FMU Export of EnergyPlus

Release 1.0.6

LBNL - Simulation Research Group

March 09, 2016

CONTENTS

CHAPTER

ONE

INTRODUCTION

This user manual explains how to install and use EnergyPlusToFMU. EnergyPlusToFMU is a software package written in *Python* which allows users to export the building simulation program EnergyPlus version 8.0 or higher as a *Functional Mock-up Unit* (FMU) for co-simulation using the *Functional Mock-up Interface* (FMI) standard version 1.0. This FMU can then be imported into a variety of simulation programs that support the import of the *Functional Mock-up Interface* for co-simulation. This capability allows for instance to model the envelope of a building in EnergyPlus, export the model as an FMU, import and link the model with an HVAC system model developed in a system simulation tool such as the *Modelica* environment *Dymola*.

CHAPTER

TWO

DOWNLOAD

The EnergyPlusToFMU release includes scripts and source code to export EnergyPlus version 8.0 or higher as an FMU for co-simulation for Windows 32/64 bit, Linux 32/64 bit, and MAC OS X 64 bit.

To install EnergyPlusToFMU, follow the section Installation and Configuration.

2.1 Release 1.0.6 (March 09, 2016)

Download EnergyPlusToFMU-1.0.6.zip.

Release notes

This version improves the compliance of the Energyplus's FMU to the FMI 1.0 specification.

2.2 Release 1.0.5 (May 26, 2015)

Download EnergyPlusToFMU-1.0.5.zip.

Release notes

This version fixes a memory leak.

2.3 Release 1.0.4 (April 27, 2015)

Download EnergyPlusToFMU-1.0.4.zip.

Release notes

This version fixes a bug that occurred when a large number of variables were exchanged with the EnergyPlus's FMU.

2.4 Release 1.0.3 (May 23, 2014)

Download EnergyPlusToFMU-1.0.3.zip.

Release notes

This version contains a bug fix which was causing the EnergyPlus's FMU to write an incorrect RunPeriod.

2.5 Release 1.0.2 (March 20, 2014)

Download EnergyPlusToFMU-1.0.2.zip.

Release notes

This version contains a bug fix which was causing a division by zero because of an invalid timeStep.

2.6 Release 1.0.1 (December 13, 2013)

Download EnergyPlusToFMU-1.0.1.zip.

Release notes

This version contains many improvements and bug fixes.

2.7 Release 1.0.0 (November 01, 2013)

Download EnergyPlusToFMU-1.0.0.zip.

Release notes

First release that uses FMI version 1.0 for co-simulation.

CHAPTER

THREE

INSTALLATION AND CONFIGURATION

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

3.1 Software requirements

To export an EnergyPlus simulation as an FMU, EnergyPlusToFMU needs:

1. Python 2.6 or 2.7.
2. A C compiler and linker.
3. A C++ compiler and linker.

EnergyPlusToFMU has been tested on:

- Linux Ubuntu 12.04 (both 32- and 64-bit).
- Windows XP Professional.
- Mac OS X 10.5 and 10.6 (both 32- and 64-bit).

3.2 Installation

To install EnergyPlusToFMU, 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:

- `Scripts/` (Python scripts that create an FMU from an IDF file)
  - `win/` (batch files for Windows)
  - `linux/` (shell scripts for Linux)
  - `darwin/` (shell scripts for Mac OS X)
- `SourceCode/` (C and C++ source code for creating an EnergyPlus FMU)
  - `EnergyPlus/`
  - `Expat/`
  - `fmu-export-prep/`
  - `read-ep-file/`

- `utility/`

3.3 Configuration

EnergyPlusToFMU comes pre-configured to handle the most common system configurations. Usually, you can simply create an FMU, as described in Creating an FMU, and everything will work.

The remainder of this section provides tips for configuring EnergyPlusToFMU to match your particular system and needs. This may be needed in case problems arise. In addition, you may wish to reconfigure EnergyPlusToFMU, for example to optimize the FMU code, or to change the memory model under which it runs.

3.4 Batch files

EnergyPlusToFMU requires a C compiler and linker (in order to build the FMU), and a C++ compiler and linker (in order to build supporting tools). Configuring EnergyPlusToFMU mainly consists of:

- Choosing a different compiler or linker than the default.
- Passing the compiler or linker different options than the default.

EnergyPlusToFMU controls both aspects of the configuration using batch files on Windows, and shell scripts on Linux and Macintosh. Batch files have the extension `.bat`, while shell scripts have the `.sh` extension. For convenience, this guide refers to all such files as batch files.

EnergyPlusToFMU includes the following batch files:

Batch (shell) file	Purpose
`compile-c.bat`	Compiles a C source code file into an object file.
`link-c-exe.bat`	Links object files, generated by the C compiler, into an executable (i.e., a stand-alone application).
`link-c-lib.bat`	Links object files, generated by the C compiler, into a shared or dynamic link library.
`compile-cpp.bat`	Compiles a C++ source code file into an object file.
`link-cpp-exe.bat`	Links object files, generated by the C++ compiler, into an executable (i.e. a stand-alone application).
`test-c-exe.bat`	Automates a test of batch files `compile-c.bat` and `link-c-exe.bat`.

The EnergyPlusToFMU installation includes default versions of these batch files. However, the contents of these batch files depends on:

1. The operating system.
2. The compiler/linker environment (e.g., gcc or Microsoft Visual Studio).
3. The options desired (e.g., a 32-bit or 64-bit FMU).

As shown above, the *installation directory* contains a batch subdirectory for each of the supported platforms. Therefore to configure your system, you should first identify the appropriate subdirectory.

Note that Python detects your platform when it runs. Therefore if you are using an emulator or virtual machine (for example, Cygwin under Windows, or a Windows virtual machine on a Mac), you should look in the subdirectory corresponding to the emulated operating system.

Each system-specific batch subdirectory includes sample batch files. In addition to the default versions, some alternate versions also may be present. The alternate versions can be identified in two ways:

1. The file extension is "`.txt`", rather than "`.bat`" (or "`.sh`").

2. The file name indicates the associated options. For example, a file `compile-c-gcc-32bit.txt` is one possible version of `compile-c.sh`. It is specific to the gcc compiler/linker environment, and it generates 32-bit object files, even on a 64-bit operating system.

Note that the default batch file is an exact copy of one of the supplied alternate versions. For example, in the `win` subdirectory, the default batch file `compile-c.bat` is the same as `compile-c-mvs10.txt` (the version for Microsoft Visual Studio 10).

Once you have identified the appropriate batch file subdirectory, you can either:

- Test the existing batch files to determine whether a 32-bit or 64-bit FMU will be created. See *Running a basic test*.
- Modify the existing files, according to your particular needs. See *Modifying the batch files*.

3.5 Running a basic test

In addition to the batch files that run the compiler and linker, EnergyPlusToFMU includes a batch file that tests some of the existing batch files. In particular, batch file `test-c-exe.bat` tests the compiler and linker batch files which build an executable from `C` source code. Once these batch files work, it should not be difficult to make the other compiler and linker batch files work.

The batch file `test-c-exe.bat` runs the appropriate compiler and linker batch files, and tests the resulting executable. It reports its progress, and so should give information as to where problems occur, if any.

To run the basic test, open a command-line window (see Notation). Next, change to the appropriate batch file directory, and run the test batch file.

A sample session at the Windows DOS prompt:

```
# Change to the batch file directory.
> cd  epfmu_install_dir\Scripts\win

# Run the batch file.
> test-c-exe.bat
===== Checking for required files =====
===== Removing old output files =====
===== Running compiler =====
get-address-size.c
===== Running linker =====
===== Running output executable =====
== The address size, e.g., 32 or 64, should appear below ==
32
== The address size should appear above ==
===== Cleaning up =====
```

A sample session in a Linux command shell:

```
# Change to the batch file directory.
> cd  epfmu_install_dir/Scripts/linux

# Run the batch file.
# Note the "./" before the batch file name.
> ./test-c-exe.sh
===== Checking for required files =====
===== Removing old output files =====
===== Running compiler =====
===== Running linker =====
===== Running output executable =====
```

```
== The address size, e.g., 32 or 64, should appear below ==
32
== The address size should appear above ==
===== Cleaning up =====
```

A sample session in a MacOS Terminal window:

```
# Change to the batch file directory.
> cd  epfmu_install_dir/Scripts/darwin

# Run the batch file.
# Note the "./" before the batch file name.
> ./test-c-exe.sh
===== Checking for required files =====
===== Removing old output files =====
===== Running compiler =====
===== Running linker =====
===== Running output executable =====
== The address size, e.g., 32 or 64, should appear below ==
64
== The address size should appear above ==
===== Cleaning up =====
```

In the sessions shown above, the test batch file successfully ran to completion. If the test batch file fails at some point, then you will have to modify either the compiler or linker batch file, or possibly both. See *Modifying the batch files*.

The test batch file automates the individual steps described in *Checking compile-c.bat* and *Checking link-c-exe.bat* below. Therefore as you fine-tune your configuration, you may want to look to those sections for help.

3.6 Checking compile-c.bat

This section describes how to check the current version of batch file `compile-c.bat`. A later section describes *checking link-c-exe.bat*. Once these batch files work, it should not be difficult to make the other compiler and linker batch files work.

The check is to build and then run one of the EnergyPlusToFMU supporting applications. If the application builds and runs correctly, it reports whether your compiler generates 32-bit or 64-bit executables.

To check the compiler batch file, open a command-line window (see Notation). Next, change to the appropriate batch file directory, and run the compiler batch file. The compiler should produce an object file in the same directory.

A sample session at the Windows DOS prompt:

```
# Change to the batch file directory.
> cd  epfmu_install_dir\Scripts\win

# Run the batch file.
> compile-c.bat  ..\..\SourceCode\utility\get-address-size.c

# Check the object file.
> dir  *.obj
get-address-size.obj
```

A sample session in a Linux command shell:

```
# Change to the batch file directory.
> cd  epfmu_install_dir/Scripts/linux
```

```
# Run the batch file.
# Note the "./" before the batch file name.
> ./compile-c.sh  ../../SourceCode/utility/get-address-size.c

# Check the object file.
> ls  *.o
get-address-size.o
```

A sample session in MacOS Terminal:

```
# Change to the batch file directory.
> cd  epfmu_install_dir/Scripts/darwin

# Run the batch file.
# Note the "./" before the batch file name.
> ./compile-c.sh  ../../SourceCode/utility/get-address-size.c

# Check the object file.
> ls  *.o
get-address-size.o
```

In the sessions shown above, the compiler batch file successfully built the object file. Unfortunately, this is not always the case. Reasons for failure fall into a few broad categories:

- The commands in the batch file are wrong for your system. This is the most likely cause of failure. The subsections below give hints on providing an appropriate `compile-c.bat` batch file.
- You do not have permission to run the batch file. When you run the batch file, watch for output like "Permission denied" from the operating system. See *Troubleshooting permissions* below.
- The source code file is not on the specified path. When you run the batch file, watch for output like "No such file or directory", along with the name of the source code file. Check the *installation directory* structure, as specified above.
- The compiler did not accept some statement in source code file `get-address-size.c`. In this case, please contact the EnergyPlusToFMU team to report the problem.

Successfully compiling `get-address-size.c` does not completely test the compiler batch files. In particular:

- The batch file `compile-c.bat` must specify whether or not your compiler/linker environment provides a function called `memmove`. The simple application being tested here does not use `memmove`, so this aspect of the compiler batch file is not checked. See *Troubleshooting the memmove function* below.
- The batch file `compile-cpp.bat` must be configured for the C++ language, rather than the C language. Often no particular changes are required compared to `compile-c.bat` See the sample batch files in the standard distribution.

3.7 Checking link-c-exe.bat

Once you have successfully compiled the source code file `get-address-size.c` into an object file, the next step is to link the object file into an executable.

Working in the same subdirectory where you built the object file, run the linker batch file. In response, the linker should produce an executable, called `test.exe`, which you should be able to run.

A sample session at the Windows DOS prompt:

```
# Run the batch file.
> link-c-exe.bat  test.exe  get-address-size.obj

# Check the executable.
> dir  *.exe
test.exe

# Run the executable (32-bit system).
> test.exe
32
```

A sample session for both a Linux command shell and MacOS Terminal:

```
# Run the batch file.
# Note the "./" before the batch file name.
> ./link-c-exe.sh  test.exe  get-address-size.o

# Check the executable.
> ls  *.exe
test.exe

# Run the executable (32-bit system).
> ./test.exe
32
```

Again, the batch file may not work, for a few reasons:

- The commands in the batch file are wrong for your system. The subsections below give hints on providing an appropriate `link-c-exe.bat` batch file. In particular, if the linker complains of not being able to find the standard library function `printf`, see *Troubleshooting missing libraries* below.
- You do not have permission to run the batch file. When you run the batch file, watch for output like "Permission denied" from the operating system. See *Troubleshooting permissions* below.

Successfully building `test.exe` does not completely test the linker batch files. In particular:

- The batch file `link-c-lib.bat` has an additional complication. This batch file must link objects into a shared library. Creating a shared library generally requires passing a special switch or flag to the linker, such as `-shared`, `-dynamiclib`, or `/LD`. See the documentation for your development environment.
- The batch file `link-cpp-exe.bat` must link objects generated by the C++ compiler. This may require changing the linker switches or flags used in `link-c-exe.bat`.
- Building `test.exe` uses a single object file. The linker batch file must be able to handle a list of object files. The provided batch files all do this correctly. Since proper behavior depends on the operating system, rather than on the linker, no problems should arise here.

In all cases, comparing the batch files provided by the `EnergyPlusToFMU` installation may help solve some of these problems.

3.8 Modifying the batch files

This section gives general advice for editing your batch files, in case the default versions do not work on your system, or in case you want to modify or replace the default versions (for example, to change the optimization level, or to use a different compiler/linker).

Unfortunately, it is beyond the scope of this document to give full instructions on installing and using developer tools such as compilers and linkers.

The EnergyPlusToFMU tools only use the batch files named in the table shown in subsection *Batch files*. Thus, editing `compile-c-gcc.txt` will have no effect on how the FMU gets created. Only `compile-c.bat` affects how EnergyPlusToFMU compiles `C` source code files.

If a batch file does not work, it may simply be a matter of changing the directory path hard-coded in the batch file. For example, the batch files for Microsoft Visual Studio list several known locations for finding the Visual Studio program files. If your machine has Visual Studio installed in some other location (say, on the "`D:\`" drive rather than on "`C:\`"), then editing the batch file to point to the correct path may be all that is needed.

On most systems, the compiler also can act as the linker (or call the linker, filling in appropriate options). Therefore once you have your system's compiler working, try listing the same tool in the linker batch files.

If your compiler/linker environment comes with an integrated development environment (IDE), you often can use the IDE to determine appropriate flags for controlling the compiler and linker. For example, Microsoft Visual Studio is the standard IDE for Microsoft's C/C++ compilers, and the configuration panels in Visual Studio show the flags corresponding to each option. Therefore if you are having problems compiling a source code file with a provided batch file, try using the IDE to compile that source code file, and check what options the IDE uses.

3.9 Finding a compiler/linker (Unix)

The following tips for finding the compiler/linker apply to Unix-like environments, including Linux and MacOS.

Unix-like environments often define `cc` as a link to the standard C compiler, and `c++` as a link to the standard C++ compiler.

If you have a standard compiler on your search path, the `which` command will locate it. For example, entering the command:

```
> which gcc
```

will return the path to the `gcc` compiler, provided your system has it, and provided it is on the search path. Otherwise, `which gcc` will return nothing.

If you believe you have a certain compiler, but cannot find it on your search path, try the `find` command. For example, to locate the `icc` compiler, try:

```
> find /usr -name icc
> find /bin -name icc
> find /opt -name icc
> find / -name icc
```

The first three commands search specific directories that commonly contain developer tools (your system may not have all of these directories). The last command searches the entire directory tree (and may take quite a while).

The `find` command accepts wildcards. Put them in quote marks, in order to prevent the shell from operating on the wildcard. For example:

```
> find /usr -name "*icc*"
```

searches the `/usr` directory for any file whose name contains the string `icc`.

Finally, the `apropos` command may help:

```
> apropos compiler
```

will search your help files for information pertaining to compilers. Unfortunately, it may return many entries unrelated to compiling C and C++ source code.

3.10 Troubleshooting permissions (Unix)

Permissions problems arise on Unix-like systems. The batch files must have "execute" permission, meaning you are allowed to run the file as a set of commands. To check the permissions:

```
# Linux, MacOS:
> ls -lt  *.sh
-rwxr--r--  ...  link-c-lib.sh
-rwxr--r--  ...  link-c-exe.sh
-rwxr--r--  ...  compile-c.sh
-rwxr--r--  ...  link-cpp-exe.sh
-rwxr--r--  ...  compile-cpp.sh
```

All five of the default batch files should have "`-rwx`" at the beginning of the permissions block (indicating you are allowed to read, write, and execute the file). If not, then set the permissions:

```
# Linux, MacOS:
> chmod  u=rwx,g=r,o=r  *.sh
```

and try running the compiler batch file again.

3.11 Troubleshooting the `memmove` function

The batch file that runs the C compiler, `compile-c.bat`, needs to indicate whether or not your C compiler/linker environment provides a non-standard function called `memmove`. While `memmove` is non-standard in C, it is standard for C++. Therefore many C environments provide it as well.

If your C compiler/linker environment does provide `memmove`, then the batch file should pass the macro definition `HAVE_MEMMOVE` to the compiler. The included batch files show how to define a macro for various compilers.

If, on the other hand, your C compiler/linker environment does not provide `memmove`, then do not define the macro in the compiler batch file.

If you are not sure whether or not your system provides the function, simply watch for any errors while building your first FMU. If you fail to define `HAVE_MEMMOVE` when your system has it, the linker will complain about duplicate definitions of `memmove`. Conversely, if you define `HAVE_MEMMOVE` when your system does not have it, the linker will complain about not being able to find `memmove`.

3.12 Troubleshooting missing libraries

Some linkers need explicit instructions to link to library code. Libraries are needed to provide standard functions, such as `printf`, that are called by the EnergyPlusToFMU source code. If the linker emits an error message indicating it cannot find a particular function, then consult your development environment's documentation to determine which libraries it may need.

Note that specifying libraries is often somewhat arcane. For example, on Unix-like systems, to link a library `libm.a` typically requires the linker flag `-lm`. Furthermore, the order in which libraries are linked can matter, and you may need to add another flag to indicate the path(s) where the linker should search for libraries.

3.13 Uninstallation

To uninstall EnergyPlusToFMU, delete the *installation directory*.

CHAPTER

FOUR

CREATING AN FMU

This chapter describes how to create a Functional Mockup Unit, starting from an EnergyPlus IDF file. It assumes you have followed the Installation and Configuration instructions, and that you have created an IDF file 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 EnergyPlusToFMU tool is:

```
> python  <path-to-scripts-subdir>EnergyPlusToFMU.py  -i <path-to-idd-file>  \
  -w <path-to-weather-file>  <path-to-idf-file>
```

For example:

```
# Windows:
> python  scriptDir\EnergyPlusToFMU.py  -i C:\eplus\Energy+.idd  test.epw  test.idf

# Linux, MacOS:
> python  scriptDir/EnergyPlusToFMU.py  -i ~/eplus/Energy+.idd  test.epw  test.idf
```

where `scriptDir` is the path to the scripts directory of EnergyPlusToFMU. Typically this is the `Scripts/` subdirectory of the installation directory. See Installation and Configuration for details.

All file paths, including those to the weather, IDD, and IDF files, can be absolute or relative. For readability, the rest of these instructions omit the paths to the script and input files.

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

Script `EnergyPlusToFMU.py` supports the following command-line switches:

option <argument>	Purpose
-i <path-to-idd-file>	Use the named Input Data Dictionary (required).
-w <path-to-weather-file>	Add the named weather file to the FMU [optional].
-d	Print diagnostics [optional]. Produces a status line for every major action taken by the EnergyPlusToFMU tools. This option may be helpful for debugging.
-L	Litter, that is, do not clean up intermediate files [optional]. Typically the EnergyPlusToFMU tools delete most of the intermediate files that ultimately get packaged into the FMU. This option lets you inspect intermediate output.

The switches may be given in any order. However, all must appear before the name of the IDF file. For repeated switches such as `-i` or `-w`, the last one specified will be used.

For example:

```
# Windows:
> python  EnergyPlusToFMU.py  -d  -i C:\eplus\Energy+.idd  test.idf
```

4.2 Output

The main output from running `EnergyPlusToFMU.py` consists of an FMU, named after the IDF file (e.g., `test.fmu` in the examples given above). The FMU is written to the current working directory, that is, in the directory from which you entered the command.

The FMU is complete and self-contained. Any secondary output from running the EnergyPlusToFMU tools can be deleted safely. Secondary output includes:

- A utility executable, with the base name `idf-to-fmu-export-prep`. This executable will appear in your current working directory. If deleted, it will be rebuilt on the next run of EnergyPlusToFMU. Note that the full name of this executable depends on the operating system. This allows users with dual-boot or virtual machines to work in the same directory. The full names are:
  - `idf-to-fmu-export-prep-win.exe` on Windows.
  - `idf-to-fmu-export-prep-linux` on Linux.
  - `idf-to-fmu-export-prep-darwin` on Macintosh OS X.
- Compiled Python files, with the extension "`.pyc`". They are written to the script directory. These files speed up Python the next time you run the EnergyPlusToFMU tools, and may be deleted.

If the EnergyPlusToFMU tool fails, you may also see intermediate files, including:

- The configuration files for the FMU, `variables.cfg` and `modelDescription.xml`.
- A utility executable `util-get-address-size.exe`. This program is rebuilt every time you run the EnergyPlusToFMU tools (to ensure it uses the most recent compiler/linker batch files, as described in Installation and Configuration).
- Build directories, named like `bld-*`.
- A shared library, named like `*.dll` or `*.so` or `*.dylib`, depending on the system.
- A log file, `output.log`, containing error messages from `idf-to-fmu-export-prep`.

All these intermediate files can be deleted.

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`".

4.3 Troubleshooting

To check whether `EnergyPlusToFMU.py` has run correctly, look for an FMU in your current working directory. If you do not get an FMU, there will be some error output, indicating the nature of the problem.

The error message should be explicit enough to guide you to the source of the problem. If not, consider the following hints.

If you have successfully made an FMU in the past, the problem is most likely with your IDF file. Try running the export-preparation application directly on your IDF file:

```
# Windows:
> idf-to-fmu-export-prep-win.exe  Energy+.idd  test.idf

# Linux:
#   Note the "./" before the name of the application.
> ./idf-to-fmu-export-prep-linux  Energy+.idd  test.idf

# MacOS:
#   Note the "./" before the name of the application.
> ./idf-to-fmu-export-prep-darwin  Energy+.idd  test.idf
```

If running the export-preparation application as shown above works correctly, it produces two files, `modelDescription.xml` and `variables.cfg`. Otherwise, it should produce an error message, which should also be echoed to an output file `output.log`.

Note that the export-preparation application processes only parts of the IDF file. It does not attempt to identify modeling errors, or problems in IDF sections that do not relate to the FMU. Therefore EnergyPlus may fail to run an IDF file, even if the export-preparation application handles it successfully.

If you do not find the export-preparation application in your working directory, then EnergyPlusToFMU did not advance to creating the application. Therefore you should check the configuration, according to the instructions in Installation and Configuration.

If the export-preparation application runs, then try turning on option `-d` when running `EnergyPlusToFMU.py`. By announcing each major step before it is taken, this option helps to localize the problem.

CHAPTER

FIVE

USAGE OF ENERGYPLUS AS AN FMU

The following items need to be observed when importing an FMU that contains EnergyPlus:

1. A tool that imports the FMU needs to make sure that the version of EnergyPlus which has been used to export the FMU is (a) installed and (b) added to the system path environment variable. Otherwise, the simulation will fail with an error. If EnergyPlus has been properly added to the system path environment variable, then it can be started from any DOS prompt on Windows, command shell console on Linux, or Terminal window on Mac OS by typing `RunEPlus.bat` on Windows and `runenergyplus` on Linux or Mac OS. [1]

2. The `Number of Timesteps per Hour` in EnergyPlus must be equal to the sampling time of the FMU. For example, consider the following EnergyPlus IDF snippet:

```
Timestep,
6;        !- Number of Timesteps per Hour
```

   Then, a tool that imports the FMU must synchronize it every 10 minutes. Otherwise, the simulation will stop with an error. [2]

3. EnergyPlus contains the object `RunPeriod`. The start and end day of this object is ignored. [3] However, the entry `Day of Week for Start Day` will be used. For example, consider the following IDF snippet:

```
RunPeriod,          ! Winter Simulation
Winter Simulation, !- Name
1,                  !- Begin Month
2,                  !- Begin Day of Month
3,                  !- End Month
31,                 !- End Day of Month
Monday,             !- Day of Week for Start Day
Yes,                !- Use Weather File Holidays and Special Days
Yes,                !- Use Weather File Daylight Saving Period
No,                 !- Apply Weekend Holiday Rule
Yes,                !- Use Weather File Rain Indicators
Yes;                !- Use Weather File Snow Indicators
```

   This IDF snippet declares January 2 to be a Monday. Hence, if an FMU is simulated with start time equal to 3 days, then the first day of the simulation will be Tuesday. There should only be one instance of RunPeriod in the IDF input file.

4. A tool that imports the FMU must specify the start and stop time in seconds at initialization. The start and stop time cannot have the value "infinity".

5. The weather file which comes along with an FMU is used to determine if the year is a `leap year`. If no weather file is included in the FMU, then the assumption is that the year is not a `leap year`.

[1] This is because the FMU implements the FMI for co-simulation in the Tool Coupling scenario.
[2] This is because the External Interface in EnergyPlus synchronizes the data at the zone time step which is constant throughout the simulation.
[3] This is because a tool that imports an FMU has its own definition of start time and stop time.

6. During the warm-up period and the autosizing of EnergyPlus, no data exchange occurs between the FMU and the master program. Thus, inputs of EnergyPlus remain constant during these times and are equal to the initial values specified in the IDF input file.
7. The simulation results are saved in a result folder which is created in the current working directory. The name of the result folder is `Output_EPExport_xxx`, where `xxx` is the FMU model `instanceName` as defined in the FMI specifications.

CHAPTER

SIX

MATHEMATICAL DESCRIPTION

This section describes the algorithm for exchanging data between EnergyPlus, packaged as an FMU, and an external program.

Suppose we have a system with two simulation programs. Let simulation program 1 be EnergyPlus, the slave simulation program, which is packaged as an FMU for co-simulation; and let simulation program 2 be the master simulation program which supports the import of FMU for co-simulation. Suppose each program solves an initial-value ordinary differential equation that is coupled to the differential equations of the other program.

Let N denote the number of time steps and let k denote the time step. We will use the subscripts 1 and 2 to denote the state variable and the function that computes the next value of the state variable of the simulator 1 and 2, respectively. Then program 1 computes the sequence

$$x_1(k+1) = f_1(x_1(k), x_2(k)),$$

and, similarly, program 2 computes the sequence

$$x_2(k+1) = f_2(x_2(k), x_1(k)),$$

with initial conditions $x_1(0) = x_{1,0}$ and $x_2(0) = x_{2,0}$.

To advance from time k to $k+1$, each program uses its own integration algorithm. At the end of the time step, program 1 sends its new state $x_1(k+1)$ to program 2, and receives the state $x_2(k+1)$ from program 2. The same procedure is done with the program 2. Program 2, which is the master simulation program, imports the FMU, and manages the data exchange between the two programs. In comparison to numerical methods of differential equations, this scheme is equal to an explicit Euler integration, which is an integration algorithm that solves an ordinary differential equation with specified initial values,

$$dx/dt = h(x),$$

$$x(0) = x_0,$$

on the time interval $t \in [0, 1]$, using he following steps:

Step 0:

Initialize counter $k = 0$ and number of steps, $N > 0$.

Set initial state $x(k) = x_0$ and set time step $\Delta t = 1/N$.

Step 1:

Compute new state $x(k+1) = x(k) + h(x(k))\Delta t$.

Replace k by $k+1$.

Step 2:

If $k = N$ stop, else go to Step 1.

However, this scheme does not require each simulation tool to use the explicit Euler method for its internal time integration algorithm; the analogy to explicit Euler applies only to the data exchange between the programs. In the situation where the differential equation is solved using co-simulation, the above algorithm becomes

Step 0:

Initialize counter $k = 0$ and number of steps, $N > 0$.

Set initial state $x_1(k) = x_{1,0}$ and $x_2(k) = x_{2,0}$. Set the time step $\Delta t = 1/N$.

Step 1:

Compute new states

$x_1(k+1) = x_1(k) + f_1(x_1(k), x_2(k))\Delta t$, and

$x_2(k+1) = x_2(k) + f_2(x_2(k), x_1(k))\Delta t$.

Replace k by $k + 1$.

Step 2:

If $k = N$ stop, else go to Step 1.

CHAPTER

SEVEN

BEST PRACTICE

This section explains to users the best practice in configuring an EnergyPlus model for an FMU.

To export EnergyPlus as an FMU, four objects have been added to the EnergyPlus data structure. These objects are:

- The `ExternalInterface:FunctionalMockupUnitExport:From:Variable` object, which is used to map the outputs of the FMU to the EnergyPlus objects `Output:Variable` and `EnergyManagementSystem:OutputVariable`.
- The `ExternalInterface:FunctionalMockupUnitExport:To:Schedule`, `ExternalInterface:FunctionalMockupUnitExport:To:Actuator`, and `ExternalInterface:FunctionalMockupUnitExport:To:Variable`, which are used to map the inputs of the FMU to EnergyPlus schedules, EMS actuators, and variables.

These objects are described in the Input/Output reference of the EnergyPlus manual.

7.1 Configuring an EnergyPlus model which uses the `Schedule` object

Suppose we like to export an EnergyPlus model of a room with an ideal HVAC system that adds heating or cooling to the zone as schedules, to maintain a certain room temperature.

Such an EnergyPlus model could be exported as an FMU with one input and one output. The input of the FMU will write to the heating/cooling time schedule, whereas the output of the FMU will read the room dry-bulb temperature.

The Energyplus model needs to contain the following three items:

- An object that instructs EnergyPlus to activate the external interface.
- EnergyPlus objects that read inputs of the FMU and send the values to EnergyPlus.
- EnergyPlus objects that read data from EnergyPlus and send the values to the outputs of the FMU.

The code below shows how to declare these objects in the IDF. To activate the external interface, we use:

```
    ExternalInterface,           !- Object to activate external interface
    FunctionalMockupUnitExport;  !- Name of external interface
```

To define the input of the FMU, we use:

```
    ExternalInterface:FunctionalMockupUnitExport:To:Schedule,
    FMU_OthEqu_ZoneOne,      !- Name
    Any Number,              !- Schedule Type Limits Names
    Q,                       !- FMU Variable Name
    0;                       !- Initial Value
```

To define the output of the FMU, we use:

```
ExternalInterface: FunctionalMockupUnitExport:From:Variable,
ZONE ONE,                  !- Output:Variable Index Key Name
Zone Mean Air Temperature, !- Output:Variable Name
TRooMea;                   !- FMU Variable Name
```

Along with the FMU's output definition, the EnergyPlus output variable which corresponds to the FMU output needs to be specified in the IDF file:

```
Output:Variable,
ZONE ONE,                    !- Key Value
Zone Mean Air Temperature,   !- Variable Name
TimeStep;                    !- Reporting Frequency
```

These specifications are used in the example that is available in `Examples/Schedule`.

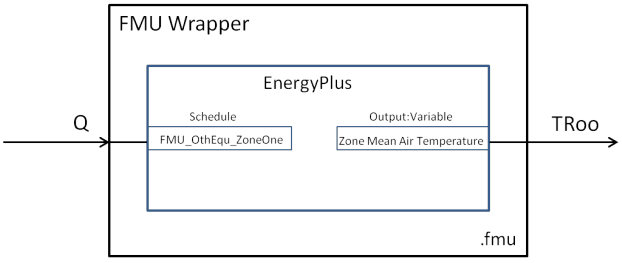

Fig. 7.1: Configuration of an EnergyPlus model which uses the `Schedule`.

7.2 Configuring an EnergyPlus model which uses the `EMS Actuator` object

Suppose we like to export an EnergyPlus model of a room that has a shading controller. The shading controller requires as input the shading actuation signal `yShade`, and has as outputs the outside temperature `TRoo` and the solar irradiation `ISolExt` that is incident on the window.

We will export such an EnergyPlus model as an FMU with one input and two outputs. The input of the FMU will write to the shading actuation signal, whereas the outputs will read the outside temperature `TRoo` and the solar irradiation `ISolExt`.

The code below shows how to declare these objects in the IDF. To activate the external interface, we use:

```
ExternalInterface,           !- Object to activate external interface
FunctionalMockupUnitExport;  !- Name of external interface
```

To define the input of the FMU, we use:

```
ExternalInterface:FunctionalMockupUnitExport:To:Actuator,
Zn001_Wall001_Win001_Shading_Deploy_Status,  !- Name
Zn001:Wall001:Win001,                       !- Actuated Component Unique Name
Window Shading Control,                     !- Actuated Component Type
Control Status,                             !- Actuated Component Control Type
yShade,                                     !- FMU Variable Name
6;                                          !- Initial Value
```

To define the outputs of the FMU, we use:

```
ExternalInterface:FunctionalMockupUnitExport:From:Variable,
WEST ZONE,                                  !- Output:Variable Index Key Name
Zone Mean Air Temperature,                  !- Output:Variable Name
TRoo;                                       !- FMU Variable Name

ExternalInterface:FunctionalMockupUnitExport:From:Variable,
Zn001:Wall001:Win001,                                          !- Output:Variable Index Key Name
Surface Outside Face Incident Solar Radiation Rate per Area,   !- Output:Variable Name
ISolExt;                                                       !- FMU Variable Name
```

Along with the FMU's outputs definition, the EnergyPlus output variables which correspond to the FMU outputs need to be specified in the IDF file:

```
Output:Variable,
Zn001:Wall001:Win001,                                          !- Key Value
Surface Outside Face Incident Solar Radiation Rate per Area,   !- Variable Name
TimeStep;                                                      !- Reporting Frequency

Output:Variable,
WEST ZONE,                          !- Key Value
Zone Mean Air Temperature,          !- Variable Name
TimeStep;                           !- Reporting Frequency
```

These specifications are used in the example that is available in `Examples/Actuator`.

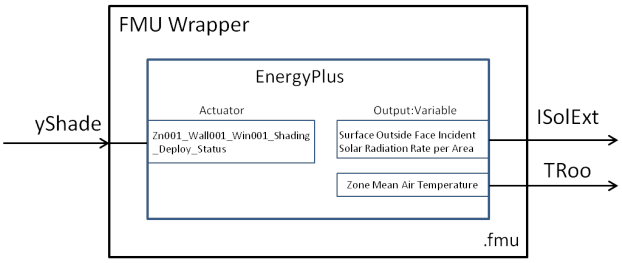

Fig. 7.2: Configuration of an EnergyPlus model which uses the `EMS Actuator`.

7.3 Configuring an EnergyPlus model which uses the `EMS Variable` object

This configuration is almost the same as in the previous example with the only difference being that the shading actuation signal will be written to an EMS variable `Shade_Signal` that can be used in an EMS program.

To define the input of the FMU, we use:

```
ExternalInterface:FunctionalMockupUnitExport:To:Variable,
Shade_Signal,           !- Name
yShade,                 !- FMU Variable Name
6;                      !- Initial Value
```

These specifications are used in the example that is available in `Examples/Variable`.

Please see the Input/Output reference of the EnergyPlus manual.

Please read Installation and Configuration to see how to generate an FMU.

CHAPTER

EIGHT

DEVELOPMENT

The development site of this software is at https://corbu.lbl.gov/svn/fmu/EnergyPlus/export/.

To check out the trunk, type

svn checkout https://corbu.lbl.gov/svn/fmu/EnergyPlus/export/trunk/

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)
```

For example, to see a list of files in the current directory on Linux or MacOS:

```
# List all Python files.
> ls  *.py
EnergyPlusToFMU.py   makeExportPrepApp.py   ...
```

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

HELP

Before submitting a bug report,

- Check https://corbu.lbl.gov/trac/fmu/browser/EnergyPlus#export/trunk if the bug is already known.
- Make sure that it is a bug of the FMU implementation and not a bug of the EnergyPlus model. EnergyPlus bugs should be isolated so that they do not require use of an FMU, and then sent to the EnergyPlus help desk at http://energyplus.helpserve.com/.

When submitting a bug report, please provide:

- A model that is as small as possible and still reproduces your error. The chances of quickly finding and fixing a bug are much higher if the bug is part of a small test problem. In addition, creating a small test problem may help finding the root cause for the bug yourself, or realizing that there is no bug at all, and hence your problem can be solved much sooner.
- A description of the expected and the observed results.
- Information about the operating system and the EnergyPlus version.

To report the bug, send email to https://groups.google.com/group/energyplus-fmu. This is an open group and everyone can join it. No invitation is needed.

Known Issues

Release 1.0.0

Dymola 2014 cannot import EnergyPlus as an FMU. The reason is a bug in *Dymola* 2014 which does not pass the path to the resources location folder to the FMU when invoking the `fmiInstantiateSlave()` method. This information is needed by the FMU for its correct execution. A workaround will be to manually specify the path to the resources location in the `fmiInstantiateSlave()` method in `main.c`. `main.c` can be found in the `SourceCode` folder of the EnergyPlusToFMU installation.

CHAPTER

ELEVEN

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

TWELVE

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. DE-AC02-05CH11231.

The following people contributed to the development of this program:

- Thierry Stephane Nouidui, Lawrence Berkeley National Laboratory
- David M. Lorenzetti, Lawrence Berkeley National Laboratory
- Michael Wetter, Lawrence Berkeley National Laboratory

CHAPTER

THIRTEEN

DISCLAIMERS

CHAPTER

FOURTEEN

COPYRIGHT AND LICENSE

Functional Mock-up Unit Export of EnergyPlus 2015, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from the U.S. Department of Energy). All rights reserved.

If you have questions about your rights to use or distribute this software, please contact Berkeley Lab's Technology Transfer Department at TTD@lbl.gov. referring to "Functional Mock-up Unit Export of EnergyPlus (LBNL Ref 2013-088)".

NOTICE: This software was produced by The Regents of the University of California under Contract No. DE-AC02-05CH11231 with the Department of Energy. For 5 years from November 1, 2012, the Government is granted for itself and others acting on its behalf a nonexclusive, paid-up, irrevocable worldwide license in this data to reproduce, prepare derivative works, and perform publicly and display publicly, by or on behalf of the Government. There is provision for the possible extension of the term of this license. Subsequent to that period or any extension granted, the Government is granted for itself and others acting on its behalf a nonexclusive, paid-up, irrevocable worldwide license in this data to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so. The specific term of the license can be identified by inquiry made to Lawrence Berkeley National Laboratory or DOE. Neither the United States nor the United States Department of Energy, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any data, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

Copyright (c) 2015, The Regents of the University of California, Department of Energy contract-operators of the Lawrence Berkeley National Laboratory. All rights reserved.

1. 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 copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the 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.

2. 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.

3. 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.

NOTE: This license corresponds to the "revised BSD" or "3-clause BSD" License and includes the following modification: Paragraph 3. has been added.

D

F

M

P