FMU Export of a memoryless Python-driven Simulator

Release 1.0.0

LBNL - Building Technology and Urban Systems Division

Apr 27, 2017

CONTENTS

CHAPTER

ONE

INTRODUCTION

This user manual explains how to install and use SimulatorToFMU.

SimulatorToFMU is a software package written in Python which allows users to export any memoryless Python-driven simulation program or script as a *Functional Mock-up Unit* (FMU) for model Exchange or co-Simulation using the *Functional Mock-up Interface* (FMI) standard version 2.0. This FMU can then be imported into a variety of simulation programs that support the import of the Functional Mock-up Interface.

A memoryless Python-driven simulation program/script is a simulation program which meets following requirements:

- The simulation program/script can be invoked through a Python script.
- The invocation of the simulation program/script is memoryless. That is, the output of the simulation program at any invocation time `t` depends only on the inputs at the time `t`.
- The inputs and the outputs of the simulation program/script must be of type `Float`.

CHAPTER TWO

DOWNLOAD

The SimulatorToFMU release includes scripts and source code to export a Simulator as an FMU for model exchange or co-simulation.

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

Download the latest development version of SimulatorToFMU at https://github.com/tsnouidui/SimulatorToFMU.

CHAPTER

THREE

INSTALLATION AND CONFIGURATION

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

3.1 Software requirements

To export a Simulator as an FMU, SimulatorToFMU needs:

1. Python
2. jinja2
3. lxml
4. Modelica compiler

SimulatorToFMU has been tested on Windows with:

- Python 2.7.13
- Python 3.5.0
- Dymola 2017 FD01
- OpenModelica 1.11.0

Note: SimulatorToFMU.py can use OpenModelica to export a Simulator as an FMU. However the FMU cannot be loaded in Dymola or PyFMI because of shared libraries that cannot be loaded.

3.2 Installation

To install SimulatorToFMU, proceed as follows:

1. Download the installation file from the *Download* page.
2. Unzip the installation file into any subdirectory (hereafter referred to as the "installation directory").

The installation directory should contain the following subdirectories:

- `bin/` (Python scripts for running unit tests)
- `doc/` (Documentation)
- `fmus/` (FMUs folder)
- `parser/` (Python scripts, Modelica templates and XML validator files)

3. Add following folders to your system path:

- Python installation folder (e.g. `C:\Python35`)
- Python scripts folder (e.g. `C:\Python35\Scripts`),
- Dymola executable folder (e.g. `C:\Program Files(x86)\Dymola2017 FD01\bin64`)
- OpenModelica executable folder

Note: 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:\Python35`, `C:\Python35\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 bin/simulatortofmu-requirements.txt
```

3.3 Uninstallation

To uninstall SimulatorToFMU, delete the *installation directory*.

CHAPTER

FOUR

CREATING AN FMU

This chapter describes how to create a Functional Mockup Unit. It assumes you have followed the *Installation and Configuration* instructions, and that you have created the Simulator model description file as well as the Python script required to interface the Simulator following the *Best Practice* guidelines.

4.1 Command-line use

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

```
> python3  <scriptDir>SimulatorToFMU.py <python-scripts-path>
```

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

An example of invoking `SimulatorToFMU.py` on Windows is

```
# Windows:
> python3 parser\SimulatorToFMU.py -s parser\\utilities\\Simulator.py, d:\\calcEng.py
```

Following requirements must be met hen using SimulatorToFMU

- All file paths can be absolute or relative.
- If any file path contains spaces, then it must be surrounded with double quotes.

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

Options	Purpose
-s	Paths to python scripts required to run the Simulator. On Windows Operating system, the paths must use **double backward slash**. The main Python script must be an extension of the `Simulator.py` script which is provided in `parser\utilities\Simulator.py`. The name of the main Python script must be `Simulator.py`.
-c	Path to the Simulator model file.
-i	Path to the XML input file with the inputs/outputs of the FMU. Default is `parser\utilities\SimulatorModelDescription.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 which is experimentell). Default is `dymola`.
-n	Flag to indicate if FMU needs an external execution tool to run. Options are `true` and `false`. Default is `false`.

The main functions of SimulatorToFMU are

- reading, validating, and parsing the Simulator 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 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.
- SimulatorToFMU can use OpenModelica to export a Simulator as an FMU. However the FMU cannot be loaded in Dymola or PyFMI because of shared libraries that cannot be loaded.

4.2 Outputs of SimulatorToFMU

The main output from running `SimulatorToFMU.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 SimulatorToFMU 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`".

CHAPTER

FIVE

BEST PRACTICE

This section explains to users the best practice in configuring a Simulator XML input file, and implementing the Python wrapper which will interface with the Simulator.

5.1 Configuring the Simulator XML input file

To export Simulator 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 `SimulatorModeldescritpion.xml` which shows such a file is provided in the `parser\utilities` installation folder of SimulatorToFMU. This template should be adapted to create new XML input file.

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SimulatorModelDescription
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  fmiVersion="2.0"
  modelName="Simulator"
  description="Input data for a Simulator FMU"
  generationTool="SimulatorToFMU">
  <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">
```

```
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   </SimulatorModelDescription>
```

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)

5.2 Configuring the Python Wrapper Simulator

To export Simulator as an FMU, the user needs to write a Python wrapper which will interface with the Simulator. The wrapper will be embedded in the FMU when the Simulator is exported and use at runtime on the target machine.

The user needs to extend the Python wrapper provided in `parser\utilities\Simulator.py` and implements the function `exchange`.

The following snippet shows the Simulator function.

```python
# Main Python function to be modified to integrate the main simulator.

def exchange(configuration_file, time, input_values,
             input_names, output_names, write_results):
    """
    Return  a list of output values from the Python-based Simulator.
    The order of the output values must match the order of the output names.

```

```
    :param configuration_file (String): filename for the model configurations
    :param time (Float): Current simulation time
    :param input_values (Floats): Input values
    :param input_names (Strings): Input names (same length as input_values)

    :param output_names (Strings): Output names
    :param write_results (Float): save results to a file (1.0 for saving, 0.0 else)

    Example:
        >>> configuration_file = 'config.json'
        >>> time = 0
        >>> input_names = ['VMAG_A', 'VMAG_B', 'VMAG_C', 'VANG_A', 'VANG_B', 'VANG_C']
        >>> input_values = [2520, 2520, 2520, 0, -120, 120]
        >>> output_names = ['IA', 'IAngleA', 'IB', 'IAngleB', 'IC', 'IAngleC']
        >>> write_results = 0
        >>> output_values = simulator(configuration_file, time, input_names,
                        input_values, output_names)
    """

    #######################################################################
    # EDIT AND INCLUDE CUSTOM CODE FOR TARGET SIMULATOR
    # ***Include body of the function used to compute the output values***
    # based on the inputs received by the simulator function
    # This function currently returns the input values.
    # This will need to be adapted so it return the output_values instead.
    # Assign the vector of output values with dummy values.
    output_values = [0]*len(output_names)
    #########################################################################

    return output_values

```

The arguments of the functions are in the next table

Arguments	Description
configuration_file	The Simulator model path/Simulator configuration file.
time	The current simulation model time.
input_values	The list of input values of the FMU.
input_names	The list of input names of the FMU.
output_values	The list of output values of the FMU.
output_names	The list of output names of the FMU.
write_results	A flag for writing results to a file.

Note:

- The function `exchange` must return a list of output values which match the order of the output names.
- The function `exchange` can be used to invoke external programs/scripts which do not ship with the FMU. In this situation, the FMU will must be exported with the option `<n>` set to `true`. The external programs/scripts will have to be installed on the target machine where the FMU is run. See *Creating an FMU* for details on command line options.
- Once Simulator.py is implemented, it must be saved under the same name and used as required argument for SimulatorToFMU.py

CHAPTER

SIX

DEVELOPMENT

The development site of this software is at https://github.com/tsnouidui/SimulatorToFMU.

To clone the master branch, type

git clone https://github.com/tsnouidui/SimulatorToFMU.git

CHAPTER

SEVEN

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

EIGHT

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

NINE

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
- Michael Wetter, Lawrence Berkeley National Laboratory

CHAPTER

TEN

DISCLAIMERS

CHAPTER

ELEVEN

COPYRIGHT AND LICENSE

11.1 Copyright

xxxxx.

PYTHON MODULE INDEX

p

INDEX