), also called the host ID, to uniquely identify a specific computer. Therefore, you must provide the MAC address of the computer on which you want to use the program or feature. For a server license, the MAC address for the server computer is required, not all the client computers in the network that will use the program or feature. For a node-locked license, the MAC address of the computer on which the license will be used must be provided.

Note: Modelon only allows ONE MAC address for each computer. Please disable and unplug all network devices that are not permanently connected to the computer such as laptop docking stations, virtual machines and USB network cards.

- Windows

1. Open **cmd**

   Windows 7 and Vista

   a. Click the **Start** button

   b. Type **cmd** in the search bar and press enter.

   Windows XP

   a. Click the **Start** button.

   b. Click on **Run...**.

   c. Type **cmd** in the text box and click **OK**.

2. Run `lmhostid.exe`.

   Type the full path to lmhostid.exe within quotes and press enter. `lmhostid.exe` is normally located in `<installation folder>\license_tools\lmhostid.exe`.

3. Use this hostid when you are in contact with Modelon. If multiple hostids are listed, select one that is permanent for the computer.

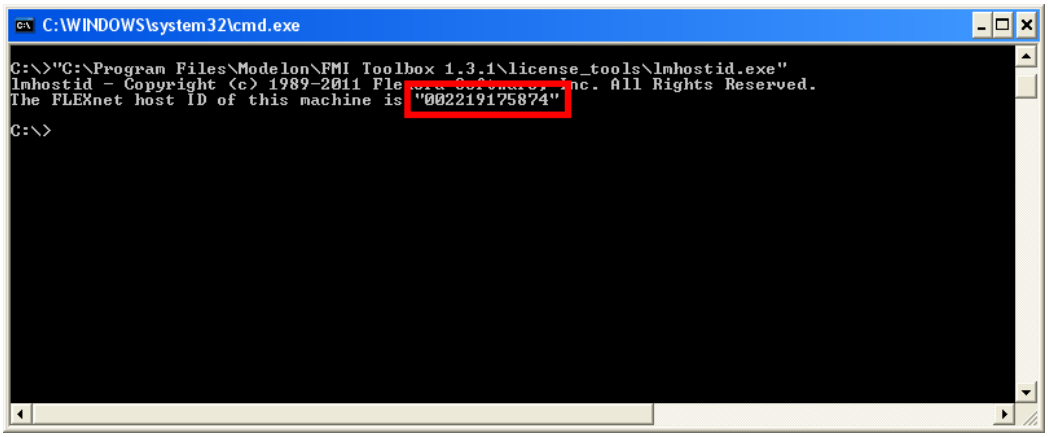


Figure 9.1 Lmhostid.exe run on Windows listing the computer's MAC address.

- Unix

1. Open a terminal and change directory to the `<installation folder>/license_tools/`.

Run `lmhostid` and use the hostid listed when you are in contact with Modelon. If multiple hostids are listed, select one that is permanent for the computer.

9.2. Install a license

After purchasing a license, you should receive a license file with the file extension `*.lic`. This file must be put in a specific folder for the application to find it.

9.2.1. Installing a node-locked license

9.2.1.1. Windows

1. Close the application if it is already running.
2. Open the *Application Data* folder.

   Windows 7 and Windows Vista

   a. Click the **Start** button.

   b. Type `shell:AppData` in the search bar and press enter.

   Windows XP

   a. Click the Start button.

   b. Click on **Run...**.

   c. Type `shell:AppData` in the text box and click **OK**.
3. The *Application Data* folder should now be open. Check that its path is of the form *C:\Users\YourUserName\AppData\Roaming*.
4. Create the folder `Modelon\Licenses\Nodelocked` if it does not exist already.
5. Put your license file in the folder `Nodelocked`.

9.2.1.2. Unix

- Copy your license file to the folder `<installation folder>\Licenses\Nodelocked`.

9.2.1.3. Updating the license

To update the license file, you should overwrite the old license file with the new one. Ensure that the old license file is overwritten or removed from the folder since it may otherwise be used instead of the new one, and the application may fail to check out a license. Note that you must restart the program for license changes to take effect.

9.2.2. Installing a server license

Note that these are not instructions for installing a license file on a server. These are instructions for the end user of the program or feature. The assumption is that the server is already up and running and that the IP address to the server and the port number is already known. The IP address and the port number, if needed, should be provided by the license server administrator.

The application can connect to the license-server and daemon either by reading a license file or an environment variable.

9.2.2.1. Windows

1. Close the application if it is already running.
2. Create an empty text file

   Windows 7 and Windows Vista

   a. Click the **Start** button.

   b. Type `Notepad` in the search bar and press enter.

   Windows XP

   a. Click the **Start** button.

   b. Click on **Run...**.

   c. Type `Notepad` in the text box and click **OK**.
3. Configure the license file.

   a. Copy the following text in to the text document

   ```
   SERVER <ip-address> ANY <port>
   USE_SERVER
   ```

   b. Change `<ip-address>` to the IP address of the server.

c. Change `<port>` to the port number that is being used. If you do not have a port number, you can remove the whole `<port>`. For example, the license file should look like the following for a license server with IP address 192.168.0.12 using port 1200.

```
SERVER 192.168.0.12 ANY 1200
USE_SERVER
```

d. Save the file with a filename with the extension `*.lic` in a temporary place. The file will be moved in a later step. You can now close Notepad.

4. Open the *Application Data* folder.

Windows 7 and Windows Vista

a. Click the **Start** button.

b. Type `shell:AppData` in the search bar and press enter.

Windows XP

a. Click the **Start** button.

b. Click on **Run...**.

c. Type `shell:AppData` in the text box and click **OK**.

The *Application Data* folder should now open.

5. Create the folder `Modelon\Licenses\Server` if it does not exist already.

6. Put the license file you just created in the folder `Server`.

9.2.2.2. Unix

1. Close the program if it is already running.

2. Create an empty file with the file extension name `*.lic`.

3. Configure the license file.

a. Copy the following text in to the text document

```
SERVER <ip-address> ANY <port>
USE_SERVER
```

b. Change `<ip-address>` to the IP address of the server.

c. Change <port> to the port number that is being used. If you do not have a port number, you can remove the whole <port>. For example, the license file should look like the following for a license server with IP address 192.168.0.12 using port 1200.

```
SERVER 192.168.0.12 ANY 1200
USE_SERVER
```

4. Copy your license file to the folder <installation folder>\Licenses\Server.

9.2.2.3. Using the environment variable

An alternative to specify how the application should connect to the license server is to set the environment variable MODELON_LICENSE_FILE. The value can be set to *port@host*, where port and host are the TCP/IP port number and host name from the SERVER line in the license file. Alternatively, use the shortcut specification, @host, if the license file SERVER line uses a default TCP/IP port or specifies a port in the default port range (27000–27009).

9.2.2.4. Updating the license

To update the license file, you can either redo the installation instructions described above or make the changes in the license file directly. Ensure that the old license file is overwritten or removed from the folder since it may otherwise be used instead of the new one, and the application may fail to check out a license. Note that you must restart the program before the changes can take effect.

9.3. Installing a license server

To install a license server, you must have a server license file. Please contact <sales@modelon.com> to obtain the server license file. This license file must also be configured prior to use by by setting the IP address and port as shown in Section 9.3.1

Modelon products use a licensing solution provided by Flexera Software. It is recommended that you install the latest version of the server software, which is available from http://learn.flexerasoftware.com/content/ELO-LM-GRD . Modelon products require a license server version number v11.10.0.0 or later. A license server and a license daemon are required and are distributed with the product you are installing. If you have not received the server application or license daemon with your product, please contact <sales@modelon.com>.

The following step by step instructions for installing a license server assume that no other Flexera license server is already installed.

9.3.1. Configure the license file

When a license server is installed, the server needs a license file provided by Modelon. This file must be configured before it can be used.

1. Open the license file in a text editor. The file may look like the example below:

```
SERVER 192.168.0.1 080027004ca5 25012
VENDOR modelon
FEATURE FMI_TOOLBOX modelon 1.0 3-feb-2012 12 SIGN="0076 305..."
```

2. Edit the SERVER line where the IP address, `192.168.0.1`, should be replaced with the IP address of the server. Also change the port address, `25012`, to the desired port or remove it to use default ports. The IP address and potentially also the port address should be provided to the end users so they can configure their license files to connect to the server.

9.3.2. Installation on Windows

In the `<installation folder>\license_tools` folder that is distributed with your product, you will find the files listed below.

The listed files are used to set up and configure the license server.

- *lmgrd.exe* (license server)
- *modelon.exe* (license daemon)
- *lmutils.exe* (configure- and utility functions)
- *lmtools.exe* (Windows GUI for setting up the license server as a Windows service)

To configure a license server manager (`lmgrd`) as a service, you must have Administrator privileges. The service will run under the LocalSystem account. This account is required to run this utility as a service.

1. Make sure that license daemon `modelon.exe` is in the same folder as the license server, `lmgrd.exe`.
2. Run `lmtools.exe`
3. Click the **Configuration using Services** button, and then click the **Config Services** tab.
4. In the **Service Name**, type the name of the service that you want to define, for example, `Modelon License Server`.
5. In the **Path to the lmgrd.exe** file field, enter or browse to `lmgrd.exe`.
6. In the **Path to the license file field**, enter or browse to the server license file.
7. In the **Path to the debug log file**, enter or browse to the debug log file that this license server should write. Prepending the debug log file name with the + character appends logging entries. The default location for the debug log file is the `c:\winnt\System32 folder`. To specify a different location, be careful to specify a fully qualified path.

8. Make this license server manager a Windows service by selecting the **Use Services** check box.

9. **Optional**. Configure the license server to start at system startup time by selecting the **Start Server at Power Up** check box.

10. To save the new `Modelon License Server` service, click **Save Service**.

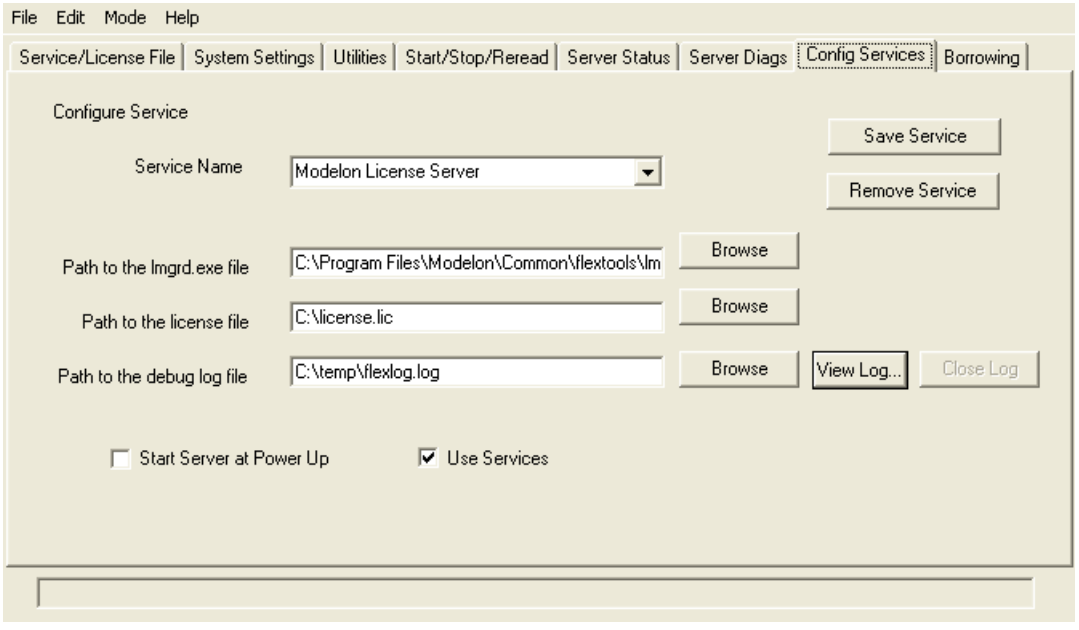


Figure 9.2 Setup the license server with lmtools.exe

11. Click the **Service/License File** tab. Select the service name from the list presented in the selection box. In this example, the service name is `Modelon License Server`.

12. Click the **Start/Stop/Reread tab**.

13. Start `Modelon License Server` by clicking the **Start Server** button. `Modelon License Server` license server starts and writes its debug log output to the file specified in the **Config Services** tab.

9.3.3. Installation on Unix

In the `<installation folder>\license_tools` folder that is distributed with the product, you can find the files listed below.

- *lmgrd* (license server)
- *modelon* (license daemon)

- *lmutil* (configure- and utility functions)

Before you start the license server, *lmgrd*, make sure that license daemon *modelon* is in the same folder.

Start `lmgrd` from the UNIX command line using the following syntax:

```
lmgrd -c license_file_list -L [+]debug_log_path
```

where `license_file_list` is either the full path to a license file or a directory containing license files where all files named `*.lic` are used. If the license_file_list value contains more than one license file or directory, they must be separated by colons. `debug_log_path` is the full path to the debug log file. Prepending `debug_log_path` with the + character appends logging entries.

Starting `lmgrd` from a root account my introduce security risks, and it is therefore recommended that a non-root account is used instead. If `lmgrd` must be started by the root user, use the `su` command to run `lmgrd` as a non-privileged user:

```
su username -c "lmgrd -c license_file_list -l debug_log_path"
```

Ensure that the vendor daemons listed in the license file have execute permissions for `username`.

9.4. Troubleshooting license installation

If you experience any problems with the license, the error messages are usually descriptive enough to provide hints as to the root cause of the problem. If the problem persists, please contact Modelon at `<support@modelon.com>`. Before contacting Modelon, support you should run `lmdiag` and provide the resulting information. Follow the instructions below to run `lmdiag`.

9.4.1. Running lmdiag

- Windows

1. Open **cmd**

   Windows 7 and Vista

   a. Click the **Start** button.

   b. Type **cmd** in the search bar and press enter.

   Windows XP

   a. Click the **Start** button.

   b. Click on **Run...**.

c. Type **cmd** in the text box and click **OK**.

2. Run `lmdiag.exe`.

   Type the full path to `lmdiag.exe` within quotes and press enter. `lmdiag.exe` is normally located in `<installation folder>\license_tools\lmdiag.exe`.

- Unix

  - Open a terminal and change directory to the `<installation folder>/license_tools/`.

    Run `lmdiag` with the `./lmutil lmdiag` command.

Note: `lmutil` requires LSB (Linux Standard Base) compliance to run. Some distributions, e.g. Ubuntu, do not have LSB compliance by default and can thus not run the program. `./lmutil` then fails with a message like

```
$> ./lmutil lmdiag
 bash: ./lmutil: No such file or directory
```

If this error occurs, please check if the required interpreter is installed on your system. The requirement can be found with the `readelf` command, and the output should look similar to

```
$> readelf -a lmutil | grep interpreter
    [Requesting program interpreter: /lib64/ld-lsb-x86-64.so.3]
```

LSB should be available for install through a package manager. If installing it is not an alternative, a quick fix is to symlink the required interpreter to the one on your system, i.e.

```
$> ln -s <your ld> <required ld>
```

Note 2: FlexLM requires that network devices are named *eth0*, *eth1*, etc. When other names are used, `lmhostid` will always return 0 as host ID. Device names can be shown with the `ifconfig` command. If your Linux distribution uses a different naming scheme, it needs to be changed. The steps to change the naming scheme depend on the distribution and release.

Chapter 10. Release notes

10.1. Release 2.6.4

- Added support for MATLAB 2018a
  - Due to a glibc bug that exists for the glibc of Ubuntu 14.04 MATLAB 2018a is currently not supported on Linux (Ubuntu 14.04), the related bug report can be found at: https://bugs.launchpad.net/ubuntu/+source/eglibc/+bug/1695080.
- Other minor improvements and bug fixes.

10.2. Release 2.6.3

- Added support for MATLAB 2017b.
- Added the new FMU block options `useStaticLibraryInCodeGeneration` and `staticLibraryPathForCodeGeneration`. These allow using custom FMU binaries when using Simulink Coder. For more information use `help fmuSetOptionSimulink` in MATLAB.
- For the tables in Section 5.10, compatibility with the FMU targets was added for the following blocks:
  - Argument Inport
  - Argument Outport
  - Function Caller
  - Matlab System
  - Simulink Function
  - Event listener
  - Initialize Function
  - Reset Function
  - State Reader
  - State Writer
  - Terminate Function
- Other minor improvements.

10.3. Release 2.6.2

- Added support for Visual Studio 2013.
- Fixed bug where the wrong FMU resource path was used by the FMU block when simulating in Accelerator mode.
- Other minor improvements.

10.4. Release 2.6.1

- Added support for MATLAB 2017a.
- Fixed bug where adding outputs in the FMU Block GUI failed.
- Other minor improvements.

10.5. Release 2.6

- Better support and documentation for using the FMU block with Simulink Coder targets, see Section 3.4.
- Improved performance for Simulating Model Exchange 2.0 FMUs using the mex interface.
- Updated the FMI Library version to 2.0.2.
- Added warning for not creating string ports when loading an FMU with string inputs/outputs into an FMU block.
- Improved logging for the FMU block. Ensured that for MATLAB 2015b and forward logging is done in the Diagnostic Viewer when using the GUI, otherwise to the MATLAB prompt.
- Other improvements and bug fixes.

10.6. Release 2.5

- Added support for Exporting FMUs on Linux. See Section 2.2.1.2 for details.
- It is no longer necessary to configure a zip tool. A default implementation in Java is now available.
- Added trim and linearize to FMUModelME2 class, also improved the FMUModelME1 implementations with better interface and documentation.
- Note: The FMIToolbox 2.5.x releases will be the last to have support for Windows XP, Ubuntu 8.04, Ubuntu 11.04 and early versions of MATLAB. FMIToolbox 2.6 will support MATLAB 2010b (32-bit and 64-bit) and forward on Windows (7 and 10) and MATLAB 2015a (64-bit) and forward on Linux (64-bit).
- Other improvements and bug fixes.

10.7. Release 2.4

- Added support for MATLAB 2016a and 2016b.
- Added support for Visual Studio 2012 and Visual Studio 2015.

- Added the FMU export option **Include internal signals**. When enabled, internal signals will be available in the exported FMU. See Section 5.8 for more information.
- Removed the vector sizes from the vector names for ports in FMU blocks when loading an FMU. Reloading the FMU for a block will not change the naming. For example the names for the vector with FMI names "a[1]", "a[2]", "a[3]" will now have the block port name "a" rather than "a[3]".
- Most of the scripting functions for the FMU block (see Section 3.3.8) now works for FMU blocks with structured ports activated. Improved error handling for calling the scripting functions with blocks not valid.
- Other improvements and bug fixes.

10.8. Release 2.3.3

- Better support for removing algebraic loops using dependency data. New FMU blocks will use this by default. The option `useDirectFeedthroughData` can be used for old blocks.
- Unconnected inputs of FMU blocks now behave as if the Ground block was connected to it, meaning that the input will be a zero value.

10.9. Release 2.3.2

- Support for FMU Model Exchange export from Simulink with Global (tunable) parameters.
- Added full support for MATLAB 2015b.

  Since the `revertInlineParametersOffTo2013b` command is removed in MATLAB 2015b, it is no longer used in FMU Model Exchange export.
- Exported FMUs of Simulink models will now have parameters with names that reflect the models structure, see Chapter Section 5.7.
- The structured names of the inputs and outputs of exported FMUs will now always be legal FMI names (unique and with no illegal characters).
- Fixed bug where modifying a copied FMU block with structured ports would change the original FMU block.

10.10. Release 2.3.1

- Tunable parameters are now supported for FMU Co-Simulation 2.0 export.
- Added support for MATLAB 2015b with the exception of the targets fmu_me1.tlc and fmu_me2.tlc.
- Added the methods getMin, getMax and getNominal to the ScarlarVariable1 class.

- Fixed a bug causing the fmiGetXXX functions in the FMI 1.0 MEX interface to return values with wrong dimensions.
- Other improvements and bug fixes.

10.11. Release 2.3

- Support for FMU Model Exchange 2.0 export from Simulink.
- Changed behavior for Stop Simulation blocks when exporting Co-Simulation and Model Exchange FMUs. They now cause the simulation to stop without an error (Model verification blocks still causes stops with an error).
- Improved performance of ScalarVariable1 methods.
- Exported Co-Simulation FMUs now support variable communication points. Capability added to modelDescription.xml: `canHandleVariableCommunicationTimeStep="true"`.
- Fixed a bug affecting S-Functions with row vector parameters.

10.12. Release 2.2.1

- Added optional argument to ScalarVariable1.directDependency to check dependency only on specified input variables. This greatly improves performance.
- Fixed a bug where simulation of imported FMU:s would not work when using interpolated input signals.

10.13. Release 2.2

- Support for loading and simulating 2.0 FMUs in MATLAB. See Chapter 8 for what is not yet implemented.
- New methods set and get for the MATLAB interface, setValue and getValue is deprecated.
- Added support for MATLAB 2015a.

10.14. Release 2.1

- Support for FMU Co-Simulation 2.0 export from Simulink.
- FMU blocks are now inlined when loaded with 2.0 FMUs and are then supported by Simulink Coder/Real-Time workshop.

10.15. Release 2.0.1

- Better handling of finding resources for exported Co-Simulation and Model Exchange FMUs.

10.16. Release 2.0

- Support for loading and simulating 2.0 FMUs with the FMU blocks.
- Optimized the time for loading large FMUs into the FMU blocks.
- Optimized the time for opening the GUI of the FMU blocks loaded with a large FMU.
- **Write simulation result to file** and the **Logger** drop down menu in the GUI have been moved from the **Advanced** tab to the new **Log** tab.
- Other improvements and bug fixes.

10.17. Release 1.9

- Added support for MATLAB 2014a and MATLAB 2014b.

  From MATLAB 2014a and later, FMU Model Exchange export calls *revertInlineParametersOffToR2013b*. See MATLAB 2014a release notes for details regarding implication of this function call.
- Changed linger time from 30 minutes to 2 minutes for **FMI Toolbox Coder add-on**. Changed linger time from 0 minutes to 2 minutes for **FMI Toolbox** license.
- Updated MATLAB class constructors FMUModelME1, FMUModelCS1 and loadFMU to take arguments for setting log file name and instance name. The new interface use name value pair arguments. The old interface for setting log level is deprecated.
- New method getScalarVariable added. It returns a ScalarVariable1 class given a variable name.
- New method getInstanceName added. It returns the instance name.
- New method getLogFilePath added. It returns the full log file path.
- New method getFMUFilePath added. It returns the full FMU file path.
- Improved simulation time for FMUModelCS1´s simulate function. Use of the old simulate function is deprecated.
- Other improvements.

10.18. Release 1.8.6

- Support for Accelerator mode using the FMI blocks.

- Other improvements and bug fixes.

10.19. Release 1.8.5

- Fix for a memory leak.

10.20. Release 1.8.4

- Support enumerations for Co-Simulation and Model Exchange export.
- Unsupported data types for parameters does no longer give an error when exporting an FMU, now a warning is given and the parameter is not exposed by the FMU.
- Exported FMUs of Simulink models with bus outputs will use the signal labels of these buses to construct structured names for the outputs in the FMU.
- When importing FMUs in Simulink with structured names it is possible to use structured naming of the inputs/outputs. These inputs/outputs will then be buses with signal labels based on the structural naming.
- Model verification blocks and Stop Simulation blocks now causes exported Co-Simulation and Model Exchange FMUs to stop the simulation when these are triggered.

10.21. Release 1.8.3

- Sample and offset time for the Co-Simulation block can be set to symbols that are evaluated by Simulink at simulation time.
- Support for FMU Co-Simulation export from Simulink with Global (tunable) parameters.
- Other improvements and bug fixes.

10.22. Release 1.8.2

- When building simulink models with FMU blocks, the FMUs shared library can now be used.
- Updated getModelVariables method in the FMU classes. It now supports filtering on variable names.
- New method directDependency added to ScalarVariable1 class. It returns a list of all the direct dependencies defined by the FMU for a particular output variable.
- New factory function loadFMU. It creates an instance of one of the supported FMU classes.
- New FMU block script function added fmuGetInputPortsSimulink. It returns the input ports from an FMU block.

- New FMU block script function added fmuGetModelDataSimulink. It returns model data.
- New FMU block script function added fmuGetOptionSimulink. It returns an FMU block option.
- New FMU block script function added fmuGetOutputPortsSimulink. It returns the output ports from an FMU block.
- New FMU block script function added fmuGetValueSimulink. It returns the start value for a variable.
- New FMU block script function added fmuResetAllOutputPortsSimulink. It resets all output ports to default.
- New FMU block script function added fmuResetAllSimulink. It resets all parameter and start values, and all output ports.
- New FMU block script function added fmuSetOptionSimulink. It sets an FMU block option.
- New FMU block script function added fmuSetOutputPortsSimulink. It sets the output ports for an FMU block.
- Other improvements and bug fixes.

10.23. Release 1.8.1

- MATLAB interface now have an improved API. To get Alias base, units, displayUnits and multiple values are now supported.
- MATLAB interface now provide the functionality to forward log messages from FMU to a user provided log listener in MATLAB.
- MATLAB interface now have native and effective support for arrays of variables.

10.24. Release 1.8

- Simulink blocks from FMI Toolbox support dSPACE´s rti1006.tlc target.
- New FMU block script function added fmuResetAllValuesSimulink. It resets all parameter and start values.
- New FMU block script function added fmuResetValueSimulink. It resets one or multiple parameter and start values.
- New FMU block script function added fmuReloadFMUSimulink. It reloads an FMU block.
- New FMU block script function added fmuLoadFMUSimulink. It loads an FMU block with an FMU file.
- Changed name of the function **fmi_toolbox_lic_info** to **fmitoolbox_license**.

- Changed default value in the Model Exchange Simulink block. The Advanced setting **Use tolerance controlled FMU** is now disabled by default.
- Changed default value for the Simulink blocks. The Advanced setting **Add new output ports when the model is reloaded** is now disabled by default.
- FMU Model Exchange export target, fmu_me1, now supports linkage of existing S-function object files.
- Bug fixes.

10.25. Release 1.7.2

- Improved handling of event indicator functions for FMU export.
- Bug fix: Setting string parameter values triggered an error in Simulink.
- Other improvements and bug fixes.

10.26. Release 1.7.1

- Bug fix: Failed to link the FMU target object files properly.

10.27. Release 1.7

- Support for FMU Co-Simulation 1.0 export from Simulink.
- Simulink blocks are now supported by Simulink Coder.
- Mask parameters can now be used to set start values.
- Improved handling of the FMU blocks in a Simulink library.

  If the FMU block is located in a library(other than itself) the relative path options is now relative to this library.
- Many other improvements and bug fixes.

10.28. Release 1.6.1

- FMU export can now generate FMUs containing non-inlined S-functions(e.g DLL only).
- Improved direct dependency analysis in the FMU export.
- FMU import supports vector input ports.

- FMU import supports tool based Co-Simulation FMUs.

10.29. Release 1.6

- Support for static and dynamic analysis of FMUs through design-of-experiments (DoE) .
- General toolbox updates for better integration in MATLAB.
- New Windows installer for both 32- and 64-bit MATLAB.
- New Linux package for both 32- and 64-bit MATLAB.
- Unattended installation/uninstallation procedures are now supported.

10.30. Release 1.5

- Support for FMU Model Exchange 1.0 export from Simulink.

10.31. Release 1.4.6

- Bug fix: MATLAB interface function `getModelVariables` did not work for some MATLAB versions.

10.32. Release 1.4.5

- Updated the MATLAB interface.
- MATLAB interface now prints the log to file.

10.33. Release 1.4.4

- Bug fix: FMU file path was not saved properly causing the reload function to be triggered each time the Simulink model was opened.

10.34. Release 1.4.3

- Bug fix: FMU models containing no variables could not be properly loaded.

10.35. Release 1.4.2

- Bug fix: Enumerators could not be viewed correctly in the Simulink blocks.

- Bug fix: Minimum and Maximum values were truncated wrong in the Simulink block.

10.36. Release 1.4.1

- FMI Toolbox is built with FMI Library 2.0a2 to fix bugs.

10.37. Release 1.4

- FMI Toolbox is now based on the FMI Library from JModelica.org.
- Improved performance of the FMU parsing.
- New logger functionality.

10.38. Release 1.3.1

- New Flexera based license system.
- `fmuSetValueSimulink` now support setting vector values fast and convenient.
- User can now set how the FMU file should be located from the FMU block GUI.
- Bug fixes

10.39. Release 1.3

- FMI for Co-Simulation 1.0 is now supported.

  A restructuring of the installation folders is made to support different FMI versions in the future.
- A new block for Co-Simulation is added in Simulink, `FMU CS 1.0`.
- The Simulink block for Model Exchange changes name from `FMU` to `FMU ME 1.0`.
- New examples for Co-Simulation is added. Only for Windows.
- The FMI interface in Matlab is now based on classes. There are two new Matlab classes, one for FMI Co-Simulation 1.0 and one for FMI Model Exchange 1.0. The previous interface is replaced with these classes.

10.40. Release 1.2

- Parameter and start values can now be set to expressions in the GUI. These are evaluated just before the simulation starts.

- New function is added for setting parameter and start values of the FMU in the Simulink model from a MATLAB script.
- Replaced **Block Icon** tab with an **Advanced** tab. In the **Advanced** tab it is now possible to enable/disable the logger function in the FMU, setting tolerances in the FMU, use the block icon from the FMU if there is any, decide if the block name should be updated with the FMU name, if new output ports should be added when the FMU is reloaded.
- It is now possible to save big models.
- Major GUI update, removing bugs.

10.41. Release 1.1

- Now supports Windows 32-bit, Windows 64-bit, Linux 32-bit and Linux 64-bit.
- A new tutorial, Vehicle Tutorial, demonstrating a car model being controlled from a driver implemented in Simulink.
- Fixed bug when terminating from fmiInitialize caused segfault in Simulink.
- Fixed bug where adding output ports did not work on some systems.
- The Simulink block does not reload the FMU anymore when the Simulink model is opened. This caused for instance the output ports to be reset.

10.42. Release 1.0

Initial release:

- Simulation of compiled Modelica models, FMUs, in Simulink.
- Simulation of compiled Modelica model, FMUs, in Matlab scripts.
- Support for fixed step solvers in Simulink
- Graphical user interface for configuration of parameters and outputs in Simulink.
- Generation of result files compliant with Dymola.

Bibliography

[Jak2003] Johan Åkesson. *Operator Interaction and Optimization in Control Systems*. ISRN LUTFD2/ TFRT--3234--SE. Lund University. Sweden. 2003.