
FMU Export of CYMDIST

Release 1.0.0

LBL - 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

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.

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>.

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 fmus\cymdisttofmus\bin\cymdisttofmus-requirements.txt
```

Uninstallation

To uninstall CYMDISTToFMU, delete the *installation directory*.

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 CYMDISTModeldescription.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.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <CYMDISTModelDescription
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      fmiVersion="2.0"
5      modelName="CYMDIST"
6      description="Input data for a CYMDIST FMU"
7      generationTool="CYMDISTToFMU">
8      <ModelVariables>
9          <ScalarVariable
10             name="VMAG_A"
11             description="VMAG_A"
12             causality="input">
13              <Real
14                 unit="V"
15                 start="0.0"/>
16             </ScalarVariable>
17             <ScalarVariable
18                 name="VMAG_B"
19                 description="VMAG_B"
20                 causality="input">
21                  <Real
22                     unit="V"
23                     start="0.0"/>
24                 </ScalarVariable>
25                 <ScalarVariable
26                     name="VMAG_C"
27                     description="VMAG_C"
28                     causality="input">
29                      <Real
30                         unit="V"
31                         start="0.0"/>
32                     </ScalarVariable>
33                     <ScalarVariable
34                         name="VANG_A"
35                         description="VANG_A"
36                         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 `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)

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 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:

Options	Purpose
-s	Paths to python scripts required to run the CYMDIST. The main Python script must be an extension of the <code>cymdist_wrapper.py</code> script which is provided in <code>parser\utilities\cymdist_wrapper.py</code> . The name of the main Python script must be <code>cymdist_wrapper.py</code> .
-c	Path to the CYMDIST model file.
-i	Path to the XML input file with the inputs/outputs of the FMU. Default is <code>parser\utilities\CYMDISTModelDescription.xml</code>
-v	FMI version. Options are <code>1.0</code> and <code>2.0</code> . Default is <code>2.0</code>
-a	FMI API version. Options are <code>cs</code> (co-simulation) and <code>me</code> (model exchange). Default is <code>me</code> .
-t	Modelica compiler. Options are <code>dymola</code> (Dymola) and <code>omc</code> (OpenModelica). Default is <code>dymola</code> .
-n	Flag to indicate if FMU needs an external execution tool to run. Options are <code>true</code> and <code>false</code> . Default is <code>true</code> .

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 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. 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 (`mk1_core.dll`, `mk1_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.

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>

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 --fmi-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`.

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 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 Noudui, 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

“CyDER v01” Copyright (c) 2017, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from the U.S. Dept. of Energy). All rights reserved.

If you have questions about your rights to use or distribute this software, please contact Berkeley Lab’s Innovation & Partnerships Office at IPO@lbl.gov.

NOTICE. This Software was developed under funding from the U.S. Department of Energy and the U.S. Government consequently retains certain rights. As such, the U.S. Government has been granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable, worldwide license in the Software to reproduce, distribute copies to the public, prepare derivative works, and perform publicly and display publicly, and to permit others to do so.

License Agreement

“CyDER v01” Copyright (c) 2017, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from the U.S. Dept. of Energy). All rights reserved.

Redistribution and use in source and binary forms, with or without modification,

are permitted provided that the following conditions are met:

- (1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- (2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- (3) Neither the name of the University of California, Lawrence Berkeley National Laboratory, U.S. Dept. of Energy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

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

`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](#)