FMU Export of CYMDIST

Release 1.0.0rc1

LBNL - Building Technology and Urban Systems Division

Nov 28, 2017

CONTENTS

INTRODUCTION

This user manual explains how to install and use CYMDISTToFMU. CYMDISTToFMU is a software package written in Python which allows users to export the distribution simulation program CYMDIST version 7.2 as a *Functional Mock-up Unit* (FMU) for model exchange or co-simulation using the *Functional Mock-up Interface* (FMI) standard version 1.0 or 2.0. This FMU can then be imported into a variety of simulation programs that support the import of Functional Mock-up Units.

INSTALLATION AND CONFIGURATION

This chapter describes how to install, configure, and uninstall CYMDISTToFMU.

Software requirements

To export CYMDIST as an FMU, CYMDISTToFMU needs:

1. Python and following dependencies:

   - jinja2

   - lxml

2. Modelica parser

3. C-Compiler

CYMDISTToFMU has been tested with:

- Python 2.7.13 and 3.5.0

- Three Modelica parsers

  - Dymola 2018 on Windows

  - JModelica 2.0 on Windows, and JModelica trunk version 9899 on Linux

  - OpenModelica 1.11.0 on Windows

- C-Compiler: Microsoft Visual Studio 10 Professional

Note: CYMDISTToFMU can use OpenModelica and Dymola to export CYMDIST as an FMU. However Open-Modelica 1.11.0 does not copy all required libraries dependencies to the FMU. As a workaround, CYMDISTToFMU checks if there are missing libraries dependencies and copies the dependencies to the FMU.

Installation

To install CYMDISTToFMU, proceed as follows:

1. Add following folders to your system path:

   - Python installation folder (e.g. `C:\Python34`)

   - Python scripts folder (e.g. `C:\Python34\Scripts`),

- Dymola executable folder (e.g. `C:\Program Files(x86)\Dymola2017 FD01\bin`)

- JModelica installation folder (e.g. `C:\JModelica.org-2.0`)

- OpenModelica executable folder (e.g. `C:\OpenModelica1.11.0-32bit\bin`)

You can add folders to your system path by performing following steps on Windows 8 or 10:

- In Search, search for and then select: System (Control Panel)

- Click the Advanced system settings link.

- Click Environment Variables. In the section System Variables, find the PATH environment variable and select it. Click Edit.

- In the Edit System Variable (or New System Variable) window, specify the value of the PATH environment variable (e.g. `C:\Python34, C:\Python34\Scripts`). Click OK. Close all remaining windows by clicking OK.

- Reopen Command prompt window for your changes to be active.

To check if the variables have been correctly added to the system path, type `python`, `dymola`, `pylab`, or `omc` into a command prompt to see if the right version of Python, Dymola, JModelica, or OpenModelica starts up.

Note: To avoid adding Dymola, JModelica, or OpenModelica to the system path, provide the path to the executables to CYMDISTToFMU.py. See *Command-line use* for the lists of arguments of CYMDISTToFMU.

---

2. Install CYMDISTToFMU by running

```
> pip install --user CYMDISTToFMU
```

Note: Use the `--user` command line option to install CYMDISTToFMU so it can be installed in your Python 3.4 user installation directory and can write files to your disk. The Python 3.4 user installation directory is typically `C:\Users\YourUserName\AppData\Roaming\Python\Python34\site-packages` on Windows, where `YourUserName` is your system login user name.

The installation directory should contain the following subdirectories:

- `bin/` (Scripts for running unit tests)

- `doc/` (Documentation sources)

- `fmus/` (FMUs folder)

- `parser/` (Python scripts, Modelica templates and XML validator files)

UnitTests

To test your installation run from the installation `bin` folder

```
> python runUnitTest.py
```

Uninstallation

To uninstall CYMDISTToFMU, run

```
> pip uninstall CYMDISTToFMU
```

BEST PRACTICE

This section explains to users the best practice in configuring a CYMDIST XML input file for an FMU.

To export CYMDIST as an FMU, the user needs to write an XML file which contains the list of inputs, outputs and parameters of the FMU. The XML snippet below shows how a user has to write such an input file. A template named `CYMDISTModeldescritpion.xml` which shows such a file is provided in the `parser/utilities` installation folder of CYMDISTToFMU. This template should be adapted to create new XML input file.

The following snippet shows an input file where the user defines 6 input and 6 output variables.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<CYMDISTModelDescription
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  fmiVersion="2.0"
  modelName="cymdist"
  description="Input data for a CYMDIST FMU"
  generationTool="CYMDISTToFMU">
  <ModelVariables>
    <ScalarVariable
      name="VMAG_A"
      description="VMAG_A"
      causality="input">
      <Real
        unit="V"
        start="0.0"/>
    </ScalarVariable>
    <ScalarVariable
      name="VMAG_B"
      description="VMAG_B"
      causality="input">
      <Real
        unit="V"
        start="0.0"/>
    </ScalarVariable>
    <ScalarVariable
      name="VMAG_C"
      description="VMAG_C"
      causality="input">
      <Real
        unit="V"
        start="0.0"/>
    </ScalarVariable>
    <ScalarVariable
      name="VANG_A"
      description="VANG_A"
      causality="input">
```

```
37        <Real
38          unit="deg"
39          start="0.0"/>
40        </ScalarVariable>
41      <ScalarVariable
42        name="VANG_B"
43        description="VANG_B"
44        causality="input">
45        <Real
46          unit="deg"
47          start="-120.0"/>
48      </ScalarVariable>
49      <ScalarVariable
50        name="VANG_C"
51        description="VANG_C"
52        causality="input">
53        <Real
54          unit="deg"
55          start="120.0"/>
56      </ScalarVariable>
57      <ScalarVariable
58        name="IA"
59        description="IA"
60        causality="output">
61        <Real
62          unit="A"/>
63      </ScalarVariable>
64      <ScalarVariable
65        name="IB"
66        description="IB"
67        causality="output">
68        <Real
69          unit="A"/>
70      </ScalarVariable>
71      <ScalarVariable
72        name="IC"
73        description="IC"
74        causality="output">
75        <Real
76          unit="A"/>
77      </ScalarVariable>
78      <ScalarVariable
79        name="IAngleA"
80        description="IAngleA"
81        causality="output">
82        <Real
83          unit="deg"/>
84      </ScalarVariable>
85      <ScalarVariable
86        name="IAngleB"
87        description="IAngleB"
88        causality="output">
89        <Real
90          unit="deg"/>
91      </ScalarVariable>
92      <ScalarVariable
93        name="IAngleC"
94        description="IAngleC"
```

```
95          causality="output">
96        <Real
97          unit="deg"/>
98      </ScalarVariable>
99    </ModelVariables>
100 </CYMDISTModelDescription>
```

To create such an input file, the user needs to specify the name of the FMU (Line 5). This is the `modelName` which should be unique. The user then needs to define the inputs and outputs of the FMUs. This is done by adding a `ScalarVariable` into the list of `ModelVariables`.

To parametrize the `ScalarVariable` as an input variable, the user needs to

- define the name of the variable (Line 10),

- give a brief description of the variable (Line 11)

- give the causality of the variable (`input` for inputs, `output` for outputs) (Line 12)

- define the type of variable (Currently only `Real` variables are supported) (Line 13)

- give the unit of the variable (Currently only valid Modelica units are supported) (Line 14)

- give a start value for the input variable (This is optional) (Line 15)

To parametrize the `ScalarVariable` as an output variable, the user needs to

- define the name of the variable (Line 58),

- give a brief description of the variable (Line 59)

- give the causality of the variable (`input` for inputs, `output` for outputs) (Line 60)

- define the type of variable (Currently only `Real` variables are supported) (Line 61)

- give the unit of the variable (Currently only valid *Modelica* units are supported) (Line 62)

Note: If valid *Modelica* units can't be used (e.g. Line 14), then remove the *unit* field from the input file when defining new `ScalarVariable`.

CREATING AN FMU

This chapter describes how to create a Functional Mockup Unit, starting from a CYMDIST XML input file. It assumes you have followed the *Installation and Configuration* instructions, and that you have created the CYMDIST model description file following the *Best Practice* guidelines.

Command-line use

To create an FMU, open a command-line window (see *Notation*). The standard invocation of the CYMDISTToFMU tool is:

```
> python  <scriptDir>CYMDISTToFMU.py -s <python-scripts-path>
```

where `scriptDir` is the path to the scripts directory of CYMDISTToFMU. This is the `parser` subdirectory of the installation directory. See *Installation and Configuration* for details.

An example of invoking `CYMDISTToFMU.py` is

```
# Windows:
> python parser\CYMDISTToFMU.py -s parser\utilities\cymdist_wrapper.py,d:\calc.py
```

Following requirements must be met when using CYMDISTToFMU

- All file paths can be absolute or relative.

- If any file path contains spaces, then it must be surrounded with double quotes.

`CYMDISTToFMU.py` supports the following command-line switches:

Op-tions	Purpose
-s	Path to python script required to run the CYMDIST. The name of the script is `cymdist_wrapper.py`. It is provided provided in `parser/utilities/cymdist_wrapper.py`
-c	Path to the CYMDIST model or configuration file.
-i	Path to the XML input file with the inputs/outputs of the FMU. Default is `parser/utilities/CYMDISTModelDescription.xml`
-v	FMI version. Options are `1.0` and `2.0`. Default is `2.0`
-a	FMI API version. Options are `cs` (co-simulation) and `me` (model exchange). Default is `me`.
-t	Modelica compiler. Options are `dymola` (Dymola), `jmodelica` (JModelica), and `openmodelica` (OpenModelica). Default is `dymola`.
-pt	Path to the Modelica executable compiler.

The main functions of CYMDISTToFMU are

- reading, validating, and parsing the CYMDIST XML input file. This includes removing and replacing invalid characters in variable names such as `*+-` with `_`,

- writing Modelica code with valid inputs and outputs names,

- invoking a Modelica compiler to compile the *Modelica* code as an FMU for model exchange or co-simulation `1.0` or `2.0`.

The next section discusses requirements of some of the arguments of CYMDISTToFMU.

Simulation model or configuration file

An FMU exported by CYMDISTToFMU needs in certain cases a configuration file to run. There are two ways of providing the configuration file to the FMU:

1. The path to the configuration file is passed as the comamnd line argument `"-c"` of CYMDISTToFMU.py. In this situation, the configuration file is copied in the resources folder of the FMU.

2. The path to the configuration is set by the master algorithm before initializing the FMU.

Note: The name of the configuration variable is `_configurationFileName`. This name is reserved and should not be used for FMU input and output names.

Depending on the tool used to export the FMU, following requirements/restrictions apply:

Dymola

- If the path to the configuration file is provided, then Dymola copies the file to its resources folder and uses the configuration file at runtime. In this case, the path to the configuration file can't be set and changed by the master algorithm.

- If the configuration file is not provided, then the path to the configuration file must be set by the master algorithm prior to initializing the FMU.

JModelica

- If the path to the configuration file is provided, then JModelica will not copy it to the resources folder of the FMU. Instead, the path to the configuration is hard-coded in the FMU. As a further restriction, the path to the configuration file can't be set and changed by the master algorithm.

  These are known limitations in JModelica 2.0. The workaround is to make sure that the path of the configuration file is the same on the machine where the FMU will be run.

- If the configuration file is not provided, then CYMDISTToFMU will issue a warning.

OpenModelica

- If the path to a configuration file is provided, then OpenModelica will not copy it to the resources folder of the FMU. Instead, the path to the configuration is hard-coded in the FMU. However, the path to the configuration file can be set and changed by the master algorithm.

  This is a known limitation in OpenModelica 1.11.0. The workaround is to either make sure that the path of the configuration file is the same on the machine where the FMU will be run, or set the path of the configuration file when running the FMU.

- If the configuration file is not provided, then the path to the configuration file must be set by master algorithm prior to initializing the FMU.

Reserved variable names

Following variables names are not allowed to be used as FMU input, output, or parameter names.

- `_configurationFileName`: Variable name used to set the path to the CYMDIST model or configuration file.
- `_saveToFile`: Variable used to set the flag for storing simulation results (1 for storing, 0 else).
- `time`: Internal FMU simulation time.

If any of these variables is used for an FMU input, output, or parameter name, CYMDISTToFMU will exit with an error.

Outputs of CYMDISTToFMU

The main outputs from running `CYMDISTToFMU.py` consist of an FMU named after the `modelName` specified in the input file, a zip file called `"modelname" + ".scripts.zip"`, and a zip file called `"modelname" + ".binaries.zip"`. That is, if the `modelName` is called `CYMDIST`, then the outputs of `CYMDISTToFMU` will be `CYMDIST.fmu`, `CYMDIST.scripts.zip`, and `CYMDIST.binaries.zip`.

The FMU and the zip file are written to the current working directory, that is, in the directory from which you entered the command.

`"modelname" + ".scripts.zip"` contains the Python scripts that are needed to interface with the CYMDIST. The unzipped folder must be added to the `PYTHONPATH` of the target machine where the FMU will be used.

`"modelname" + ".binaries.zip"` contains subdirectories with binaries files that are needed to interface with CYMDIST. Subdirectories with binaries to be supported by the FMU must be added to the system `PATH`. That is, if the FMU is exported for Windows 32 bit, then the subdirectory `"win32"` must be added to the system `PATH` of the target machine where the FMU is run.

Any secondary output from running the CYMDISTToFMU tools can be deleted safely.

Note that the FMU itself is a zip file. This means you can open and inspect its contents. To do so, it may help to change the ".`fmu`" extension to ".`zip`".

Note:

- FMUs exported using OpenModelica 1.11.0 needs significantly longer compilation/simulation time compared to the tested versions of Dymola and JModelica.

- FMUs exported using Dymola 2018 needs a Dymola runtime license to run. A Dymola runtime license is not be needed if the FMU is exported with a version of Dymola which has the `Binary Model Export` license.

USAGE OF CYMDIST AS AN FMU

The following requirements must be met to import and run a CYMDIST FMU:

1. CYME version 7.2 must be installed. CYME can be downloaded from www.cyme.com.

2. The CYMDIST Python API directory must be added to the `PYTHONPATH`. This directory contains scripts needed at runtime by the CYMDIST FMU.

   The CYMDIST Python API directory is in the installation folder of CYME. It can typically be found in `path_to_CYME/CYME/cympy`, where `path_to_CYME` is the path to the installation folder of CYME 7.2.

   To add the CYMDIST Python API scripts folder to the `PYTHONPATH`, add `path_to_CYME/CYME` to the `PYTHONPATH` with following steps:

   - In Search, search for and then select: System (Control Panel).

   - Click the Advanced system settings link.

   - Click Environment Variables. In the section System Variables, find a variable named `PYTHONPATH` environment variable and select it. If the variable does not exist, create it. Click Edit.

   - In the Edit System Variable (or New System Variable) window, specify the value of the PYTHON-PATH environment variable which should be in our case `path_to_CYME/CYME`.

3. The CYMDIST installation directory must be added to the system `PATH`. This directory contains runtime DLLS (`mkl_core.dll`, `mkl_def.dll`) that are needed at runtime by the CYMDIST FMU.

4. Upon request, the simulation results are saved in a result file which is created in the current working directory. The name of the result file is `xxx_result_.pickle`, where xxx is the FMU model name as defined in the XML input file.

DEVELOPMENT

The development site of this software is at https://github.com/LBNL-ETA/cyder.

To clone the master branch, type

git clone https://github.com/LBNL-ETA/cyder.git

HELP

This chapter lists potential issues encountered when using CYMDISTToFMU.

Compilation failed with Dymola

If the export of the CYMDIST failed when compiling the model with Dymola, comment out `"exit()"` in `parser/utilities/CYMDISTModelica_Template_Dymola.mos` with `"//exit()"`, and re-run `CYMDISTToFMU.py` to see why the complation has failed.

Compilation failed with OpenModelica

If the export of the CYMDIST failed when compiling the model with OpenModelica, check if the variable `OPENMODELICALIBRARY` is defined in the Windows `Environment Variables`.

Note: `OPENMODELICALIBRARY` is the path to the libraries which are required by OpenModelica to compile Modelica models.

Simulation failed when running `CYMDIST.fmu`

If the simulation failed with the exported FMU, check if the unzipped `"modelname" + ".scripts.zip"`, and the subdirectories of `"modelname" + ".binaries.zip"` were added to the `PYTHONPATH`, and the system `PATH` as described in *Outputs of CYMDISTToFMU*.

Note: Any software or Python 3.4 module which is required to run the exported FMU will need to be installed on the target machine where the FMU is run.

Simulation failed with Dymola FMUs

If an FMU exported using Dymola fails to run, check if the version of Dymola which exported the FMU had the `Binary Model Export` license. The `Binary Model Export` license is required to export FMUs which can be run without requiring a Dymola runtime license. You can also inspect the model description of the FMU to see if a Dymola runtime license is required to run the FMU.

NOTATION

This chapter shows the formatting conventions used throughout the User Guide.

The command-line is an interactive session for issuing commands to the operating system. Examples include a DOS prompt on Windows, a command shell on Linux, and a Terminal window on MacOS.

The User Guide represents a command window like this:

```
# This is a comment.
> (This is the command prompt, where you enter a command)
(If shown, this is sample output in response to the command)
```

Note that your system may use a different symbol than ">" as the command prompt (for example, "$"). Furthermore, the prompt may include information such as the name of your system, or the name of the current subdirectory.

GLOSSARY

Dymola Dymola, Dynamic Modeling Laboratory, is a modeling and simulation environment for the Modelica language.

Functional Mock-up Interface The Functional Mock-up Interface (FMI) is the result of the Information Technology for European Advancement (ITEA2) project *MODELISAR*. The FMI standard is a tool independent standard to support both model exchange and co-simulation of dynamic models using a combination of XML-files, C-header files, C-code or binaries.

Functional Mock-up Unit A simulation model or program which implements the FMI standard is called Functional Mock-up Unit (FMU). An FMU comes along with a small set of C-functions (FMI functions) whose input and return arguments are defined by the FMI standard. These C-functions can be provided in source and/or binary form. The FMI functions are called by a simulator to create one or more instances of the FMU. The functions are also used to run the FMUs, typically together with other models. An FMU may either require the importing tool to perform numerical integration (model-exchange) or be self-integrating (co-simulation). An FMU is distributed in the form of a zip-file that contains shared libraries, which contain the implementation of the FMI functions and/or source code of the FMI functions, an XML-file, also called the model description file, which contains the variable definitions as well as meta-information of the model,additional files such as tables, images or documentation that might be relevant for the model.

Modelica Modelica is a non-proprietary, object-oriented, equation-based language to conveniently model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents.

MODELISAR MODELISAR is an ITEA 2 (Information Technology for European Advancement) European project aiming to improve the design of systems and of embedded software in vehicles.

PyFMI PyFMI is a package for loading and interacting with Functional Mock-Up Units (FMUs), which are compiled dynamic models compliant with the Functional Mock-Up Interface (FMI).

Python Python is a dynamic programming language that is used in a wide variety of application domains.

ACKNOWLEDGMENTS

The following people contributed to the development of this program:

- Thierry Stephane Nouidui, Lawrence Berkeley National Laboratory

- Jonathan Coignard, Lawrence Berkeley National Laboratory

- Michael Wetter, Lawrence Berkeley National Laboratory

DISCLAIMERS

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or The Regents of the University of California.

COPYRIGHT AND LICENSE

Copyright

License Agreement

You are under no obligation whatsoever to provide any bug fixes, patches, or upgrades to the features, functionality or performance of the source code ("Enhancements") to anyone; however, if you choose to make your Enhancements available either publicly, or directly to Lawrence Berkeley National Laboratory, without imposing a separate written license agreement for such Enhancements, then you hereby grant the following license: a non-exclusive, royalty-free, perpetual license to install, use, modify, prepare derivative works, incorporate into other computer software, distribute, and sublicense such enhancements or derivative works thereof, in binary and source code form.

PYTHON MODULE INDEX

p