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”.

March 2017 (Dymola 2018)

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

Dassault Systèmes AB
Ideon Gateway
Scheelevägen 27 – Floor 9
SE-223 63 Lund
Sweden

E-mail: http://www.3ds.com/support
URL: http://www.Dymola.com
Phone: +46 46 270 67 00

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):

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. 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.

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 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 16 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. Note also that general source code documentation is available in the Documentation folder inside the generated FMU folder.

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.

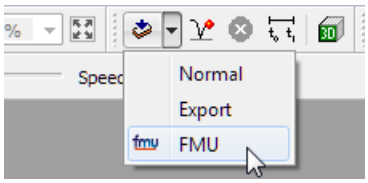
The generated FMU contains information about if it has been generated without export options. In the corresponding XML file of such an FMU, the following is seen:

```
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.

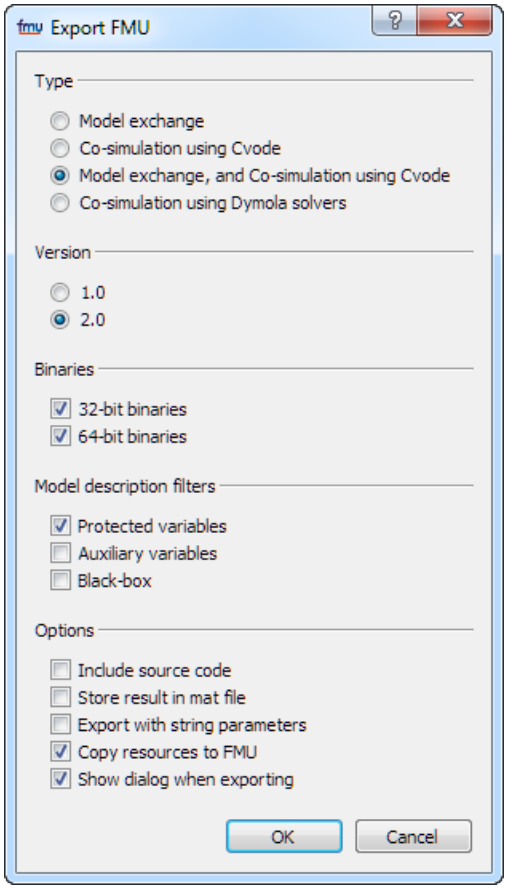
Commands in Dymola for FMU export

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

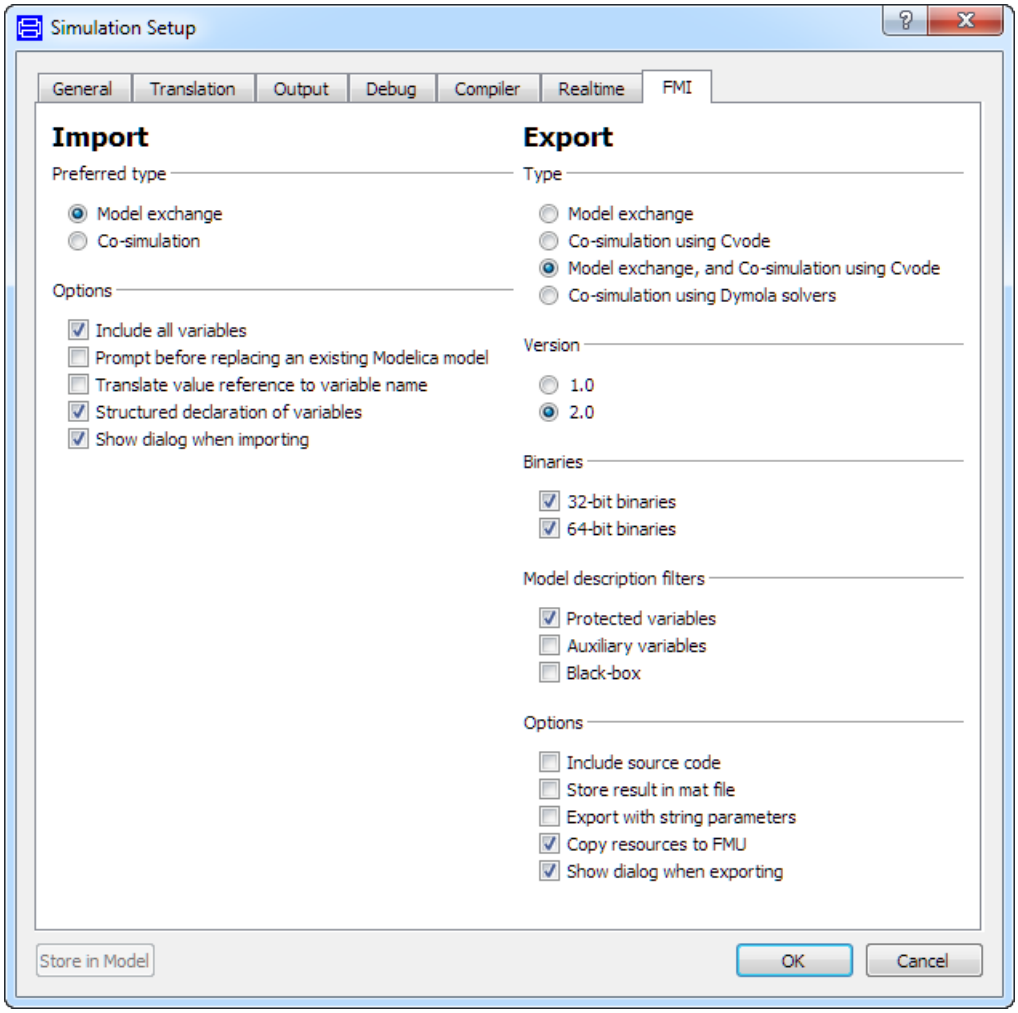


The above is also available as the command **Simulate > Translate > FMU**.

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 export part of the **FMI** tab of the simulation setup, reached by the command **Simulate > Setup…**, the **FMI** tab:



Changing settings when exporting will impact also this menu. Changed settings are remembered in the session, but not between sessions.

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).

Version group

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

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 is this case.

Options group

Five 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. Note also that general source code

documentation is available in the Documentation folder inside the generated FMU folder.

- **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.
- **Export with string parameters** – enables using string parameters when exporting FMUs. All types of FMUs are supported. ***Note:***
    - 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.
- **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.
- **Show dialog when exporting** – this option is by default ticked. If unticked, the Export FMU dialog is not displayed when exporting FMUs.

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.)

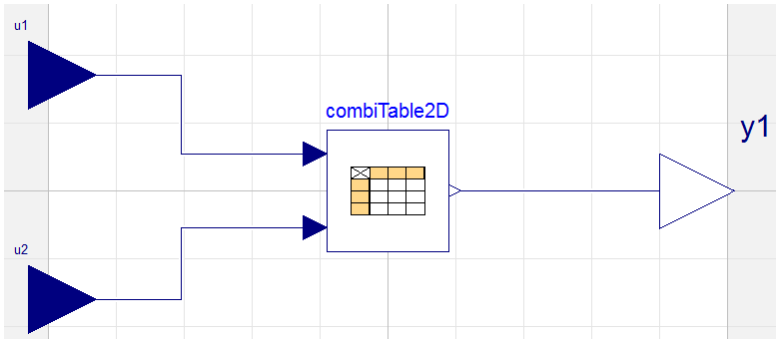
String parameter support - examples

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.

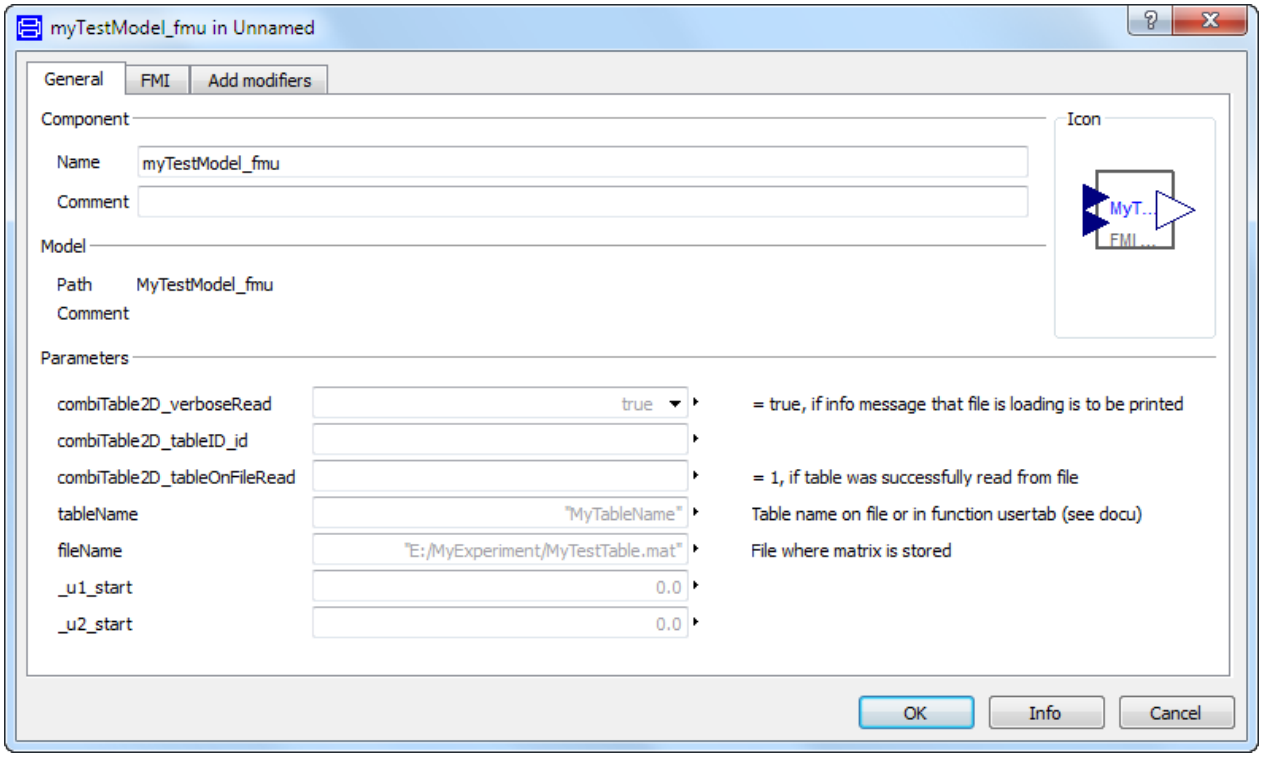
Create a model; drag an instance of `Modelica.Blocks.Tables.CombiTable2D` into the model. Connect the two inputs and the output and create the corresponding connectors. The result is:



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:

```
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:

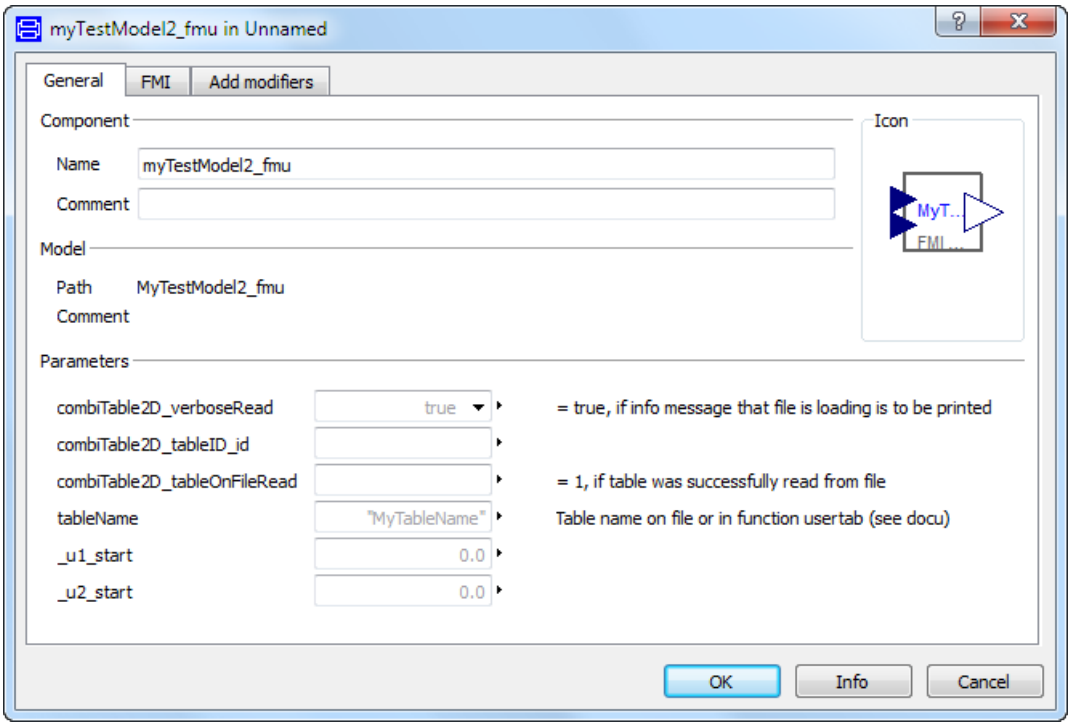
Rename the parameter **fileName** to **includeFileInFMU** (really not needed, but for clarity). Use, in the variable definition dialog of **includeFileInFMU**, in the default value input field, the context command **Insert Function Call...** to access `Modelica.Utilities.Files.loadResources`, and specify 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

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.

Optional FMI 2.0 features	Model Exchange	Model Exchange with inline integration	Co-simulation using Cvode	Co-simulation with inline integration	Co-simulation using Dymola solvers
needsExecutionTool	FALSE	FALSE	FALSE	FALSE	FALSE
completedIntegratorStepNotNeeded	false	false	NA	NA	NA
canBeInstantiatedOnlyOncePerProcess	FALSE	FALSE	FALSE	FALSE	true
canNotUseMemoryManagementFunctions	FALSE	FALSE	FALSE	FALSE	true
canGetAndSetFMUState	TRUE	TRUE	TRUE	TRUE	Partly[1]
canSerializeFMUState	false	false	false	false	false
providesDirectionalDerivative	TRUE	TRUE	TRUE	TRUE	Partly[2]
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	FALSE	FALSE	true
canNotUseMemoryManagementFunctions	FALSE	FALSE	true

[1] Supported except for the solvers Lsodar, Dassl, Cvode, Euler, Rkfix2, Rkfix3, and Rkfix4.
[2] Supported except for the solvers Lsodar, Dassl, Euler, Rkfix2, Rkfix3, and Rkfix4.

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.

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 24.

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.

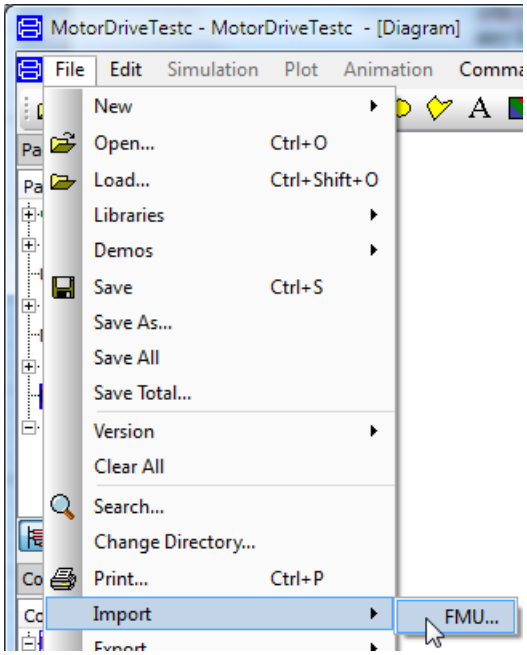
The generated Modelica file will get the name `model_fmu.mo` or `model_fmu_black_box.mo`, depending on the value of `includeAllVariables`.

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 > Import > FMU...**.

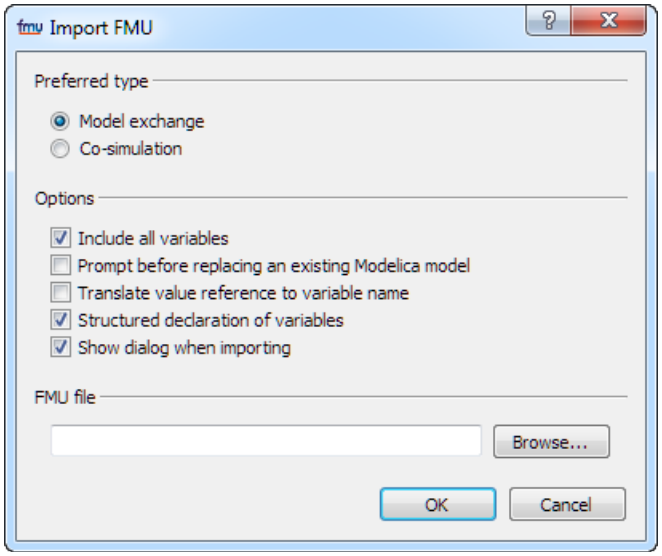


Notes:

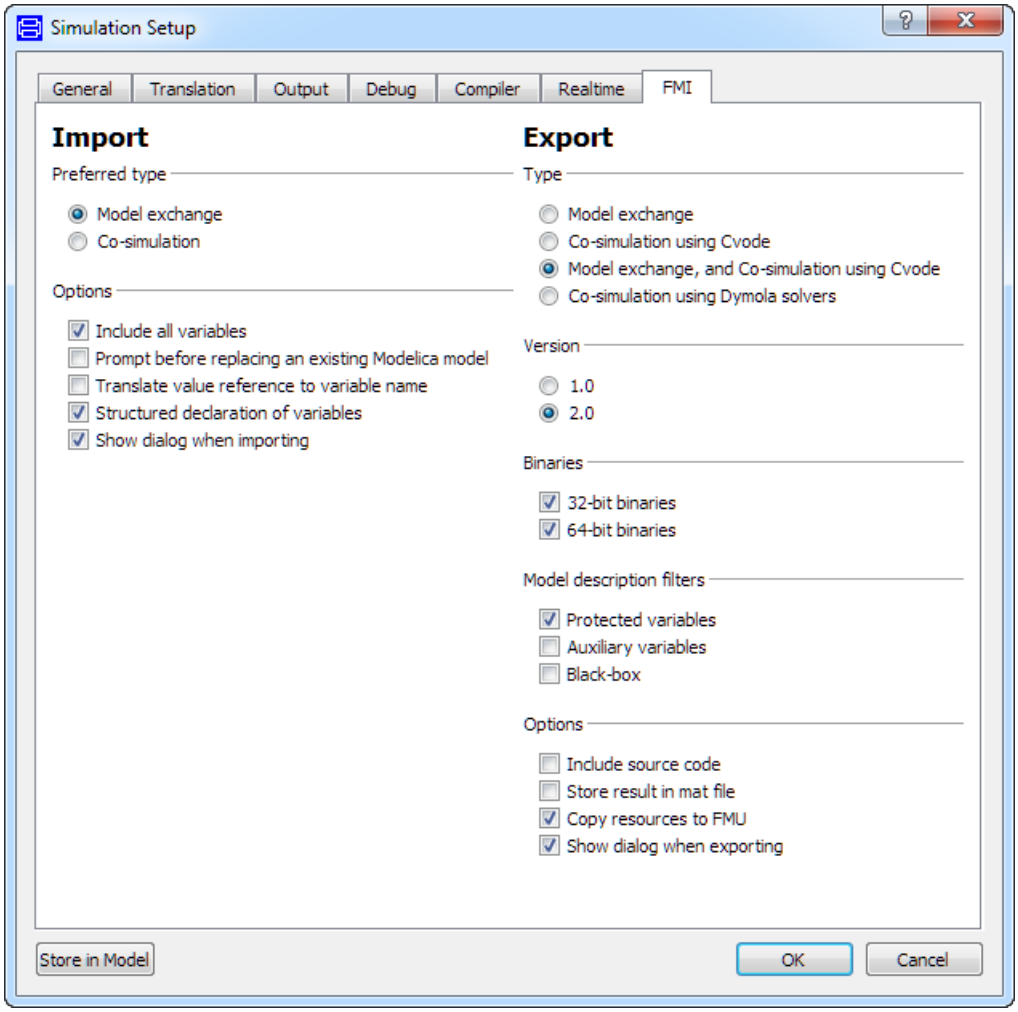
- This command also will be automatically applied on an .fmu file by dragging it into the Dymola main window.

- The command can also be given by clicking the button **Import FMU**  in the Files toolbar.

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 import part of the **FMI** tab of the simulation setup, reached by the command **Simulate > Setup…**, the **FMI** 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).

Five options are available:

- **Include all variables** – corresponds to the function parameter `includeAllVariables` (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 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"`.
- **Show dialog when importing** – this option is by default ticked. If unticked, the Import FMU dialog is not displayed when importing FMUs.

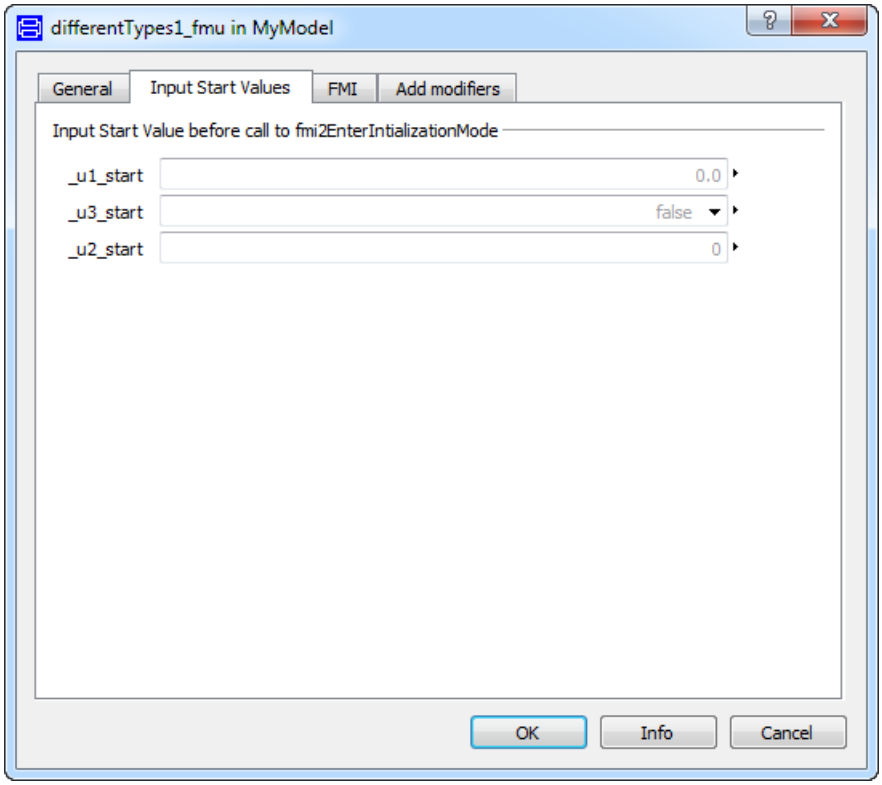
Settings of the imported FMU

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

Input Start Values tab

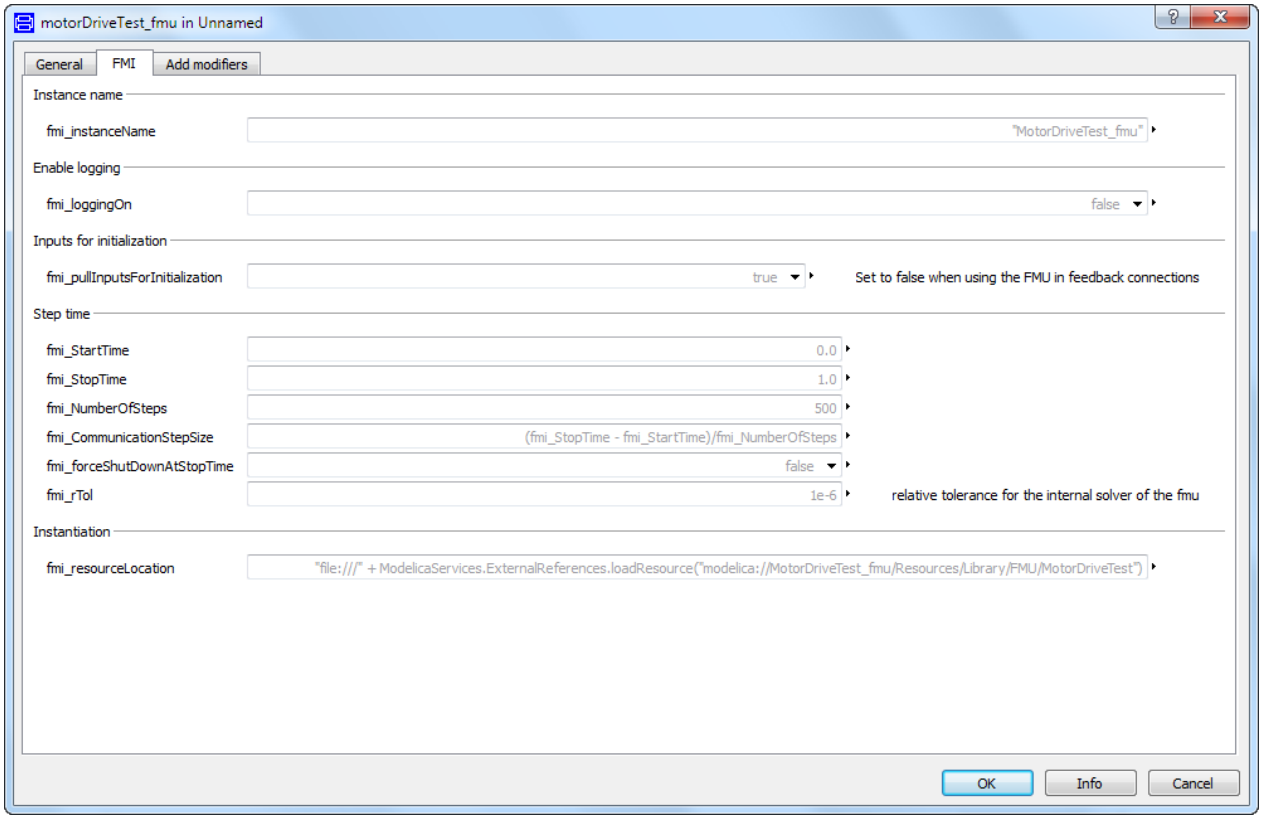
For FMI Model Exchange in FMI version 2.0, input start values can be set before initialization. This should however only be necessary if your FMU is constructed in such a way that the default start values for an input is illegal in the FMU, e.g. division with an input variable having a default value 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

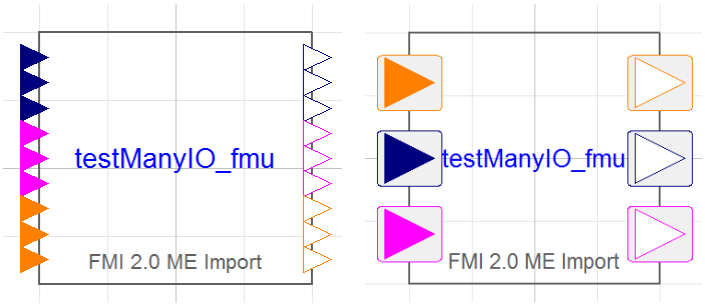


The available settings depend on the FMU type.

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.

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.

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 `t0->t1`.

Your choices are input at time `t0` input at time `t1` or `pre` on input at time `t1` (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 2018 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 supports interpolation and not extrapolation, similar restrictions exists 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

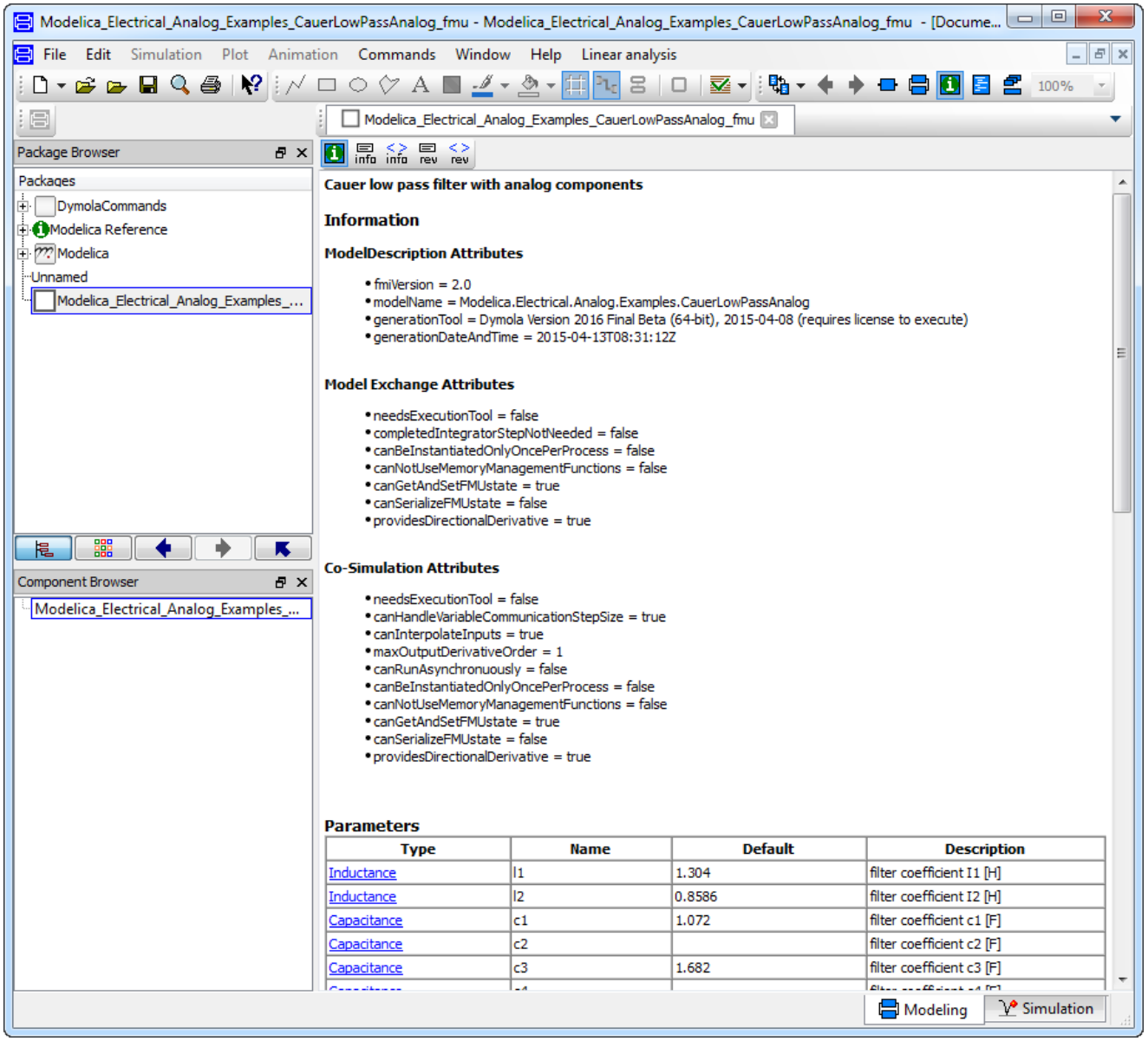
```
Advanced.FMI.UseTrueVariableNames = false;
```

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.



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

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

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

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");
```

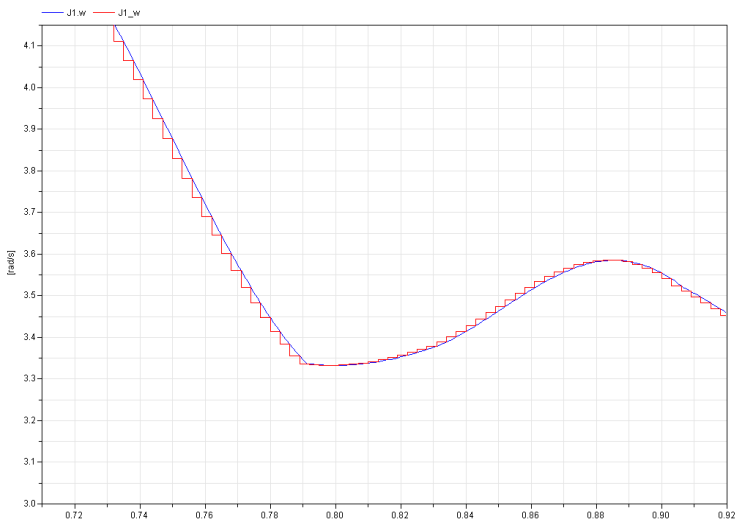
Re-import, in a non-interactive mode, the FMU for co-simulation:

```
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.

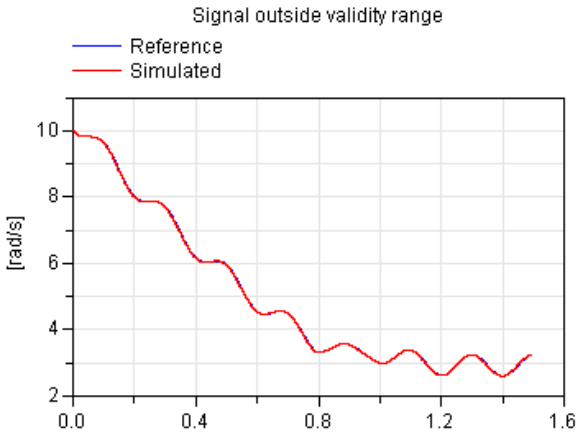
It is available in Modelica\Library under the Dymola installation.

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 FMU Export from Simulink/FMU Import into Simulink: The FMI Kit for Simulink

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

Introduction

FMU Export from Simulink

FMI Kit for Simulink provides a Simulink Coder Target (`rtwsfcnfmi`) to support export of FMUs from Matlab/Simulink. The FMU export package contains implementations of the FMI standards on top of model code generated by Simulink Coder (formerly Real-Time Workshop). The Matlab Target Language Compiler (TLC) is used to construct the XML model description.

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

The utility builds on the Simulink Coder 'S-function Target' configuration that is available in Matlab. In fact, the same model C code is generated by the 'S-function target with FMI' as for the Simulink Coder 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 generated model code and the S-function FMI wrapper, and links with required libraries
- Copies resources, such as images and MEX files, to the FMU `resources` folder
- Constructs the FMI zip archive (.fmu) according to the FMI distribution structure

Release History:

Note: The FMU Export from Simulink package is independent of Dymola and updates are sometimes released in between the official Dymola releases. Information about new released versions can be found at www.dymola.com/FMI.

- 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
- Version 2.0, March 31, 2015 (included with Dymola 2016)
    - FMI 1.0 and 2.0 support
    - Model Exchange and Co-Simulation
    - Support for all Simulink built-in data types
    - MATLAB support for R2010a - R2014b (32- and 64-bit)
    - Support for Visual Studio 2008 and later compilers
- Version 2.1, May 29, 2015
    - Loading of binary MEX S-functions
    - C++ source S-functions

        - Simulink I/O buses with structured naming
        - Black-box FMU generation
        - Block hierarchy in variable names
- Version 2.1.1, June 24, 2015 (a maintenance version)
- Version 2.1.2, October 9, 2015 (included with Dymola 2016 FD01)
        - Support for Matlab R2015a and R2015b
- Version 2.2.0, April 15, 2016 (included with Dymola 2017)
        - Released as part of FMI Kit for Simulink (export and import)
        - Support for global tunable workspace parameters
        - Full support for Matlab R2015b code generation, especially support for parameter references to workspace or mask variables
- Version 2.3.0, October 7, 2016 (included with Dymola 2017 FD01)
        - Support for Matlab R2016a
        - Improved input handling and support for input interpolation
        - Support for FMU export on Linux
- Version 2.4.0, April 7, 2017 (included with Dymola 2018)
        - Support for Matlab R2016b

FMU Import into Simulink

FMI Kit for Simulink contains a Simulink FMU block, which enables embedding of FMUs into Simulink models. With source code FMUs exported with Dymola 2016 or later it is also possible to use FMUs in Rapid Accelerator mode and create target code for RSIM, GRT, and dSPACE ds1005, ds1006, and SCALEXIO platforms.

The package for FMU import into Simulink together with the Dymola support for FMU export facilitates simulation of Dymola models in Simulink. In particular, this enables use of Dymola solvers in Simulink through the FMI Co-Simulation interface.

Support and Usage

FMI Kit for Simulink has full support for both export and import, which means that both versions 1.0 and 2.0 of the FMI standard are supported for both Model Exchange and Co-Simulation. Supported Matlab releases are R2010a to R2016b (32- and 64-bit). FMU export supports both Windows and Linux. FMU import is currently only supported on Windows.

FMI Kit for Simulink can be used for free without any license key.

Support and maintenance is offered to Dymola customers through the regular support channel at www.3ds.com/support.

FMI Kit for Simulink is independent of Dymola and updates are sometimes released in between the official Dymola releases. Information about new released versions can be found at www.dymola.com/FMI.

Installation

FMI Kit for Simulink is located in the `$DYMOLA/Mfiles/FMIKit_for_Simulink` directory of the Dymola distribution or may be also be downloaded as a zip archive through DS FileTransfer after contacting your DS support channel. Since the package is independent of Dymola it may be extracted or copied to any location.

Follow these steps to set up the environment in Matlab:

- Add the `FMIKit_for_Simulink` directory to your Matlab path and then execute the script `FMIKit.initialize()`.
- Optionally, you may add the following to your Matlab startup script to automatically perform the setup for each new session:

  ```
  addpath('C:\Program Files\FMIKit_for_Simulink');
  FMIKit.initialize()
  ```

  (the `addpath` command should be changed to match your system)

Exporting FMUs from Simulink

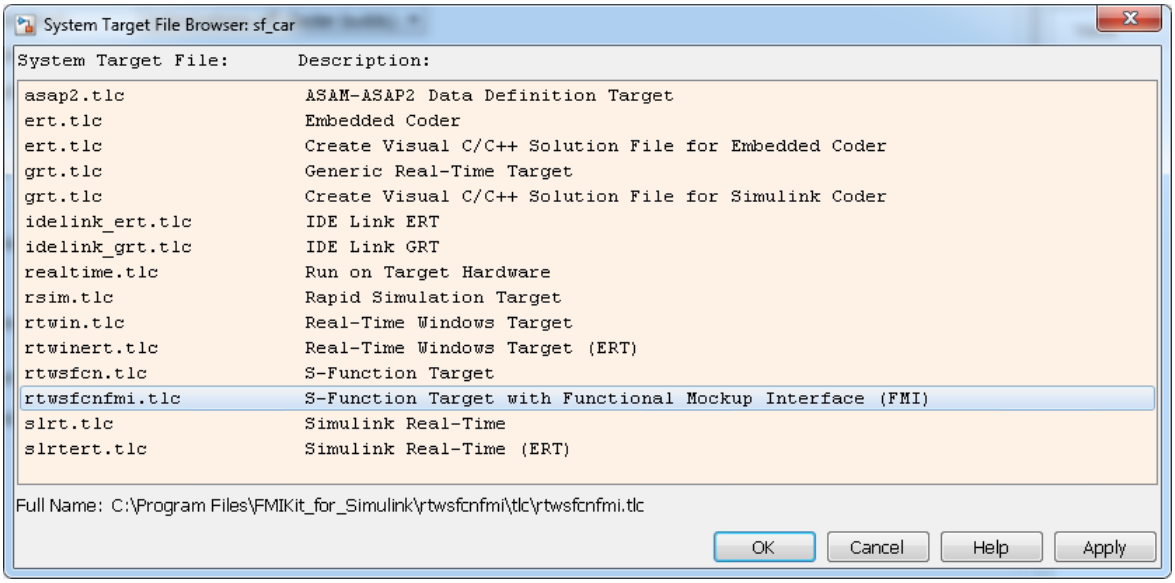
This section describes the procedure to export an FMU from Simulink and the associated settings / configurations.

Adding input and output ports

If the Simulink model to be exported as an FMU should be possible to connect to 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. Hierarchical Simulink buses are supported as input and output port types.

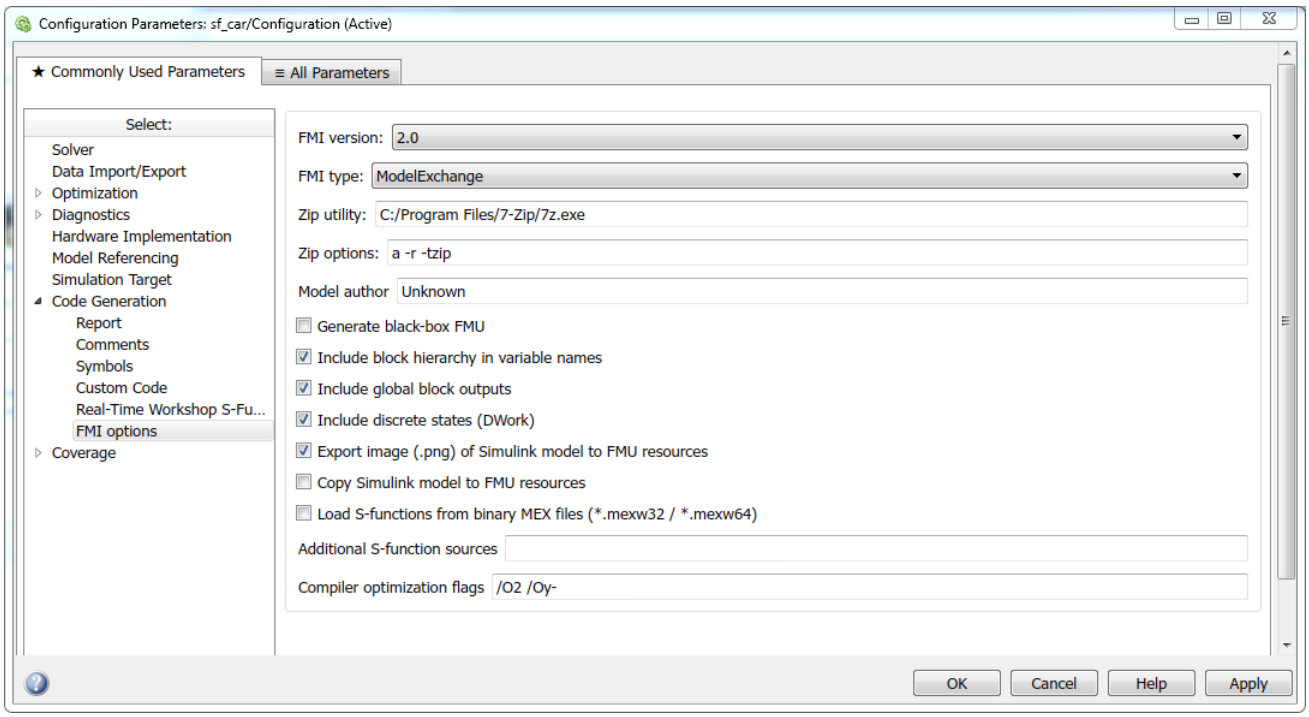
Selecting system target file

In the Simulink **Configuration Parameters** dialog, choose the **Code Generation** tab and click Browse to select a different System Target File. Select `rtwsfcnfmi.tlc` in the list:



Options for FMU export

After selecting the `rtwsfcnfmi.tlc` target, the tab **FMI options** becomes available in the **Code Generation** tab. A description of each option follows below.



- FMI version
    - Selects FMI version for the export (1.0 or 2.0)
- FMI type
    - Specifies FMI type (ModelExchange or CoSimulation)
- Zip utility
    - Path to Zip utility used to build the FMU archive (the default on Windows is 7-zip, which can be downloaded from www.7-zip.org.)
- Zip options
    - Command line options passed to the Zip utility
- Model author
    - Specifies the model author for the FMU XML file
- Generate black-box FMU
    - Selects if the FMU should be generated as a black box (only inputs and outputs exposed).
- Include block hierarchy in variable names
    - Selects if variable names in the FMU XML file should be generated in a structured view using block hierarchy notation. Read more about variable naming below.
- Include global block outputs
    - Selects if block outputs should be included in the FMU XML file. Has no effect if black-box export has been selected.
- Include discrete states (DWork)
    - Selects if discrete states and modes should be included in the FMU XML file. Has no effect if black-box export has been selected.
- Export image (.png) of Simulink model to FMU resources
    - Selects if an image of the top-level Simulink model should be exported with the FMU. The exported image will be located in the subfolder `SimulinkModel` of the FMU resources.
- Copy Simulink model to FMU resources
    - Selects if the whole Simulink model should be copied to the FMU. The model will be located in the subfolder `SimulinkModel` of the FMU resources.
- Load S-functions from binary MEX files

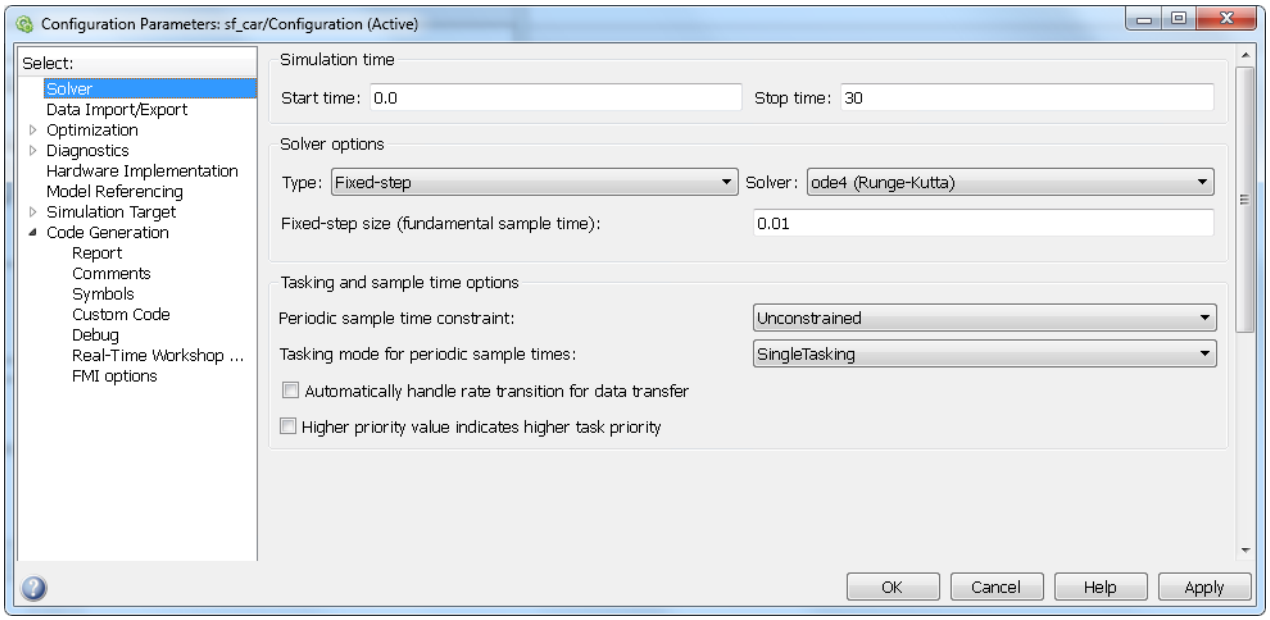
- Selects that S-functions in the model will be loaded from pre-compiled binary MEX files instead of using stand-alone compilation of S-function sources (more details on S-functions below). **Note:** This checkbox should only be used if your model has S-function blocks.

- Additional S-function sources
    - List of additional user source files for stand-alone S-function compilation. Should be used instead of Custom Code > Source Files to ensure that the correct compiler options are used.
- Compiler optimization flags
    - User-defined optimization flags to be used by the compiler (default /O2 /Oy- for Visual Studio on Windows and –O2 for GCC on Linux).

Solver settings

These are the recommended settings for **Configuration Parameters -> Solver**

- **Model Exchange export**: Both Variable-step and Fixed-step solvers supported (recommended to use Variable-step when possible to support accurate event detection using non-sampled zero crossings).
- **Co-Simulation export**: Requires a Fixed-step solver (the selected solver is compiled into the FMU).
- It is also recommended to explicitly set the Tasking mode to SingleTasking.



Including S-functions in the exported FMU

Models containing S-functions can be exported and the S-functions can be included in the FMU either from C/C++ sources or as binary MEX files. Note that the S-functions are not allowed to call into the Matlab environment, e.g., using `mexCallMATLAB` or `mexEvalString`.

Including S-functions from C/C++ sources

Source compilation of S-functions is default and is used if the option *Load S-functions from binary MEX files* is not selected.

The S-function sources (C or C++) should be available and located in the same directory as the Simulink model. The S-function sources are then automatically compiled and linked to the FMU and no further configuration is needed in the Simulink model.

Note that source compilation of S-functions defines the flag NRT, which is used to indicate that the S-function is generated by Simulink Coder (or user-written) for non-real-time applications using a variable-step (or fixed-step) solver. For S-functions that should be built as MEX files for use in Simulink, it is recommended to use the binary MEX file inclusion as described below.

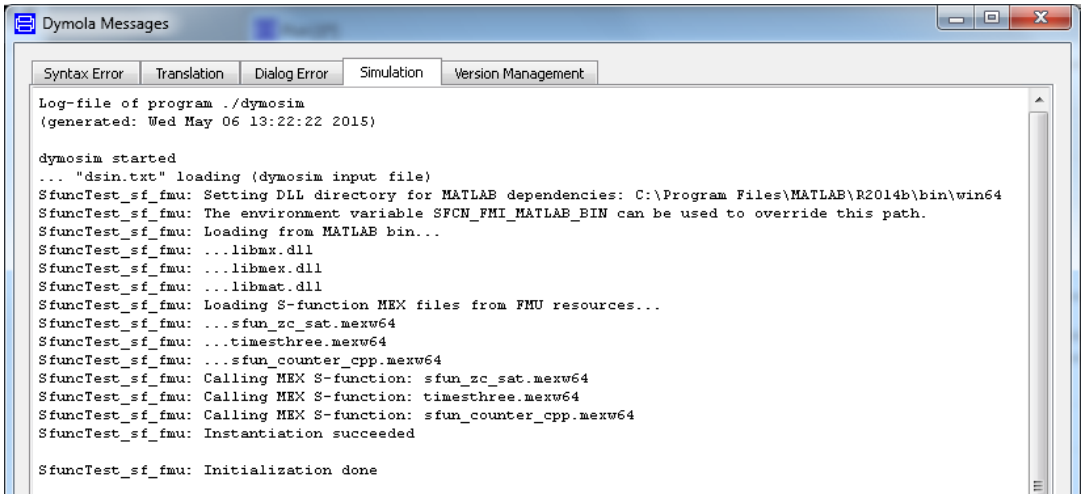
Including S-functions from binary MEX files

If the option *Load S-functions from binary MEX files* is selected, no compilation of S-function sources is performed. Instead, the S-function MEX files are copied to the FMU (to `resources\SFunctions`) and code is added to dynamically load and call the MEX files when the FMU is instantiated. This option will also create dependencies on Matlab binaries (which will not be copied to the FMU).

On Windows, the FMU will by default try to load the Matlab binaries from the bin directory of the exporting MATLAB installation, which means that export / import on the same computer should work seamlessly. The environment variable `SFCN_FMI_MATLAB_BIN` can be used to specify a different directory from where to load the Matlab DLLs (for example a Matlab runtime installation on a different computer).

On Linux, it is required to use the environment variable `LD_LIBRARY_PATH` to specify the path to the Matlab binaries.

With logging enabled the FMU outputs information about the loading of binaries and MEX files during instantiation. The following is an example of importing a 64-bit FMU with MEX file dependencies into Dymola on Windows:



Configuring Visual Studio Compiler

The FMI binary is built using the same version of compiler as used when building MEX files in Matlab. The compiler is configured in Matlab using the command

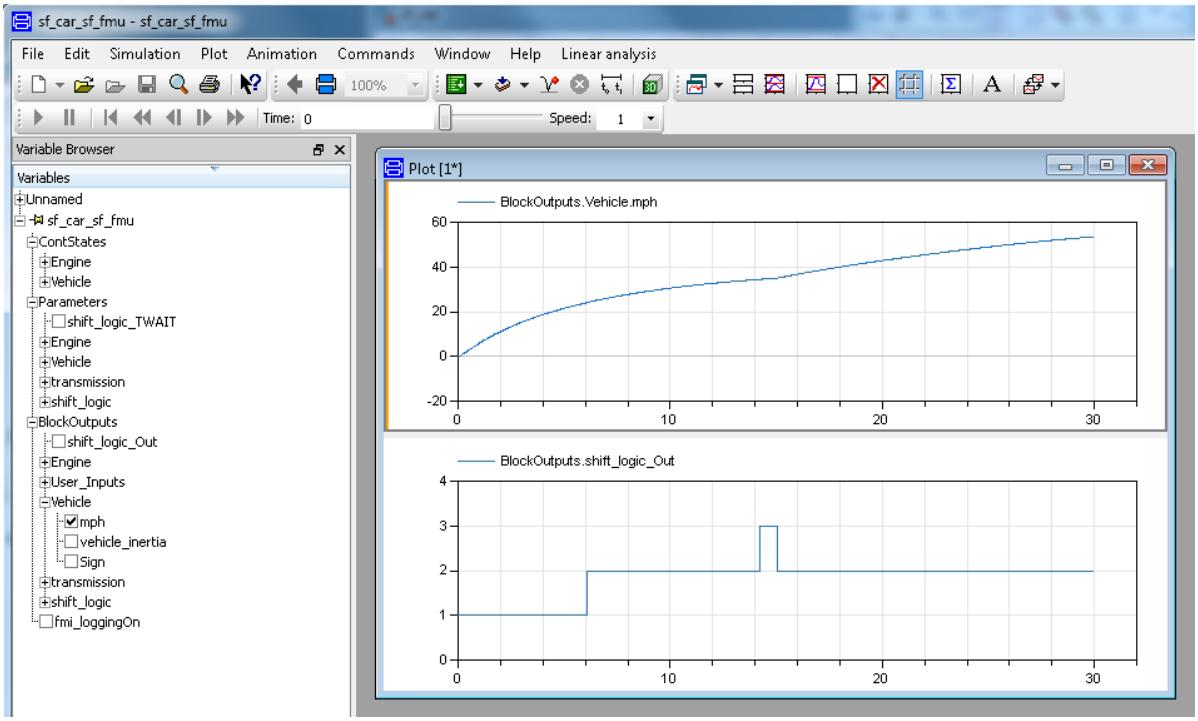
```
>> mex –setup
```

Variable naming

Two options are available for naming of variables in the FMU XML file. With the option *Include block hierarchy in variable names* selected, variable names are generated with block-hierarchical notation and the XML model description specifies the attribute `variableNamingConvention="structured"`. Alternatively, with the box de-selected, the Simulink Coder C code identifiers (not traceable back to model) are used as variable names and the XML specifies `variableNamingConvention="flat"`.

The variable names for continuous-time states, discrete states, parameters, and block outputs are separated into the top-level categories *ContStates*, *DiscStates*, *Parameters*, and *BlockOutputs* in the structured view (see example from Dymola structured FMU import below). This is to ensure unique variable names in the FMU XML file, since variable names from different categories are not guaranteed to be unique within a block. In the flat view, the variable names are appended with `_xc`, `_xd`, `_pm`, and `_wb`, respectively.

The flat view is guaranteed to generate unique variable names in all cases, whereas the structured view in some rare cases could produce name conflicts (on limitations, see section “Limitations and Trouble-Shooting” below).



Building the FMU

Start the build process by pressing **Ctrl-B** (or through the Simulink Code menu).

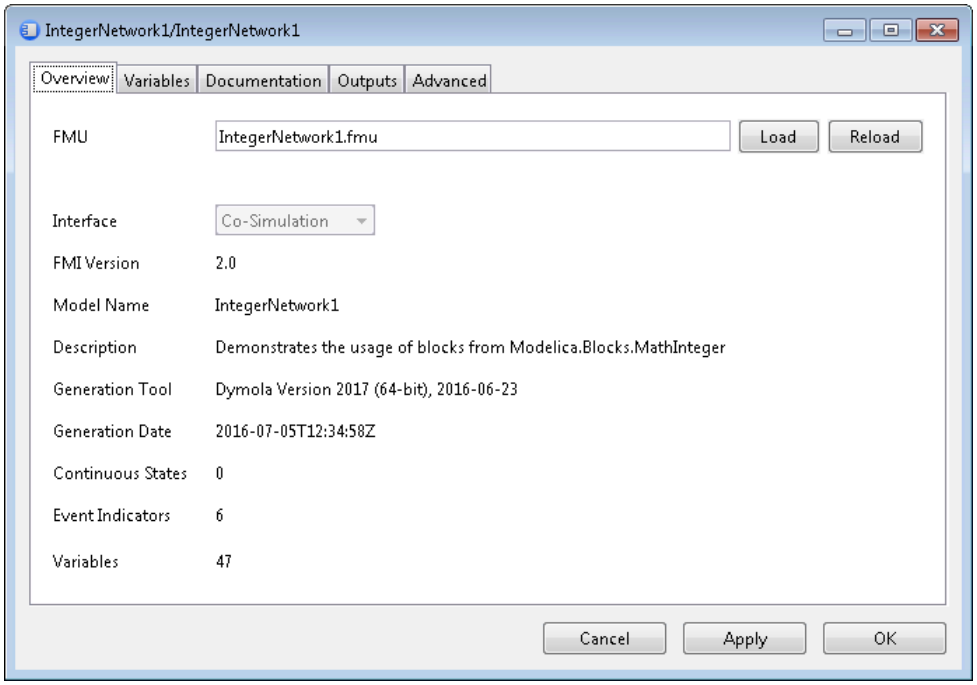
The build process will compile the generated model code using the FMI Simulink wrapper and link with the required Matlab and system libraries to create the FMU binaries. The build process will also create the FMI XML model description, `modelDescription.xml`, and construct the FMI archive, `<modelname>_sf.fmu` in the current working directory.

Importing FMUs into Simulink

This section describes the procedure to import an FMU into Simulink and the associated settings / configurations in the user interface. There is also a set of Matlab commands to interact with the FMI Kit import. These are described more in detail in the HTML documentation accessed through `FMIKit_for_Simulink/html/fmikit.html`

Adding FMUs to a model

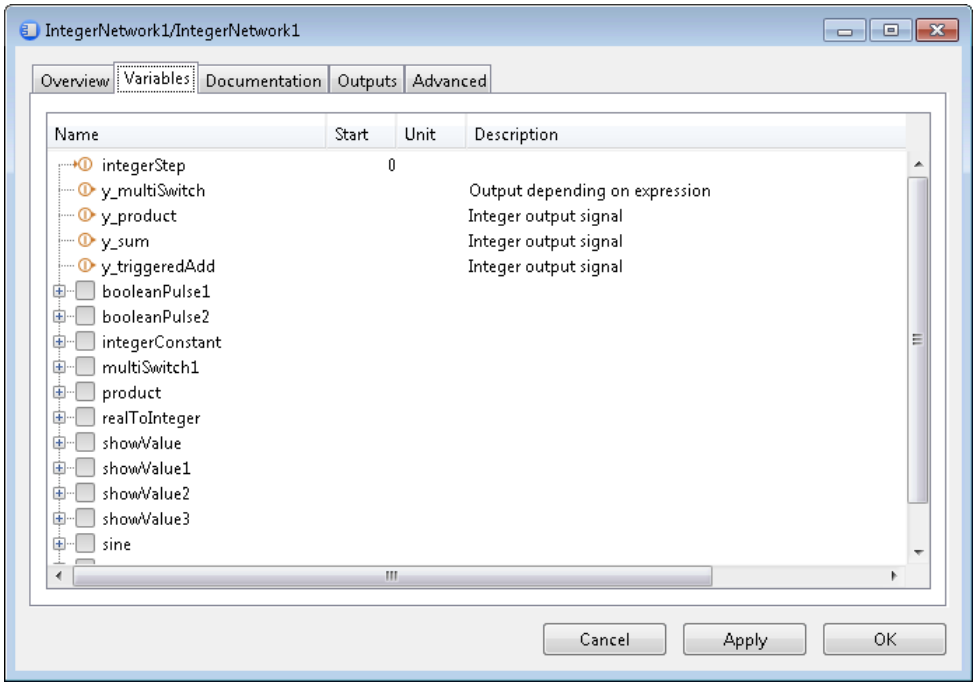
- Open the Simulink library browser (**View > Library Browser**) and drag the FMU block from the FMI Kit library into your model.
- Double-click the FMU block, select **Load** and choose the FMU.
- Click **OK**.



The FMU is automatically extracted to the directory specified under Advanced > Upzip Directory. This directory must remain in the same relative path when the model is moved to a different directory or machine.

For FMI 2.0 FMUs that support both model exchange and co-simulation the interface kind can be selected.

Variables and start values

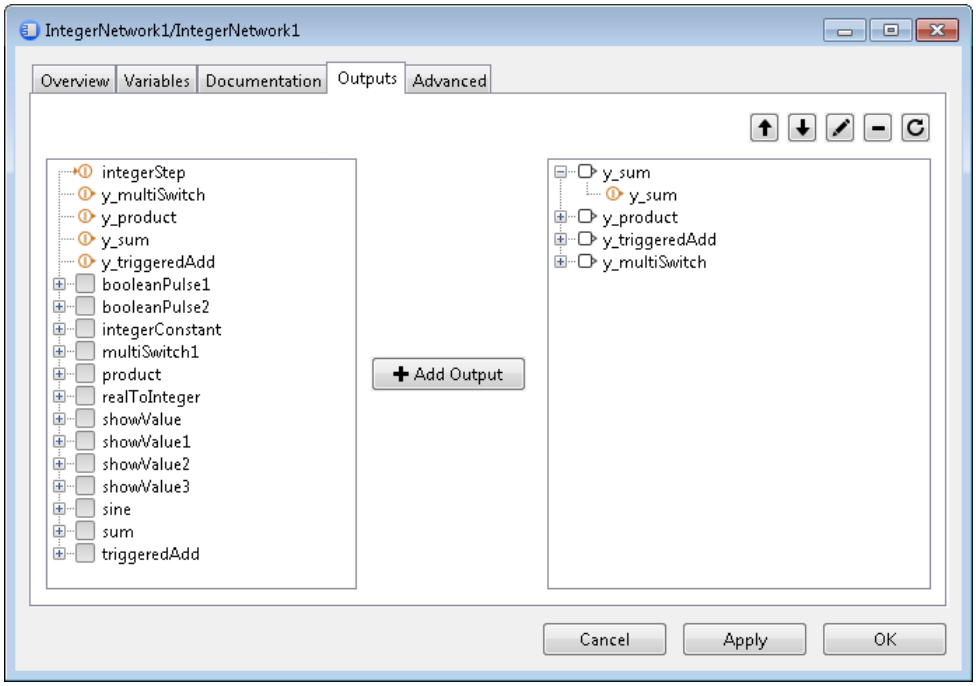


The **Variables** tab shows all variables of the FMU. Input variables are marked with an arrow on the left, output variables with an arrow on the right of the icon.

The start value, unit, and description of the variable (if provided) are displayed in the Start, Unit, and Description columns. Start values that were explicitly set are displayed as bold text.

To change the start value of a variable, click in the respective field in the “Start” column and enter an expression that evaluates to the respective type of the variable. Changed start values are indicated by bold text. To reset the start value to its default, clear the “Start” field.

Output ports

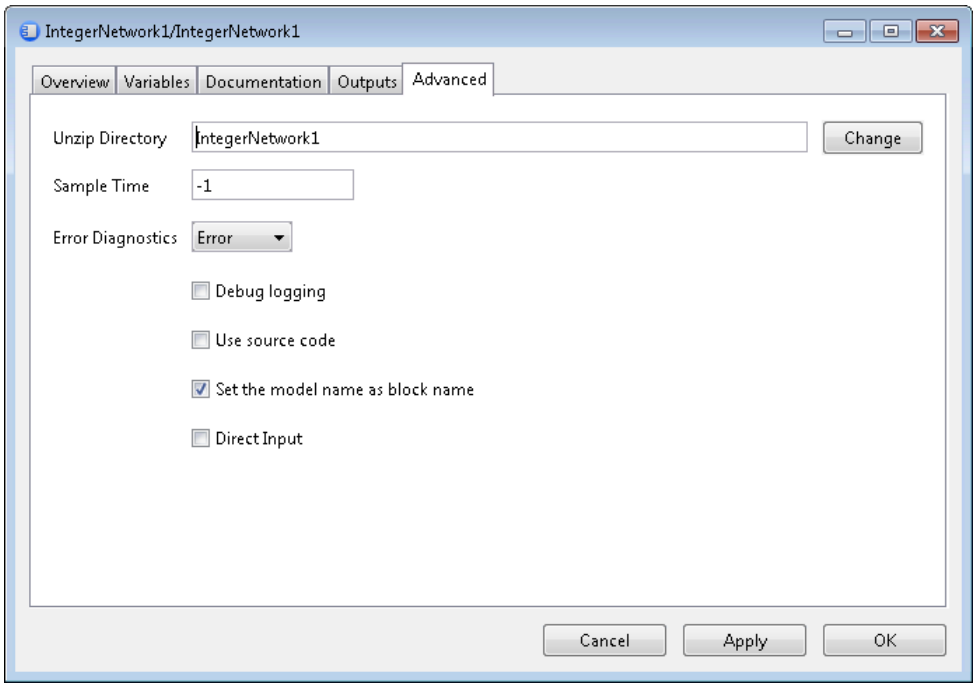


By default the block has the output ports defined by the FMU.

- To add output ports, select one or more variables in the left view and click “Add Output”
- To remove output ports select the ports in the right view and click “-“
- To move an item in the right view, select it and use the up and down buttons
- To restore the default output ports click the reset button.

Advanced settings

On the **Advanced** tab you can change additional settings for the FMU block:



- Unzip Directory
    - The folder where the FMU is extracted. The path can be absolute or relative to the model file. To use a custom path, change this field before loading the FMU.
- Sample Time
    - The sample time for the block (use -1 for inherited)
- Error Diagnostics
    - Determines how to handle errors reported by the FMU
- Debug Logging
    - Enables the debug logging to the Matlab console
- Use Source Code
    - If checked, a source S-function `sfun_<model_name>.c` is generated from the FMU source code which gets automatically compiled when **Apply** or **OK** button is clicked. For FMI 1.0 this feature is only available for FMUs generated with Dymola 2016 and later.

- Set Model Name
    - Use the model name of the FMU as block name
- Direct Input
    - If checked, `ssSetInputPortDirectFeedThrough(true)` is set for all input ports of the FMU and the value of the block´s inputs `u` and `t+1` is applied to the input variables of the FMU at time `t`. This gives better result for FMUs that contain direct terms and do not support input interpolation.

      If not checked, `ssSetInputPortDirectFeedThrough(true)` is only set for input ports whose input variables have output variables with a direct dependency. The derivative `der_u` for these input variables is set such that `u(t)+der_u(t)*step_size=u(t+1)` if the FMU supports input interpolation. Variables that are manually added to the block´s output ports are assumed to depend on all input variables.

Source code FMUs

With source code FMUs you can use advanced simulation targets that require code generation.

To use FMU source code, open the block dialog and on the "Advanced" tab select "Use source code". After clicking **OK**, FMI Kit generates a source S-function `.c` and builds S-function MEX file `mexw32` (or `mexw64` on a 64-bit platform). You can now use the following additional simulation targets: Rapid Accelerator, RSIM, GRT, ds1005, ds1006.

Limitations and Trouble-Shooting

FMU Export

The following relates to version 2.4.0 of the rtwsfcnfmi target:

On Windows, the export supports Visual Studio 2008 (9.0) and later compilers as supported with the respective MATLAB releases.

On Linux, the package should support the versions of gcc supported with the respective Matlab releases. The object files shipped in the package have all been compiled using gcc 4.3.4.

The FMU is compiled with dynamic loading of the C run-time on Windows. This may require installation of the corresponding Visual Studio redistributables on the target platform.

The option *Include block hierarchy in variable names* could in very rare cases give rise to name conflicts in the XML variable names. For example, any special characters in Simulink block names will be converted to underscore which may lead to name conflicts. It is recommended to avoid using special characters in block names with this option (carriage return and space are safe to use).

For multiple instances of conditionally executed nonvirtual subsystems or Stateflow charts, it is required to select “Treat as atomic unit” and set “Functions packaging” to “Inline” for the subsystems/charts.

S-functions in the exported model are not allowed to call into the MATLAB environment, e.g., using `mexCallMATLAB` or `mexEvalString`.

The FMU export target is not model reference compliant.

The package is subject to the same limitations as the standard S-Function Target.

File Structure

The `rtwsfcnfmi` target folder (`FMIKit_for_Simulink\rtwsfcnfmi`) consists of six sub-directories and the included files are described briefly below.

bin

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

c

This directory holds C source files to include and compile the Simulink Coder-generated model code. The standard FMI header files are located in the sub-directory `fmi`.

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\spec

This folder contains the official specification documents for FMI 1.0 and 2.0 as a reference.

tlc

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

2 Index

M

S

X