Dymola

Dynamic Modeling Laboratory

FMI Support in Dymola

Contents: Section “FMI Support in Dymola” from Chapter 6 “Other Simulation Environments” from the manual “Dymola User Manual Volume 2”.

September 2014

The information in this document is subject to change without notice.

Dassault Systèmes AB
Ideon Science Park
SE-223 70 Lund
Sweden

E-mail: http://www.3ds.com/support
URL: http://www.Dymola.com
Phone: +46 46 2862500

Contents

1 FMI Support in Dymola

This document is an extract from the FMI section in Dymola User Manual Volume 2, Chapter 6, “Other Simulation Environments”.

That chapter describes how to interface models created in Dymola to other simulation environments. Here only the following extract is covered:

- Support for the Functional Mockup Interface (FMI)
    - Import and export of FMI models in Dymola
    - Validating FMUs from Dymola
    - Export of FMI models from Matlab/Simulink

1.1 FMI Support in Dymola

1.1.1 Introduction

FMI

The FMI (“Functional Mock-up Interface”) standard allows any modeling tool to generate C code or binaries representing a dynamic system model which may then be seamlessly integrated in another modeling and simulation environment.

FMI started as a key development effort within the MODELISAR project, see

https://itea3.org/project/modelisar.html

The FMI standard is today maintained and developed as a long-term project within the Modelica Association.

Three official FMI specifications have been released. The 'FMI for Model Exchange' specification version 1.0 was released on January 28, 2010, and the 'FMI for Co-Simulation' specification version 1.0 was released on October 12, 2010. FMI 2.0 which merges the model exchange and co-simulation specifications into one document was published on July 25, 2014.

The model exchange specifications focus on the model ODE interface, whereas the co-simulation specifications deal with models with built-in solvers and coupling of simulation tools. All specification documents can be downloaded here

http://www.fmi-standard.org/

The specification documents are also available in Dymola using the command **Help > Documentation**. The specifications are separated into an execution part (C header files) and a model description part (XML schema). A separate model description is used in order to keep the executable footprint small. Both FMI 1.0 specifications use essentially the same XML schema (a couple of capability flags are introduced for FMI for Co-Simulation).

In summary, an FMU (Functional Mock-up Unit) implementing an FMI specification consists of

- The XML model description.
- Implementation of the C function interface in binary and/or source code format.
- Resources such as input data.
- Image and documentation of the model.

FMI support in Dymola

The Dymola FMI support consists of the two built-in functions described below for FMU export and import, respectively. Commands are also available in the Dymola user interface to execute these functions.

The first three items in the list above are currently supported by Dymola. FMI (both Model Exchange and Co-Simulation) is supported for Windows and Linux.

Unless otherwise stated, features are available both for FMI version 1.0 and version 2.0.

Note that the FMI 2.0 Beta 4 and FMI 2.0 RC1specifications are no longer supported.

For the latest information about limitations and supported features of FMI, please visit www.Dymola.com/FMI.

Online tunable parameters

Online tunable parameters are supported in FMI version 2.0 (tunable parameters were not allowed in FMI version 1.0).

1.1.2 Exporting FMUs from Dymola

FMU export by the built-in function translateModelFMU

Exporting FMU models from Dymola is achieved by the function

```
translateModelFMU(modelToOpen, storeResult, modelName,
fmiVersion, fmiType, includeSource)
```

The input string `modelToOpen` defines the model to open in the same way as the traditional `translateModel` command in Dymola.

The Boolean input `storeResult` is used to specify if the FMU should generate a result file (`dsres.mat`). If `storeResult` is true, the result is saved in `<model id>.mat` when the FMU is imported and simulated, where `<model id>` is given at FMU initialization. (If empty, "`dsres`" is used instead.) This is useful when importing FMUs with parameter `allVariables = false`, since it provides a way to still obtain the result for all variables. Simultaneous use of result storing and source code inclusion (see below) is not supported.

The input string `modelName` is used to select the FMU model identifier. If the string is empty, the model identifier will be the name of the model, adapted to the syntax of the model identifier (e.g. dots will be exchanged with underscores).The name must only contain letters, digits and underscores. It must not begin with a digit.

The input string `fmiVersion` controls the FMI version ("`1`" or "`2`") of the FMU. The default is "`1`".

The input string `fmiType` defines whether the model should be exported as

- Model exchange (`fmiType="me"`)
- Co-simulation using Cvode (`fmiType="cs"`),
- Both model exchange, and Co-simulation using Cvode (`fmiType="all"`)
- Co-simulation using Dymola solvers (`fmiType="csSolver"`).

The default setting is `fmiType="all"`. This parameter primarely affects modelDescription.xml. For the three first choices binary and source code always contains both model exchange and Co-simulation. Note – Co-simulation using Dymola solvers is currently only available on Windows,, and the Binary Model Export license is required. For this choice the binary code only contains Co-simulation; the solver and tolerance that is selected in Dymola is also used by the exported FMU. Please see also "Notes on Co-Simulation" on page 13 concerning Co-simulation

The Boolean input `includeSource` is used to specify if source code should be included in the FMU. The default setting is that it is not included (`includeSource=false`). Simultaneous use of result storing (see above) and source code inclusion is not supported. Note that source code generation is not supported for Co-simulations using Dymola solvers.

The function outputs a string `FMUName` containing the FMU model identifier on success, otherwise an empty string.

As an example, translating the Modelica CoupledClutches demo model to an FMU with result file generation, is accomplished by the function call

```
translateModelFMU("Modelica.Mechanics.Rotational.Examples.
CoupledClutches", true);
```

After successful translation, the generated FMU (with file extension .fmu) will be located in the current directory. The user can select if 32-bit and/or 64-bit FMU binaries should be generated – see the FMI tab description below.

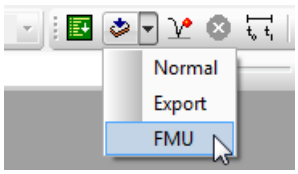
The generated FMU contains information about if it has been generated without export options. In the corresponding XML file of such an FMU, the following is seen:

```
generationTool="Dymola Version 2015 (64-bit), 2014-02-21
(requires license to execute)"
```

FMUs exported from Dymola support intermediate results for event update (fmiEventUpdate) for Model Exchange for FMI version 1.0.

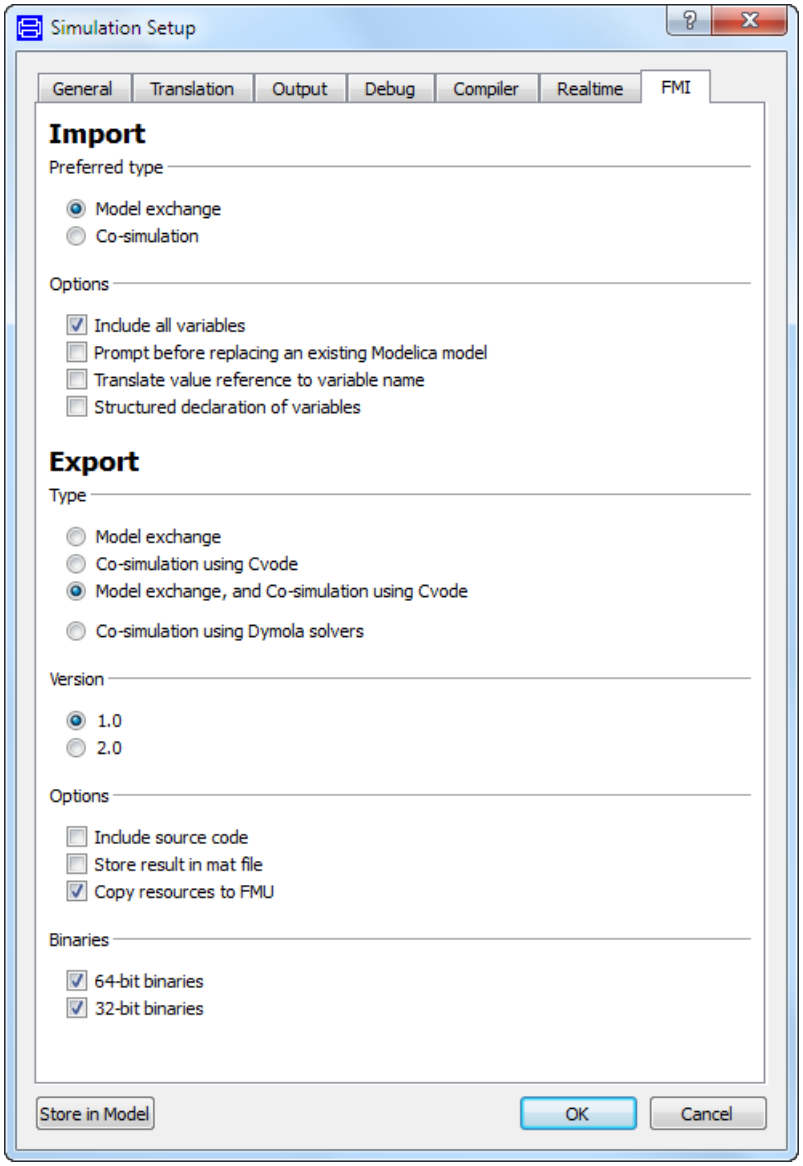
Commands in Dymola for FMU export

An alternative to executing the `translateModelFMU` function from the command line is to use the **FMU** option of the **Translate** button as illustrated below.



The above is also available as the command **Simulate > Translate > FMU**.

What settings will be used when using any of the above commands is specified in the export part of the **FMI** tab of the simulation setup, reached by the command **Simulate > Setup…**, the **FMI** tab:



FMI type can be selected as **Model exchange**, **Co-simulation using Cvode**, **Model exchange, and Co-simulation using Cvode** or **Co-simulation using Dymola solvers**; this setting corresponds to the parameter `fmiType` in `translateModelFMU` (see description above of this setting for more information).

The FMI version can be selected as "`1`" or "`2`", the default being "`1`".

Three general options are available; see the description above of the corresponding parameters for more information concerning the first two. Note that the two first ones cannot be ticked simultaneously.

- **Include source code** – corresponds to the parameter `includeSource` in `translateModelFMU`. If ticked (`includeSource=true`) source code is included, if unticked the source code is not included. Note that for Co-simulation, source code export is currently only supported for the CVODE solver.
- **Store result in mat file** – corresponds to the parameter `storeResult` in `translateModelFMU`. If ticked (`storeResult=true`) a result file is generated and stored as a .mat file `<model id>.mat`, if unticked no result file is generated.
- **Copy resources to FMU** – external resources using the functions `ModelicaServices.ExternalReferences.loadResource` or `Modelica.Utilities.Files.loadResource` are by default copied to the FMU. The resulting FMU will be larger due to this. If this is not wanted, de-selecting the setting will not copy the resources to FMU, but the resource-paths using Windows-shares will be changed to UNC-paths when possible. This makes the FMU usable within a company – without increasing its size. An example of using the resource copying is given below, the extended example in the "String parameter support" section.

Finally, the user can select whether 32- and/or 64-bit FMU binaries should be generated. This option is not available in `translateModelFMU`. Note that even if the option **64-bit binaries** is selected, no such binaries are created unless 64-bit compilation is enabled. In a 32-bit version of Dymola, this can be enabled by setting the flag `Advanced.CompileWith64=2`.

FMU black-box export

It is possible to export FMUs as black-box models, where only top-level inputs and outputs are included in the XML model description. This can be used to export sensitive models without exposing the names of parameters and internal variables. This export is activated by setting the flag

```
Advanced.FMI.BlackBoxModelDescription = true;
```

Including settings in the exported FMU

Note the possibility to include settings in the exported FMU by ticking **Settings included in translated model**, reachable by the command **Simulation > Setup...**, the **Debug** tab. (If such settings are included in a Dymola-generated FMU, they can be logged by activating `fmi_loggingOn` in the FMI tab of the parameter dialog of the imported and instantiated FMU.)

String parameter support

String parameters are supported in FMUs (except for FMI 1.0 FMUs exported with Dymola solvers). For the FMU export to support string parameters, the following flag must be set:

```
Advanced.AllowStringParameters=true
```

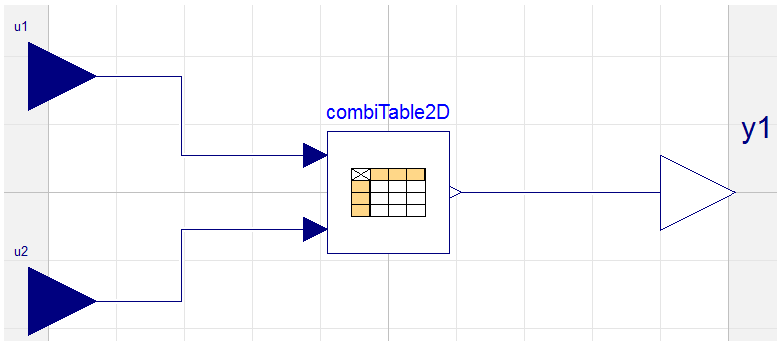
The flag is by default `false`.

(String variables are however presently not supported.)

Example

String parameter support can be illustrated by a simple example of changing tables for an FMU; consider creating a simple model for linearization.

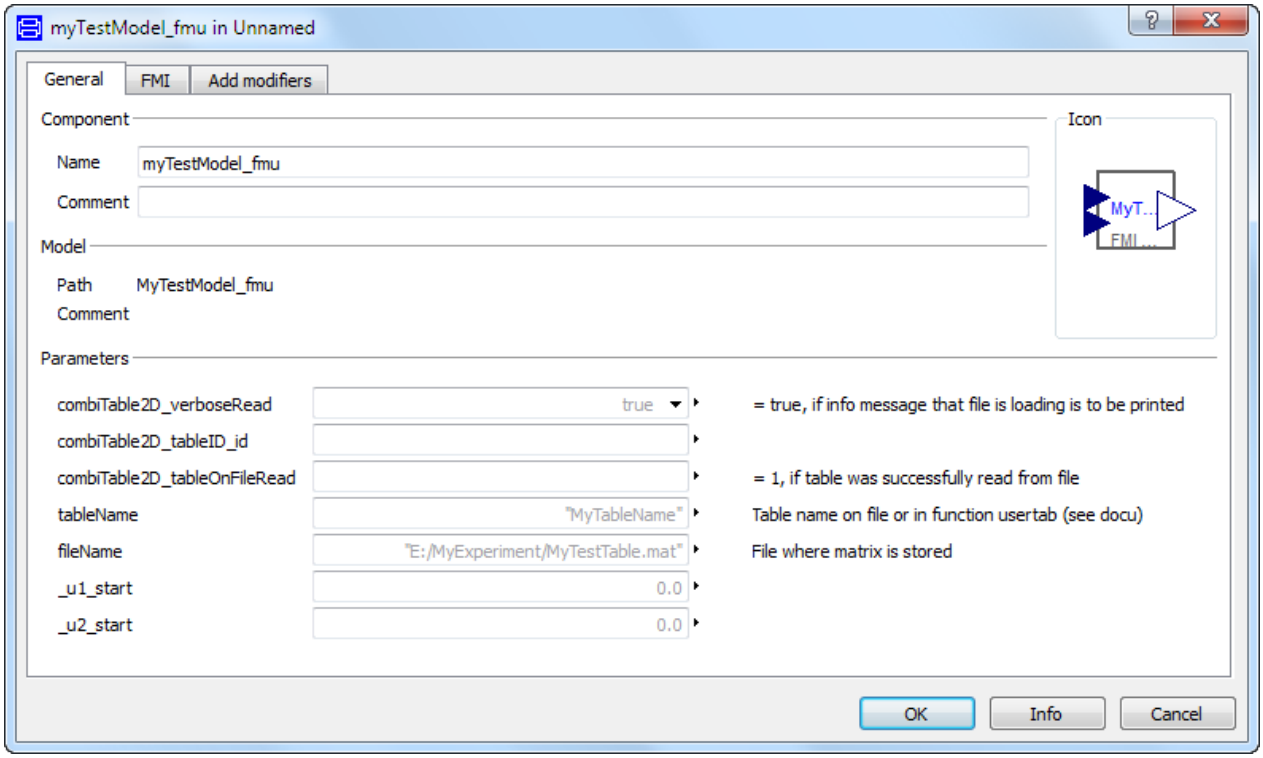
Create a model; drag an instance of `Modelica.Blocks.Tables.CombiTable2D` into the model. Connect the two inputs and the output and create the corresponding connectors. The result is:



In the parameter dialog of combiTable2D, select **tableOnFile** to true, and propagate **tableName** and **fileName**. Give relevant default values for them. As an example, looking at the resulting Modelica code when having specified a table name and file name as default value, we find:

```
model MyTestModel

  parameter String tableName="MyTableName"
    "Table name on file or in function usertab (see docu)";
  parameter String fileName="E:/MyExperiment/MyTestTable.mat"
    "File where matrix is stored";
equation

  a
end MyTestModel;
```

Saving the model, and then generating an FMU from it (do not forget to set the flag above), we can import this FMU and look at the resulting parameter dialog of an instance of that FMU:



This FMU supports changing the table name and file name as string parameters.

Extended example (resource handling)

If the FMU should contain the table as a resource, the following can be done (Note – the used resource handling in this example is currently only supported on Windows.):

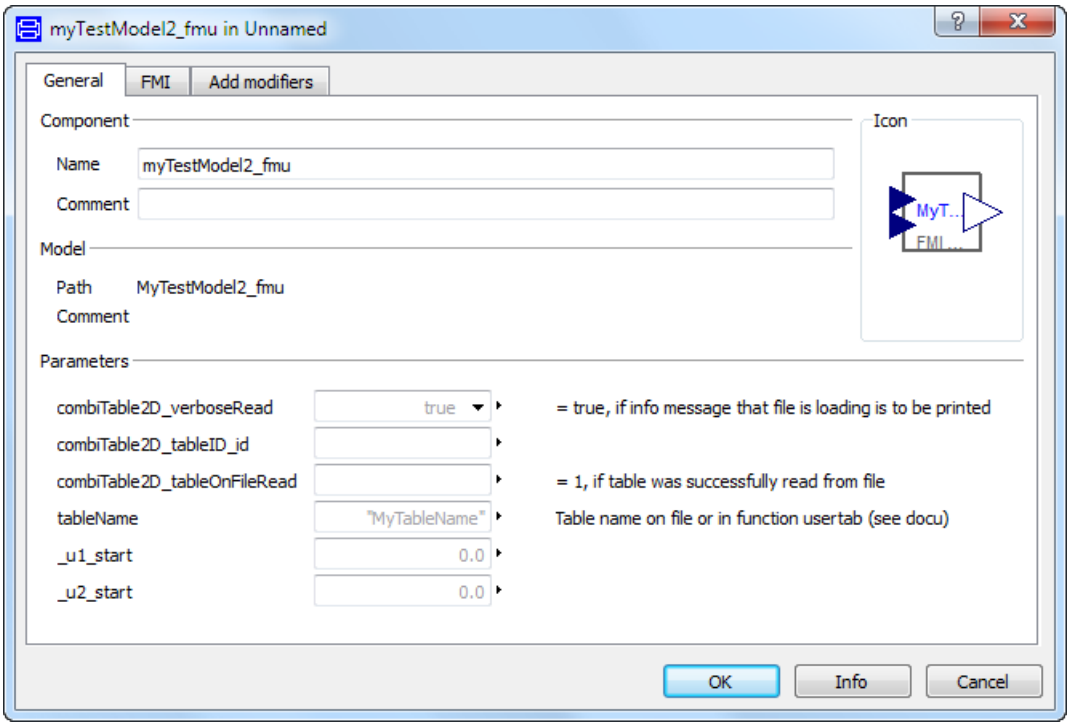
Rename the parameter **fileName** to **includeFileInFMU** (really not needed, but for clarity). Use, in the variable definition dialog of **includeFileInFMU**, in the default value input field, the context command **Insert Function Call...** to access `Modelica.Utilities.Files.loadResources`, and specify the file name. The resulting code is (given a new model MyTestModel2 is created):

```
model MyTestModel2

  parameter String tableName="MyTableName"
    "Table name on file or in function usertab (see docu)";

  parameter String includeFileInFMU=Modelica.Utilities.Files.loadResource("E:/MyExperiment/MyTestTable.mat")
    "File where matrix is stored";
equation

  a
end MyTestModel2;
```

Save the model. Before generating the FMU, check:

- that `Advanced.AllowStringParameters=true`.
- that **Copy resources to FMU** is ticked in the **FMI** tab of the simulation setup.

We can import the generated FMU and look at the resulting parameter dialog of an instance of that FMU:



The **includeFileInFMU** parameter is not displayed, it is evaluated, and the corresponding file has been copied to the Resources directory of the FMU.

Handling multiple FMUs

In Dymola 2014 and later an extra source code file `all.c` is provided; it includes all other C files. This file is needed to compile all FMUs source code as one unit, which in turn is required because the demand that all internal functions and symbols needs to be static to be able to combine several source code FMUs.

The only disadvantage compiling this file instead of the separate C files, is that any modification in any source code file requires re-compilation of everything.

Notes on Co-Simulation

Note that all Dymola solvers are supported on Windows for FMU Co-simulation export (if the Binary Model Export license is available); however, the CVODE solver can be selected

as a particular solver. The support for features is currently larger when selecting CVODE as a particular solver than for Dymola solvers:

- Including source code is currently only supported for the CVODE solver.
- String parameters are currently only supported for the CVODE solver.
- For Linux, only the CVODE solver is currently supported for FMU Co-simulation.

CVODE solver

The SUNDIALS suite of numerical solvers (version 2.4.0) can be used in the co-simulation FMUs. The SUNDIALS CVODE solver with Backward Differentiation Formula (BDF) and Newton iteration can be used as solver in the exported co-simulation FMUs. For further details, visit

https://computation.llnl.gov/casc/sundials/main.html

Fixed-step embedded (inline) solvers for FMU Co-Simulation export

The Dymola inline integration solvers are supported also for FMU Co-Simulation export. Note that the fixed step-size used for the inline integration should also be used as step-size when calling the `fmiDoStep` routine of the generated FMU.

For source code export it is also required to set the flag

```
#define ONLY_INCLUDE_INLINE_INTEGRATION
```

in the header file `conf.h`.

Support for optional FMI Export options

Support for optional FMI Export options in FMI 2.0

The following tables list Dymola support for optional export options in FMI 2.0. Since both “True” and “False” can be a limitation, the cells are color coded: green means “underlying feature supported in Dymola”, yellow means “underlying feature not supported in Dymola”. Furthermore, capital letters are used for “underlying feature supported”.

The order of the features is the order they appear in the specification.

Optional FMI 2.0 features	Model Exchange	Model Exchange with inline integration	Co-simulation using Cvode	Co-simulaton with inline integration	Co-simulation using Dymola solvers
needsExecutionTool	FALSE	FALSE	FALSE	FALSE	FALSE
completedIntegratorStepNotNeeded	false	false	NA	NA	NA
canBeInstantiatedOnlyOncePerProcess	true	true	true	true	true
canNotUseMemoryManagementFunctions	FALSE	FALSE	FALSE	FALSE	true
canGetAndSetFMUState	TRUE	TRUE	TRUE	TRUE	false
canSerializeFMUState	false	false	false	false	false
providesDirectionalDerivative	TRUE	TRUE	TRUE	TRUE	false
canHandleVariableCommunicationStepSize	NA	NA	TRUE	false	TRUE
canInterpolateInputs	NA	NA	TRUE	false	false
maxOutputDerivativeOrder	NA	NA	1	0	0
canRunAsynchronuously	NA	NA	false	false	false

Support for optional FMI Export options in FMI 1.0

Optional FMI 1.0 Co-simulation features	Co-simulation using Cvode	Co-simulation with inline integration	Co-simulation using Dymola solvers
canHandleVariableCommunicationStepSize	YES	false	YES
canHandleEvents	YES	YES	YES
canRejectSteps	false	false	false
canInterpolateInputs	YES	false	false
maxOutputDerivativeOrder	1	0	0
canRunAsynchronuously	false	false	false
canSignalEvents	false	false	false
canBeInstantiatedOnlyOncePerProcess	true	true	true
canNotUseMemoryManagementFunctions	FALSE	FALSE	true

FMU export on Linux

The FMU export on Linux requires the Linux utility "zip". If not already installed, please install using your packaging manager (e. g. apt-get) or see e.g. http://www.info-zip.org.

Export of 32-bit and 64-bit FMUs is supported on Linux.

Using Dymola compilers for FMI Co-simulation export is not supported on Linux.

Limitations

- The value `meUndefinedValueReference` is never returned when value references are requested. As a consequence, some value references returned may not be present in the model description file.
- The result file generation is currently only fully supported for the traditional solvers (Lsodar, Dassl, Euler, Rkfix2, Rkfix3, and Rkfix4) when importing the FMU in Dymola. For the other solvers, the number of result points stored will typically be too low. However, the values are accurate for the time-points at which they are computed.
- String variables cannot be used in models which are exported as FMUs. (String parameters are however supported (except for FMUs exported with Dymola solvers).)
- When imported by a target simulator, the generated FMU does currently not support inclusion of several instances.
- For Linux, only the CVODE solver is currently supported for FMU Co-simulation.

1.1.3 Importing FMUs in Dymola

The Dymola FMU import consists of (1) unzipping the .fmu archive, (2) transforming the XML model description into Modelica, and (3) opening the resulting Modelica model in Dymola.

Importing FMU models to Dymola is achieved by the function

```
importFMU(fileName, includeAllVariables, integrate,
promptReplacement, packageName)
```

The input string `fileName` is the FMU file (*with* the .fmu extension).

By setting the variable `includeAllVariables` to false, only inputs, outputs and parameters from the model description are included when generating the Modelica model. Such black-box import can be used as separate compilation of models to substantially reduce translation times. For large model this is recommended since the generated .mo file otherwise becomes huge and will take long time for Dymola to parse and instantiate.

The parameter `integrate` controls if integration is done centralized or in the FMU, i.e. `integrate=true` means import the Model Exchange part of the FMU and `integrate=false` means use the Co-Simulation part of the FMU. By default this parameter is true. This setting is only relevant if the FMU to import supports both types. Otherwise this setting is silently ignored. If the Co-Simulation part is used, the macro step-size can be set as the parameter `fmi_CommunicationStepSize` in the FMI tab of the

parameter dialog of the imported FMU. See also section “Settings of the imported FMU” on page 20.

The parameter `promptReplacement` can be set to true to generate prompting before replacement of any existing Modelica model being the result of a previous import. Having no prompting is useful when repeatedly importing FMUs using scripting. By default this parameter is false.

The string parameter `packageName` can be set to the package to where the FMU should be imported. The package must be open in Dymola when importing.

The function outputs true if successful, false otherwise.

The generated Modelica file will get the name `model_fmu.mo` or `model_fmu_black_box.mo`, depending on the value of `includeAllVariables`.

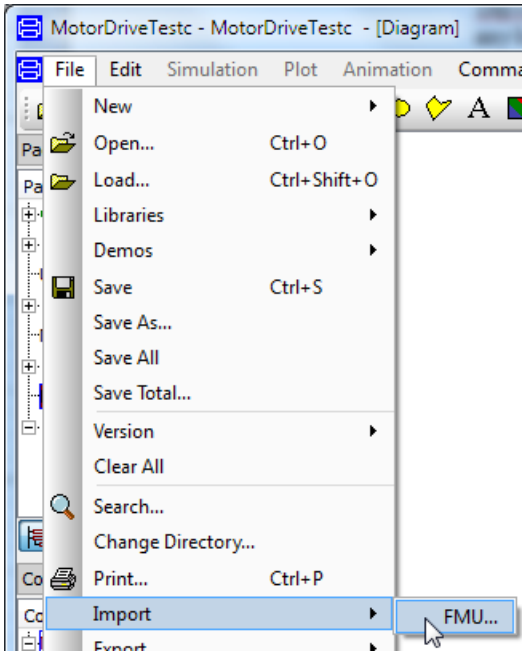
For the 64-bit version of Dymola, it is required to set the flag `Advanced.CompileWith64=2` when compiling applications (dymosim.exe) that contain imported 64-bit FMUs.

ASCII characters of values larger than 32 are supported in the xml file of the imported FMU. Also UTF characters are supported, but not recommended.

Note: The binary library files from any previous import are replaced when calling `importFMU` and thus translations of previously imported models are not guaranteed to work any longer (in the unlikely event of a name clash).

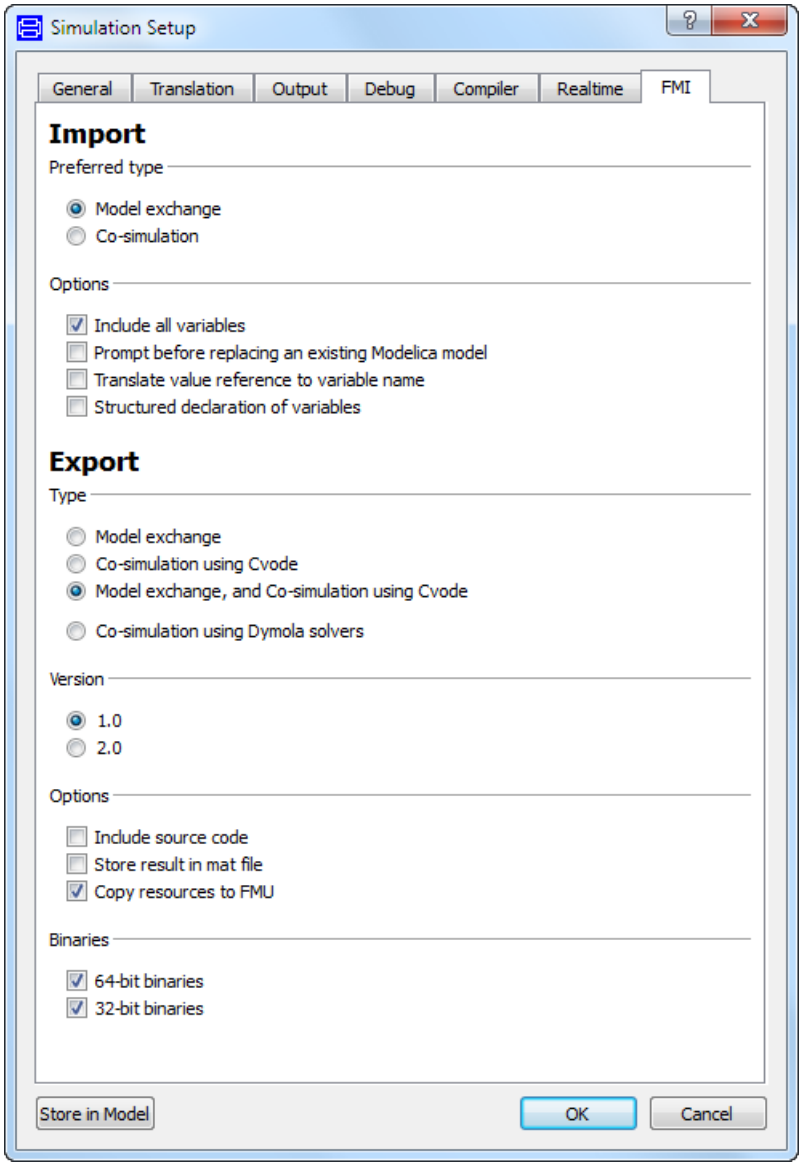
Commands in Dymola for FMU import

An alternative to executing the `importFMU` function from the command line is to use the command **File > Import > FMU...**.



Note that this command also will be automatically applied on an .fmu file by dragging it into the Dymola main window.

What settings will be used when using any of the above commands is specified in the import part of the **FMI** tab of the simulation setup, reached by the command **Simulate > Setup…**, the **FMI** tab:



Preferred type can be selected as **Model exchange** or **Co-simulation**. This setting is only relevant if the FMU to import supports both types. Otherwise this setting is silently ignored. This setting corresponds to the parameter `integrate` in `importFMU` (see above for description).

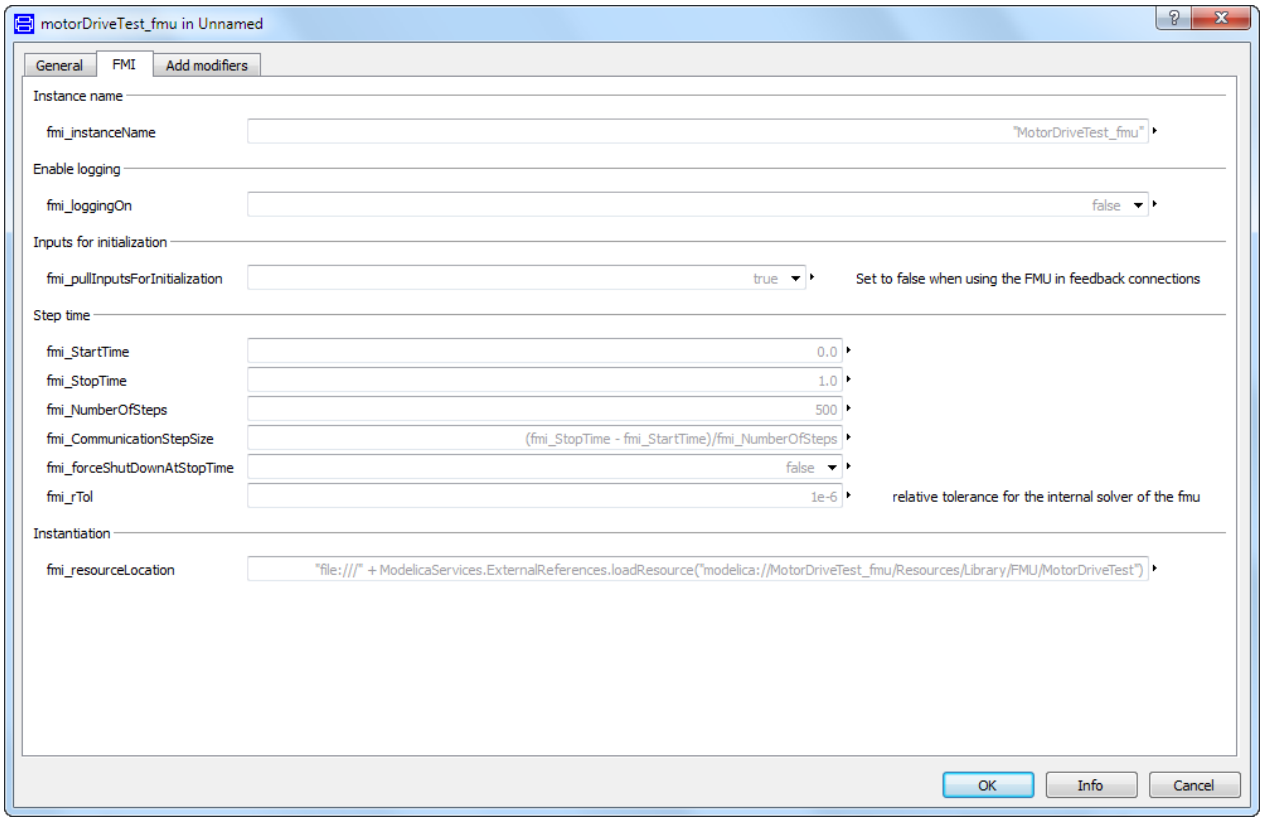
Four options are available:

- **Include all variables** – corresponds to the function parameter `incudeAllVariables` (see above).

- **Prompt before replacing an existing Modelica model** – corresponds to the function parameter `promptReplacement` (see above).
- **Translate value reference to variable name** – this option is not present in `importFMU`. If ticked, the imported FMU will contain a translation from value references to variable names. This is useful for debugging, however will decrease the performance.
- **Structured declaration of variables** – this option is not present in `importFMU`. If ticked, the variables of the imported FMU will be presented in a hierarchical structure, that is, as records. This is useful when e.g. wanting to change variable values. To be able to use this option, the attribute `variableNamingConvention` in the model description file of the FMU to be imported must be set to `variableNamingConvention="structured"`.

Settings of the imported FMU

The parameter dialog of the imported and instantiated FMU contains an FMI tab:



The available settings depend on the FMU type.

fmi_resourceLocation might be needed when importing FMUs from other vendors, to specify the location of external resources. (For FMI version 1.0 Co-simulation the name is

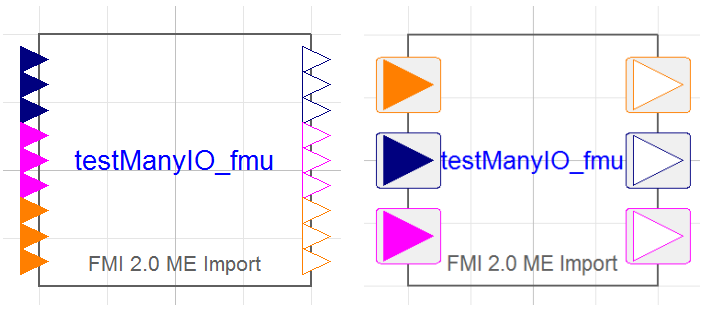
fmi_fmuLocation.) By default the parameter displays the location where the FMU is unpacked, which is usually the location of external resources (dlls, tables, etc,) as well

Handling of start values for initialization

Start values can be set for FMI Model Exchange, to set input start values before initialization. This can be useful when wanting to avoid e.g. division by zero when initializing.

Importing FMUs with many inputs/outputs

When importing FMUs with many inputs/outputs, the input and output connectors of the imported FMU are automatically stacked at the same location, one location for each type (Integer, Real, and Boolean) of input and output connectors (the image to the right below).



The limit of the number of connectors when stacking should be applied is defined by the flag

```
Advanced.FMI.OverlappingIOThreshold
```

The default value of the flag is 10 (so for creating the figure above, the value was set to 4).

Dragging a connection from/to a stacked connector displays a dialog to conveniently select what connectors to connect. See previous chapter for details.

FMU import on Linux

The FMU import on Linux requires the Linux utility “unzip”. If not already installed, please install using your packaging manager (e. g. apt-get) or see e.g. http://www.info-zip.org.

Limitations

- For FMI version 1.0, the attribute nominal for scalar variables is not supported when importing FMUs with Model Exchange. (For FMI version 2.0, this is supported.)

1.1.4 Validating FMUs from Dymola

Once the dynamic behavior of a model is verified and it is ready to be exported as FMU, one would like to verify that this behavior can be repeated on the targeted simulation environment. For model exchange, which is dependent on the solver of the target, this is

naturally less straight-forward than for co-simulation, where the solver is built into the FMU. We focus this discussion on the co-simulation case, although all is possible for model exchange as well.

Normally, the FMU contains inputs that need to be connected to signal generators (sources) before this validation can be commenced. Since this is model and test dependent and hard to automate, we will assume the model inputs have been connected to necessary sources beforehand. The result is a test model with no disconnected inputs. After the validation, these sources are of course removed before the final FMU is created.

Since Dymola supports FMU import, it becomes natural to re-import the FMU in Dymola and compare its simulation with the original model. We demonstrate this for the demo model CoupledClutches. For brevity, we use a scripting perspective. First, export as FMU with, say, both model exchange and co-simulation support:

```
translateModelFMU(
  "Modelica.Mechanics.Rotational.Examples.CoupledClutches",
  false, "", "1", "all");
```

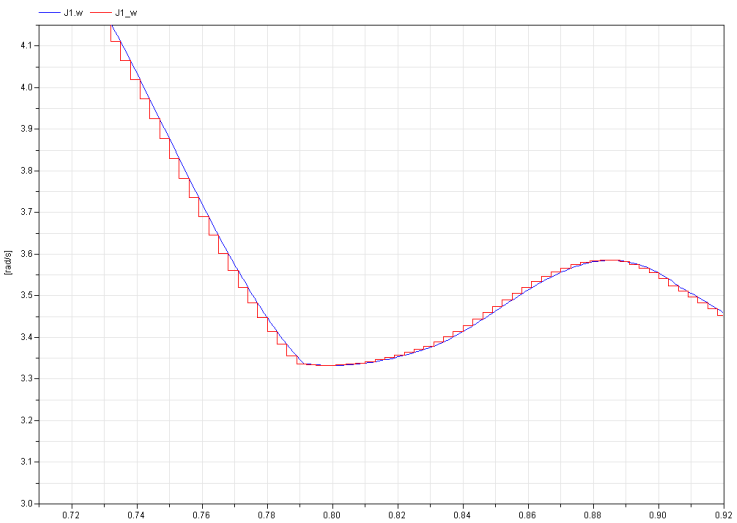
Re-import, in a non-interactive mode, the FMU for co-simulation:

```
importFMU(
  "Modelica_Mechanics_Rotational_Examples_CoupledClutches.fmu",
  true, false, false);
```

Simulate the model being the result of the import:

```
simulateModel(
  "Modelica_Mechanics_Rotational_Examples_CoupledClutches_fmu",
  stopTime=1.5, method="dassl");
```

Finally, the resulting trajectories can be plotted and compared visually with the original (non-FMU) simulation. Note that, since the imported model is flattened, the trajectory names are somewhat different; e.g. J1.w becomes J1_w:



The blue trajectory is from the reference simulation and the red is from the co-simulation. Note that the latter is rendered as constant between the sample points.

While this validation is ok for sample testing of a single model, this clearly becomes infeasible for systematic validation of several trajectories.

The remedy is a new function `validateModelAsFMU`, which automates the following steps:

- Generation of reference trajectories.
- Exporting of the FMU.
- Importing of the FMU.
- Mapping of trajectories names to those of the original model.
- Numeric comparison of trajectories.
- Graphical HTML presentation of deviating trajectories in fashion similar to the plot above.

Main features include:

- Using a default set of trajectories to compare or specifying it explicitly. The default it the set of all state candidates.
- Choosing tolerance for the comparison.
- Optional generation of reference trajectories which is typically only needed once.
- Optional FMU export which might not be needed each time.
- Test of co-simulation or model exchange.
- Test of FMI version 1.0 or 2.0.

It is available in Modelica\Library under the Dymola installation.

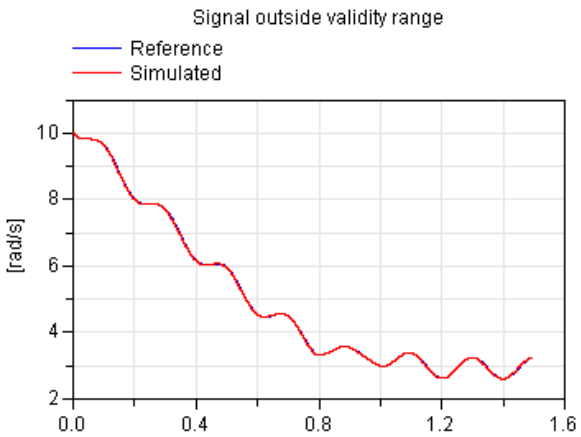
Note that the ModelManagement license is needed to use this feature.

Below call validates CoupledClutches as a co-simulation FMU for FMI 1.0:

```
validateModelAsFMU(
  "Modelica.Mechanics.Rotational.Examples.CoupledClutches");
```

An excerpt from the log file is given below:

```
Variable: J1.w has scalar criteria 0.00248651 larger than tolerance 0.001
```



In this case we may argue that the comparison tolerance should be increased to avoid the report of this trajectory.

1.1.5 Importing Simulink models using FMI

Introduction

The Dymola 2013 and later distributions include a package to support export of FMUs from MATLAB/Simulink. This package contains an implementation of the FMI 1.0 model exchange specification on top of model code generated by Real-Time Workshop (RTW). The MATLAB Target Language Compiler (TLC) is used to construct the XML model description.

This package for Simulink FMU export together with the Dymola support for FMU import facilitates simulation of Simulink models in Dymola.

Important!

This package does currently not support FMI version 2.0.

The project builds on the RTW 'S-function Target' configuration that is available in MATLAB. In fact, the same model C code is generated by the new 'S-function target with FMI' as for the RTW S-function target. In addition, the FMI target performs the following

- Constructs the model description interface, `modelDescription.xml`, from the `<modelname>.rtw` model description
- Compiles the RTW-generated model code and the S-function FMI wrapper, `fmiModelFunctions.c`, and links with required libraries
- Constructs the FMI zip archive according to the specified structure

Release History

- Version 1.0, February 10, 2010
    - First version
- Version 1.1, August 20, 2010
    - Supporting MATLAB R2010a
    - Support for S-function blocks written in C
- Version 1.2, June 1, 2012
    - MATLAB support up to R2011b
    - Support for Visual Studio 2010
    - 64-bit support
- Version 1.2.1, March 4, 2013
    - Compliant to FMU Checker ver. 1.0.2

Files

The package (`rtwsfcnfmi.zip`) is located in the `$DYMOLA/mfiles/` directory, but since it is independent of Dymola it may be extracted to any location. The archive consists of four sub-directories and the included files are described briefly below.

rtwsfcnfmi\bin

Pre-compiled 32-bit binaries of the FMI implementation for the supported Visual Studio compilers and MATLAB releases.

rtwsfcnfmi\bin64

Pre-compiled 64-bit binaries of the FMI implementation for the supported Visual Studio compilers and MATLAB releases.

rtwsfcnfmi\c

This directory holds C source files to include and compile the RTW-generated model code. The standard FMI 1.0 header files (`fmiModelTypes.h` and `fmiModelFunctions.h`) are also located in this directory.

rtwsfcnfmi\m

This directory contains MATLAB help files called from the TLC scripts. These are used to construct the date, GUID, and value reference attributes used in the XML model description.

rtwsfcnfmi\tlc

The TLC scripts used for code generation and for constructing FMI-specific files are included in this directory. The template makefile and compiler-dependent settings can also be found here.

Installation

Follow these steps to set up the environment in MATLAB

- Define the environment variable SFCN_FMI_ROOT as the directory where you have extracted the files, for example

  ```
  SFCN_FMI_ROOT = c:\Program Files\rtwsfcnfmi
  ```

  using Control Panel / System / Advanced / Environment Variables

- Add the rtwsfcnfmi sub-directories SFCN_FMI_ROOT\m and SFCN_FMI_ROOT\tlc to your MATLAB path and execute the function fmiSetBin. This can be done by adding the following to your MATLAB startup script

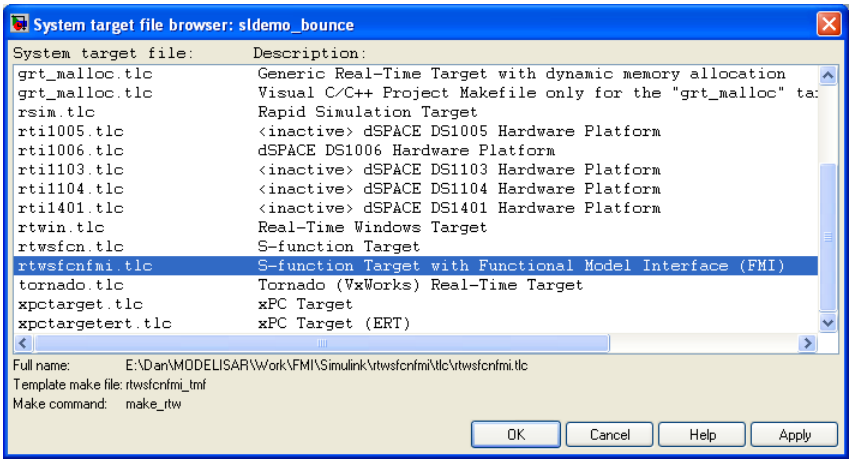
  ```
  addpath([getenv('SFCN_FMI_ROOT') '\m']);
  addpath([getenv('SFCN_FMI_ROOT') '\tlc']);
  fmiSetBin
  ```

- Open the m-file `makeDate.m` and modify the timediff variable to correspond to your time zone (relative GMT).
- The default supported compiler for 64-bit MATLAB installations is Visual Studio 8.0. Using Visual Studio 9.0 requires changing the value of `FMI_VS` (from 8 to 9) on line 205 of SFCN_FMI_ROOT\tlc\rtwsfcnfmi_vcx64.tmf.

Creating an S-function FMU

This section describes the procedure to export an FMU from Simulink

- If the Simulink model to be exported as an FMU should be integrated (connected) with other components, you need to add external input and/or output ports to your model. These can be found in the Sinks and Sources categories of the Simulink browser.



- In the Simulink 'Configuration Parameters' dialog, choose the 'Real-time Workshop' tab and click Browse to select a different System Target File. Select rtwsfcnfmi.tlc in the list (see figure above). Three FMI-specific options are then available in the Real-Time Workshop tab:
    - FMI general options
        - *Build FMI binary as DLL* (checked as default, uncheck to build the FMU as a static library).
        - *Zip utility* (path to zip utility used to build the FMI archive, default is 7-Zip).
        - *Zip options* (command line options for the zip utility).
    - FMI XML options
        - *Model author* (used to specify the model author).
        - *Use prefixes on variable names* (used to select if variable category prefixes should be used) Note: Having this box unchecked may cause compilation errors due to name conflicts!
        - *Include global block outputs* (used to select if block outputs should be included).
        - *Include discrete states (DWork)* (used to select if discrete states should be included).
- Start the Real-time Workshop build process by pressing **Ctrl-B**.

The build process will compile the RTW-generated model code using the FMI Simulink wrapper, `fmiModelFunctions.c`, and link with the required MATLAB and system libraries to create the FMI binary. The build process will also create the FMI XML model description, `modelDescription.xml`, and construct the FMI archive, `<modelname>_sf.fmu` in the RTW build directory.

The FMI binary is built using the same compiler as used when building the S-function MEX file. This is changed in MATLAB using the command

```
>> mex -setup
```

Limitations and Trouble-Shooting

This version of the package supports 32- and 64-bit MATLAB releases from R2007b to R2011b.

Supported compilers

- Microsoft Visual Studio 8.0 (2005)
    - R2007b to R2011b
- Microsoft Visual Studio 9.0 (2008)
    - R2008b to R2011b
- Microsoft Visual Studio 10.0 (2010) Note: Only 32-bit support
    - R2010b to R2011b

The package is designed and tested only for model code generated when having a variable-step solver configured in Simulink. However, it is likely to work also with fixed-step solvers assuming the tasking mode is set to SingleTasking. Note that this limitation is related to the RTW-generated code, since no solver is embedded in the FMU.

The model description contains inputs, outputs, parameters, continuous states, state derivatives, global block outputs, and discrete states (stored in S-function DWork vectors). Other block-internal variables, such as discrete modes, are not included.

The model description currently uses the RTW identifiers as model variable names, thus removing the hierarchical structure in the generated model. Also, in some cases, these identifiers will not be traceable back to original model components.

Since RTW only generates unique variable identifiers within each variable category, prefixes (in, out, pm, st, bo, and dw) are added to the variable names. The use of prefixes can be switched off, see FMI XML options above.

Only real data types are currently supported. Stateflow models and other models with discrete logic may require that you override data types with doubles. Choose 'Fixed-Point Settings' (or ‘Fixed-Point’ -> ‘Fixed-Point tool’ depending on MATLAB version) in the 'Tools' menu and then specify 'True doubles' in the 'Data type override' drop-down list.

The code generation optimization ‘Enable local block outputs’ may cause the FMU compilation to fail. For these models it is necessary to switch off this optimization when generating the FMU.

Unsupported Simulink Blocks

- The blocks ‘Discontinuities/Backlash’ and ‘Discontinuities/Rate Limiter’ currently only work correctly if used with a discrete sample time.

- Simulink models with 'MATLAB Fcn' blocks are not supported.
- Only Simulink standard blocks, the ‘LTI System’ block from Control Systems Toolbox, and the Stateflow blocks have currently been tested.

2 Index

M

R

S

X