FMU Export of CYMDIST

Release 1.0.0

LBNL - Building Technology and Urban Systems Division

May 08, 2017

CONTENTS

1 Introduction **1**

2 Download **3**

3 Installation and Configuration **5**
- 3.1 Software requirements . . . 5
- 3.2 Installation . . . 5
- 3.3 Uninstallation . . . 6

4 Best Practice **7**

5 Creating an FMU **11**
- 5.1 Command-line use . . . 11
- 5.2 Output of CYMDISTToFMU . . . 12

6 Usage of CYMDIST as an FMU **13**

7 Development **15**

8 Help **17**
- 8.1 Running PyFMI with Python 3.4 on Windows 32 bit . . . 17
  - 8.1.1 Requirements . . . 17
  - 8.1.2 Compilation . . . 18

9 Notation **19**

10 Glossary **21**

11 Acknowledgments **23**

12 Disclaimers **25**

13 Copyright and License **27**
- 13.1 Copyright . . . 27
- 13.2 License Agreement . . . 27

Python Module Index **29**

CHAPTER ONE

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.

CHAPTER TWO

DOWNLOAD

The CYMDISTToFMU release includes scripts and source code to export CYMDIST version 7.2 as an FMU for model exchange or co-simulation for Windows 32.

To install CYMDISTToFMU, follow the section *Installation and Configuration*.

Download the latest development version of CyDER at https://github.com/LBNL-ETA/CyDER/tree/master.

CHAPTER

THREE

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 on Windows with:

- Python 3.4.0
- Dymola 2017 FD01 (Modelica parser)
- OpenModelica 1.11.0 (Modelica parser)
- Microsoft Visual Studio 10 Professional (C-Compiler)

Note: CYMDISTToFMU can use OpenModelica and Dymola to export CYMDIST as an FMU. However OpenModelica 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. Download the CyDER repository from the *Download* page.

The CYMDISTToFMU directory (hereafter referred to as the "installation directory") is the `fmu` subdirectory of CyDER. The installation directory should contain the following subdirectories:

- `fmu/cymdisttofmu/`
    - `bin/` (Python scripts for unit tests)
    - `doc/` (Documentation)

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

2. 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`)
- OpenModelica executable folder (e.g. `C:\OpenModelica1.11.0-32bit\`)

  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`, or `omc` into a command prompt to see if the right version of Python, Dymola or OpenModelica starts up.

4. Install Python dependencies by running

```
pip install -r fmu\cymdisttofmu\bin\cymdisttofmu-requirements.txt
```

Uninstallation

To uninstall CYMDISTToFMU, delete the *installation directory*.

CHAPTER

FOUR

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">
      <Real
        unit="deg"
        start="0.0"/>
        </ScalarVariable>
    <ScalarVariable
      name="VANG_B"
      description="VANG_B"
      causality="input">
      <Real
        unit="deg"
        start="-120.0"/>
    </ScalarVariable>
    <ScalarVariable
      name="VANG_C"
      description="VANG_C"
      causality="input">
      <Real
        unit="deg"
        start="120.0"/>
    </ScalarVariable>
    <ScalarVariable
      name="IA"
      description="IA"
      causality="output">
      <Real
        unit="A"/>
    </ScalarVariable>
    <ScalarVariable
      name="IB"
      description="IB"
      causality="output">
      <Real
        unit="A"/>
    </ScalarVariable>
    <ScalarVariable
      name="IC"
      description="IC"
      causality="output">
      <Real
        unit="A"/>
    </ScalarVariable>
    <ScalarVariable
      name="IAngleA"
      description="IAngleA"
      causality="output">
      <Real
        unit="deg"/>
    </ScalarVariable>
    <ScalarVariable
      name="IAngleB"
      description="IAngleB"
      causality="output">
      <Real
        unit="deg"/>
    </ScalarVariable>
    <ScalarVariable
      name="IAngleC"
      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 `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)

CHAPTER

FIVE

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` on Windows is

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

Following requirements must be met hen 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:

Options	Purpose
-s	Paths to python scripts required to run the CYMDIST. The main Python script must be an extension of the `cymdist_wrapper.py` script which is provided in `parser\utilities\cymdist_wrapper.py`. The name of the main Python script must be `cymdist_wrapper.py`.
-c	Path to the CYMDIST model 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) and `omc` (OpenModelica). Default is `dymola`.
-n	Flag to indicate if FMU needs an external execution tool to run. Options are `true` and `false`. Default is `true`.

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`.

Note:

- If option `<-n>` is `true` then the simulation program/script which will be invoked in the Python scripts provided for option `<-s>` must be installed on the target machine where the FMU will be run.
- If option `<-n>` is `false` then the FMU only needs the Python scripts provided for option `<-s>` to run.

Output of CYMDISTToFMU

The main output from running `CYMDISTToFMU.py` consists of an FMU named after the `modelName` specified in the input file. The FMU is written to the current working directory, that is, in the directory from which you entered the command.

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

Note that the FMU 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 almost `10` times more compilation/simulation time compared to Dymola 2017 FD01.
- FMUs exported using Dymola 2017 FD01 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.

CHAPTER

SIX

USAGE OF CYMDIST AS AN FMU

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

1. Python 3.4 must be installed. This is needed by the master algorithm *PyFMI*.
2. CYME version 7.2 must be installed. CYME can be downloaded from www.cyme.com.
3. 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`. Note that `cympy` is not included in the name of the variable.
4. 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.

   The CYMDIST installation directory is typically found in `path_to_CYME\CYME\`, where ``path_to_CYME` is the path to the installation folder of CYME 7.2.
5. 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.

CHAPTER

SEVEN

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

CHAPTER

EIGHT

HELP

Running PyFMI with Python 3.4 on Windows 32 bit

PyFMI is a python package which can be used to import and run a CYMDIST FMU. In *PyFMI* version 2.3.1, a master algorithm was added to import and link multiple FMUs for integrate simulation. At time of writing, there was no *PyFMI* 2.3.1 executable available for Python 3.4 for Windows 32bit (See PyPyi.). The next steps describe requirements and steps to perform to compile *PyFMI* version 2.3.1 from source.

Note: To avoid having to recompile *PyFMI* dependent libraries from source, we recommend to use pre-compiled Windows binaries whenever available.

Requirements

The next table shows the list of Python modules and softwares used to compile version 2.3.1 of PyFMI from source so it can run with Python 3.4 on Windows 32 bit.

Install PyFMI dependencies with

```
pip install -r fmu/master/bin/pyfmi-requirements.txt
```

Below is a table with dependencies which fail to install using pip. For those, we recommend to use the MS Windows installer directly.

Modules	Version	Link
FMI Library	2.0.2 (source)	http://www.jmodelica.org/FMILibrary
Scipy	0.16.1	https://sourceforge.net/projects/scipy/files/scipy/0.16.1
lxml	3.4.4	https://pypi.python.org/pypi/lxml/3.4.4
Assimulo	2.7b1	https://pypi.python.org/pypi/Assimulo/2.7b1
PyFMI	2.3.1 (source)	https://pypi.python.org/pypi/PyFMI

Note:

- *PyFMI* needs a C-compiler to compile the source codes. We used the Microsoft Visual Studio 10 Professional.
- `pyfmi-requirements.txt` includes the versions of the Python modules which were tested.
- `lxml` cannot be installed using `pip`. Please download and install the executable (`lxml-3.4.4.win32-py3.4.exe`) from PyPI.

Compilation

To compile *PyFMI* from source, run

```
python setup.py install --fmil-home=path_to_FMI_Library\
```

where `path_to_FMI_Library\` is the path to the FMI library.

To use *PyFMI* as a master algorithm to couple a CYMDIST FMU with GridDyn FMU, we refer to the documentation located in `fmu/master/doc/userGuide`.

CHAPTER NINE

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.

CHAPTER TEN

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.

CHAPTER ELEVEN

ACKNOWLEDGMENTS

The development of this documentation was supported by the Assistant Secretary for Energy Efficiency and Renewable Energy, Office of Building Technologies of the U.S. Department of Energy, under contract No. xxx.

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

CHAPTER

TWELVE

DISCLAIMERS

CHAPTER

THIRTEEN

COPYRIGHT AND LICENSE

Copyright

License Agreement

PYTHON MODULE INDEX

p

parser.CYMDISTToFMU, 11

INDEX

D

Dymola, **21**

F

Functional Mock-up Interface, **21**
Functional Mock-up Unit, **21**

M

Modelica, **21**
MODELISAR, **21**

P

parser.CYMDISTToFMU (module), 11
PyFMI, **21**
Python, **21**