FMI Support in Dymola

Contents: Chapter 1 “FMI Support in Dymola” extracted from the manual “Dymola User Manual 2B: Simulation Interfaces and Export”.

March 2020 (Dymola 2021)
## Contents

1  FMI Support in Dymola ............................................................................................................. 5  
   1.1  FMI Support in Dymola ........................................................................................................ 5  
      1.1.1  Introduction ................................................................................................................... 5  
      1.1.2  Exporting FMUs from Dymola .................................................................................... 7  
      1.1.3  Importing FMUs in Dymola ......................................................................................... 20  
      1.1.4  Validating FMUs from Dymola .................................................................................. 35  
      1.1.5  Profiling of FMUs ........................................................................................................ 37  
      1.1.6  FMI Kit for Simulink ................................................................................................... 38  
2  Index ....................................................................................................................................... 39
1 FMI Support in Dymola

This chapter describes the support for the Functional Mockup Interface (FMI):
- Introduction (page 5)
- Exporting FMUs from Dymola (page 7)
- Importing FMUs to Dymola (page 20)
- Validating FMUs from Dymola (page 35)
- Profiling of FMUs (page 37)
- FMI Kit for Simulink (page 38)

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
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. A model package implementing the FMI standard is called a Functional Mockup Unit (FMU). For further details, visit:

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.

For the latest information about limitations and supported features of FMI, please visit [www.Dymola.com/FMI](http://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

\[
\text{translateModelFMU}(\text{modelToOpen}, \text{storeResult}, \text{modelName}, \\
\text{fmiVersion}, \text{fmiType}, \text{includeSource}, \text{includeImage})
\]

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

Note that, from Dymola 2020x, the function by default uses any experiment (simulation) setting of the model (that is, settings in the **General** tab of the simulation setup; for example, start time, stop time, and algorithm), to make it easier to script these settings. If this is not wanted, you can set the flag `Advanced.FMI.UseExperimentSettings=false`. (The flag is by default `true`.)

The Boolean input `storeResult` is used to specify if the FMU should generate a result file. 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. The result generation interval when storing simulation results can be controlled by the flag `Advanced.FMI.FMUStoreResultInterval`. The default value of the flag is `0.001`. 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 primarily affects `modelDescription.xml`. For the three first choices binary and source code always contains both model exchange and Co-simulation. For the last choice the binary code only contains Co-simulation; the solver and tolerance that are selected in Dymola in the general tab in the simulation setup are also used by the exported FMU. Note that co-simulation using Dymola solvers requires the Binary Model Export license. Please see also “Notes on Co-Simulation” on page 17 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.

Notes:

- Source code generation is not supported for Co-simulations using Dymola solvers.
- General source code documentation is available in the Documentation folder inside the generated FMU folder.
- You can use sparse Jacobians when exporting FMUs with source code included. See “Using sparse Jacobians when exporting FMUs with source code included” on page 13.
- From Dymola 2020, when exporting source code FMUs using external c-functions the source code of those functions are included, specifically:
  - The contents of `includeDirectory` will be included under `sources/include`
  - The contents of `__Dymola_SourceDirectory` (default “…/Resources/Source”) will be included under `sources/source`.

The Integer input `includeImage` is used to specify if an image should be included in the export. The default is 0, meaning no image is exported. 1 means including the icon of the model in the export, 2 means including the image of the diagram layer of the model in the export. The image is stored in the FMU resources with the name `model.png` which actually means it can be exchanged with any image `model.png`.

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

```
```

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 command `Simulation > Translate > FMU` as illustrated below:
The settings that will be used when using any of the above commands is specified in a dialog that appears when the command has been given:

This dialog corresponds to the **FMI Export** tab of the simulation setup (reached by the command **Simulation > Setup**, the **FMI Export** tab), except for the last setting Model identifier:
Changing settings when exporting will impact also this menu. Changed settings are remembered in the session, but not between sessions.

Note the information message that appears in the lower part of the menu if the setting **Evaluate parameters to reduce models (improves simulation speed)** is not enabled in the **Translation** tab of the simulation setup. (Reached by the command **Simulation > Setup…**, the **Translation** tab.)

**Type group**

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). There is also a corresponding flag
Advanced.FMI.FMUFMIType with the same alternatives, except the default of the flag being an empty string.

**Version group**

The FMI version can be selected as "1" or "2", the default being "1". The setting corresponds to the parameter fmiVersion in translateModelFMU. There is also a corresponding flag Advanced.FMI.FMUFIVersion with the same alternatives, except the default of the flag being an empty string.

**Binaries group**

The user can select whether 32- and/or 64-bit FMU binaries should be generated. This option is not available in translateModelFMU.

**Model description filters group**

You can control the filtering of the modelDescription.xml file with these settings:

- **Protected variables** (by default activated) filters away protected Modelica variables. This setting corresponds to the flag Advanced.FMI.xmlIgnoreProtected = true;

- **Auxiliary variables** (by default not activated) works differently in FMI version 2.0 and FMI version 1.0:
  - For FMI version 2.0 activating this setting means filtering away all variables of causality local, except states and derivatives of states.
  - For FMI version 1.0 activating this setting means all variables of causality internal except the ones with variability parameters are filtered away.

  This setting corresponds to the flag Advanced.FMI.xmlIgnoreLocal = false;

- **Black-box** (by default not activated) works differently in FMI version 2.0 and FMI version 1.0:
  - For FMI version 2.0 activating this setting means filtering away all variables except the following:
    - Variables of causality inputs and outputs
    - Variables needed for the model structure. The names are however hidden (concealed).
  - For FMI version 1.0 activating this setting means filtering away all variables except variables of causality inputs and outputs.

  This setting corresponds to the flag Advanced.FMI.BlackBoxModelDescription = false;

Black-box export can be used to export sensitive models without exposing the names of parameters and internal variables.

Note that if you activate Black-box, the settings Protected variables and Auxiliary variables are dimmed; they are not relevant in this case.
**Options group**

Four options are available. 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 (default). For more details, see the description of the parameter in the section about `translateModelFMU` above. Note that there is a corresponding flag `Advanced.FMI.FMUIncludeSource` with the same alternatives as the parameter `includeSource`, the default is true.

- **Export with string parameters** – enables using string parameters when exporting FMUs. All types of FMUs are supported. Notes:
  - The default value is `false`, since you would normally not want the regular simulation to use string parameters as this makes the code slightly less efficient.
  - The default setting corresponds to the flag `Advanced.AllowStringParametersForFMU = false`.
  - String variables are currently not supported.
  - For examples, see “String parameter support - examples” starting on page 14.

- **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 - examples” section. (There is a corresponding flag `Advanced.FMI.CopyExternalResources` with the default value `true`.)

- **Show dialog when exporting** – this option is by default ticked. If unticked, the Export FMU dialog is not displayed when exporting FMUs.

**Store result group**

- **Store result in mat file** – corresponds to the parameter `storeResult` in `translateModelFMU`. If enabled (`storeResult=true`) a result file is generated and stored as a `.mat` file `<model id>.mat`, if not enabled, no result file is generated. For more details, see the description of the parameter in the section about `translateModelFMU` above. There is a corresponding flag `Advanced.FMI.FMUIncludeSource` with the default value `false`.

- **Interval** – If the above setting is enabled, the result generation interval when storing simulation results can be controlled here. The default value is 0.001. There is a corresponding flag `Advanced.FMI.FMUStoreResultInterval` with the same default.
**Model image group**

You can select to include a model image in the FMU export. The following alternatives are possible:

- **None** (default) No image is included in the FMU export.
- **Icon** The icon of the model is included in the FMU export.
- **Diagram** The image of the diagram layer of the model is included in the FMU export.

If an FMU with an image included in the export is imported, instances of this imported FMU will display this image. See “Using the exported model image to display the imported FMU” on page 27.

The alternatives above corresponds to parameter `includeImage` in `translateModelFMU`, with the values 0, 1, and 2, respectively. There is a corresponding flag `Advanced.FMI.ImageSetting` that with the same values.

The image is stored in the FMU resources with the name `model.png`. (Note that this also means that you can use any image `model.png` to replace it.)

**Model identifier group**

This group only appears in the dialog when exporting an FMU, not in the general FMI Export tab.

**Model identifier** User-defined model identifier of the FMU. The default value is the name of the model to export, including the path. (Two images above, it is the Coupled Clutches demo.) The setting corresponds to the parameter `modelName` in the built-in function `translateModelFMU`. The setting also decides the exported FMU file name.

**Exporting a model in DAE mode**

Exporting a model in DAE mode as an FMU is supported if the selected solver supports DAE mode. If this is not the case, the export is not allowed – an error message will be displayed.

**Using sparse Jacobians when exporting FMUs with source code included**

FMU source code export includes code for building the Cvode solver. Cvode can be enhanced with sparse solver capabilities. If the flag `Advanced.SparseActivate` is set to true before the FMU export, additional files are copied to the “sources” subdirectory of the generated FMU directory.

The details on how to enable the code for sparse solvers are found in the documentation provided in the FMU directory, in the `index.html` file, located in the generated FMU. (To display this directory, use any zip tool to reveal the structure of the generated FMU.)

**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.

**Option to include “save total” of the model in the generated FMU**

It is possible to include a “save total” of the model in the generated FMU by setting the flag

```
Advanced.FMI.IncludeSaveTotal=true
```

The default value of the flag is `false`.

**String parameter support - examples**

String parameters are supported in FMUs if the option **Export with string parameters** is selected (see the setting above).

**Basic 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:
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:

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):

```plaintext
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

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:

![Parameter Dialog](image)

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

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.

Multiple instantiation of the same FMU

FMUs generated by Dymola 2016 and later support multiple instantiation. This means that the same FMU can be used several times in the same model.

The generated XML file indicates that the model can be instantiated multiple times.

Restrictions:

- Multiple instances are currently only supported for Co-simulation with Cvode, see next section.
- The support for multiple instances has a runtime cost, you can for this reason disable the support for multiple instances by setting the flag Advanced.AllowMultipleInstances=false. (This flag is by default true.)
- The old table handling, corresponding to tables in previous versions for Modelica Standard Library (3.2 or older) is not supported. If you have user models with such old table handling, those must be updated to use this feature.

Notes on Co-Simulation

Note that all Dymola solvers are supported for FMU Co-simulation export (if the Binary Model Export license is available); however, the CVODE solver can be selected as a particular solver by any export type selection containing Co-simulation using Cvode. The support for features is currently larger when selecting CVODE as a particular solver this way than when selecting Co-simulation using Dymola solvers:

- Including source code is currently only supported when selecting Co-simulation using Cvode.
- Multiple instances are currently only supported when selecting Co-simulation using Cvode.

CVODE solver

The SUNDIALS suite of numerical solvers (version 2.6.2) 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. See next page; the tables are on the same page for comparison reasons.
<table>
<thead>
<tr>
<th>Optional FMI 2.0 features</th>
<th>Model Exchange</th>
<th>Model Exchange with inline integration</th>
<th>Co-simulation using Cvode</th>
<th>Co-simulation with inline integration</th>
<th>Co-simulation using Dymola solvers</th>
</tr>
</thead>
<tbody>
<tr>
<td>needsExecutionTool</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>completedIntegratorStepNotNeeded</td>
<td>false</td>
<td>false</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>canBeInstantiatedOnlyOncePerProcess</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>true</td>
</tr>
<tr>
<td>canNotUseMemoryManagementFunctions</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>true</td>
</tr>
<tr>
<td>canGetAndSetFMUState</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>Partly¹</td>
</tr>
<tr>
<td>canSerializeFMUState</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>Partly²</td>
</tr>
<tr>
<td>providesDirectionalDerivative</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>canHandleVariableCommunicationStepSize</td>
<td>NA</td>
<td>NA</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>canInterpolateInputs</td>
<td>NA</td>
<td>NA</td>
<td>TRUE</td>
<td>false</td>
<td>TRUE</td>
</tr>
<tr>
<td>maxOutputDerivativeOrder</td>
<td>NA</td>
<td>NA</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>canRunAsynchronously</td>
<td>NA</td>
<td>NA</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

1 Supported except for the Dymola solver Cvode.
2 Supported except for the Dymola solver Cvode.

---

<table>
<thead>
<tr>
<th>Optional FMI 1.0 Co-simulation features</th>
<th>Co-simulation using Cvode</th>
<th>Co-simulation with inline integration</th>
<th>Co-simulation using Dymola solvers</th>
</tr>
</thead>
<tbody>
<tr>
<td>canHandleVariableCommunicationStepSize</td>
<td>YES</td>
<td>false</td>
<td>YES</td>
</tr>
<tr>
<td>canHandleEvents</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>canRejectSteps</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>canInterpolateInputs</td>
<td>YES</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>maxOutputDerivativeOrder</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>canRunAsynchronously</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>canSignalEvents</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>canBeInstantiatedOnlyOncePerProcess</td>
<td>FALSE</td>
<td>FALSE</td>
<td>true</td>
</tr>
<tr>
<td>canNotUseMemoryManagementFunctions</td>
<td>FALSE</td>
<td>FALSE</td>
<td>true</td>
</tr>
</tbody>
</table>

---

1 Supported except for the Dymola solver Cvode.
2 Supported except for the Dymola solver Cvode.
Propagating annotations from Modelica variables to the FMI model description

Dymola supports propagating annotations from Modelica variables to the fmi2Annotation node “Annotations” in the corresponding scalar variables in an FMI 2.0 modelDescription.xml document.

To activate this feature, set the flag

```
Advanced.FMI2.OutputVariableAnnotationsInXML = true;
```

The flag is by default false.

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](http://www.info-zip.org).

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.

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 25.

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.

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 > Open > 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 a dialog that appears when applying any of the commands:
Except the FMU file section, this dialog corresponds to the FMI Import tab of the simulation setup, reached by the command Simulation > Setup, the FMI Import tab:
Changing settings when exporting will impact also this menu. Changed settings are remembered in the session, but not between sessions.

The FMU file part of the dialog that appears when applying a command can be used to browse for the FMU. When the FMU is dragged into Dymola, the path is prefilled.

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). There is a corresponding flag `Advanced.FMI.Integrate` that has the same alternatives as the `integrate` parameter.

Five options are available:
• **Include all variables** – corresponds to the function parameter `includeAllVariables` (see above). There is a corresponding flag `Advanced.FMI.IncludeAllVariables` that has the same alternatives as `includeAllVariables`.

• **Prompt before replacing an existing Modelica model** – corresponds to the function parameter `promptReplacement` (see above). There is a corresponding flag `Advanced.FMI.PromptReplacement` that has the same alternatives as `promptReplacement`.

• **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. There is a corresponding flag `Advanced.FMI.TranslatedLoggedValueReferences` with the default value `false`.

• **Structured declaration of variables** – this option is not present in `importFMU`. If ticked, (the default value) 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"`. There is a corresponding flag `Advanced.FMI.structuralDeclaration` with the default value `true`.

• **Show dialog when importing** – this option is by default ticked. If unticked, the Import FMU dialog is not displayed when importing FMUs.

### Settings of the imported FMU

The parameter dialog of the imported and instantiated FMU contains an **Input Start Values** tab and an **FMI** tab.

#### Input Start Values tab

For FMI Model Exchange in FMI version 2.0, input start values can be set before initialization. This should however only be necessary if your FMU is constructed in such a way that the default start values for an input is illegal in the FMU, e.g. division with an input variable having a default value of zero. For such an input variable you can set the input start value to some value not being zero; sources of the FMU will then be handled properly in the `initializationMode`.

Such start values are collected in the **Input Start Values** tab:
For FMUs of FMI version 1.0, you should avoid a design where input values affect initialization, since the FMI 1.0 interface lacks proper support to iterate during initialization.
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.

**Using the exported model image to display the imported FMU**

If the exported FMU includes an exported model image model.png, this image is used to represent the FMU when importing it. Below, the demo Coupled Clutches has been exported with the diagram layer as the model image. The result importing it and instantiating it is:
If no model image is available in the imported FMU, a standard icon is used instead:
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.
Import of FMUs of FMI version 1.0 and version 2.0 to the same model

Import of both FMUs of FMI version 1.0 and version 2.0 to the same model, is supported.

Input handling for co-simulation FMU import

Input time point

You can now choose the time point used as input when calling doStep with a co-simulation FMU from time $t_0 \rightarrow t_1$.

Your choices are input at time $t_0$ input at time $t_1$ or pre on input at time $t_1$ (default and behavior of Dymola 2017).

Input time can be chosen with the parameter fmi_Input time, and if StepEnd is chosen fmi_UsePreOnInputSignals can be used to disable pre operator on the input signal.

Be aware that using StepStart at for fmi_inputTime will introduce delays in output if you have direct dependencies on the input.

Using StepEnd and disabling pre can introduce algebraic loops when connecting with feedback which cannot be solved by co-simulation FMUs.

Using pre on inputs at StepEnd will break these loops if you create them with connections by introducing an infinite small delay.

Input derivatives

Dymola supports interpolation of input and to set the input derivatives of real inputs if the FMU has the capability flag canInterpolateInputs.

The interpolation is a first order interpolation. This can be activated by setting the flag Advanced.FMI.SetInputDerivatives = true before importing an FMU that supports this feature.

Since we only support interpolation and not extrapolation, similar restrictions exist as when using StepEnd as input time and disabling pre, i.e. all FMUs in a feedback loop cannot have this feature at the same time.

Improved fmi2 initialization for co-simulation

FMI 2.0 co-simulation supports initialization with algebraic loops (but not solving algebraic loops at simulation time).

Improvement has been made to make this more robust at Dymola import, this has also removed the need for the parameter fmi_pullInputsForInitialization and thus it been removed, (it is still needed for some cases in FMI 1.0 co-simulation as you cannot solve initialization of multiple FMUs there).
Translation of underscore

The default (in Dymola 2017 and later) is to translate underscore “_” without any changes when importing an FMU. If you want underscore “_” to be translated to “_0” when you import an FMU, you can set the flag

\[
\text{Advanced.FMI.UseTrueVariableNames} = \text{false};
\]

Previously the default value was to always translate underscore “_” to “_0” because of possible conflicting names (the period “.” in Modelica paths is always translated to underscore when importing an FMU). Now, when structured variables are used by default when importing an FMU, the likeliness for conflicting names is very small, hence the changed default behavior, and the flag to revert to the old behavior.

Note! If you have a model that contains an FMU as a connected component, you might get errors if you want to reimport the FMU to the model, due to the changed translation of underscore. You need in such a case to either redo your connections or set the above flag to false to have the old naming convention when reimporting the FMU:

Display of information for an imported FMU

Information from the modelDescription.xml file of an imported FMU is displayed in the information layer of the imported FMU (two images presented below to be able to display the whole information layer):
Unit handling

FMI version 2.0 supports unit handling where an FMU exporter can define any unit for inputs and outputs as long as conversion to base units according to the FMI standard is available. This allows for proper unit checking for inputs and outputs between FMUs.

Dymola supports this; units are automatically converted to base units for inputs and outputs of imported FMUs. Such unit handling for parameters in FMUs is also supported.

The unit conversion can be disabled by setting the flag

```plaintext
Advanced.FMI.DoNotDeclareUnits = true;
```

Setting this flag means ignoring the unit declarations completely. The flag is by default false.

Refresh of FMUs

FMU components can be refreshed by using the FMU context command **Re-import FMU**:
The is no scripting support for this functionality.

**Check if Windows binaries are available when importing an FMU in Windows**

When you import an FMU in Windows that contains no Windows binaries, you will get a warning.

The warning when importing by the command **File > Open > Import FMU**... or dragging the FMU into the Dymola window:
The warning in the Command log when importing the FMU by scripting:

```
importFMU("E:/Dymola work/Dymola 2021/Empty.fmu", true, true, false, "");
```

**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](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 straightforward 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(

```
false, "", "l", "all");

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

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

Simulate the model being the result of the import:

```plaintext
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:

![Plot of trajectories](image)

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 is 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.

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

```java
```

An excerpt from the log file is given below:

```
Variable: J1.w has scalar criteria 0.0014851 larger than tolerance 0.001
```

![Graph](image)

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

### 1.1.5 Profiling of FMUs

The feature of profiling of function calls can be used to investigate FMUs. For more information of profiling of function calls, see the manual “Dymola User Manual 1B: Developing and Simulating a Model”, chapter “Simulating a Model”, section “Improving simulation efficiency”, subsection “Profiling”.
1.1.6 **FMI Kit for Simulink**

FMI Kit for Simulink supports export of FMUs from Matlab/Simulink as well as import of FMUs into Matlab/Simulink.

The FMI Kit for Simulink is from Dymola 2020x not included in the Dymola distribution, you can instead access it freely from GitHub (including all source code):

[https://github.com/CATIA-Systems/FMIKit-Simulink](https://github.com/CATIA-Systems/FMIKit-Simulink)

The support for the FMI Kit for Simulink is handled through the issue system at GitHub.
2 Index

B
black-box import using FMI, 20

C
Co-simulation
FMI for Co-simulation, 17

E
exporting models using FMI, 7
exporting models with built-in numerical solvers, 17

F
FMI, 5
exporting models, 7
FMI Kit for Simulink, 38
for Co-simulation, 17
importing black-box models, 20
importing models, 20
model exchange, 5

specification for Co-simulation, 6
specification for model exchange, 6
validating FMUs, 35
XML model description, 6
FMU, 6
black-box export, 11
exporting a model in DAE mode as FMU, 13
exporting FMU’s, 7
exporting FMUs with settings, 13
exporting FMUs with source code using sparse
Jacobians, 13
FMU export from Simulink, 38
generate Dymola result file in .mat format, 7
importing FMU’s, 20
importing FMUs with many inputs/outputs, 29
including 'save total' of the model in the generated FMU,
14
model identifier, 13
multiple FMU’s, 17
multiple instantiation of the same FMU, 17
online tunable parameters, 6
profiling, 37
refresh FMU, 33
re-import, 33
string parameters, 14
validating FMUs, 35
Functional Mock-up Unit. See FMU

I
importing models using FMI, 20
black-box models, 20

M
model description
XML model description for FMI, 6
model exchange using FMI, 5
model identifier. See FMU : model identifier MODELISAR, 5

P
profiling of FMUs, 37

R
refresh FMU, 33
re-import FMU, 33

S
Simulink
FMI Kit for Simulink, 38
FMU export from Simulink, 38
specification
FMI for Co-simulation, 6
FMI for model exchange, 6
SUNDIALS suite of numerical solvers, 17

X
XML
model description for FMI, 6