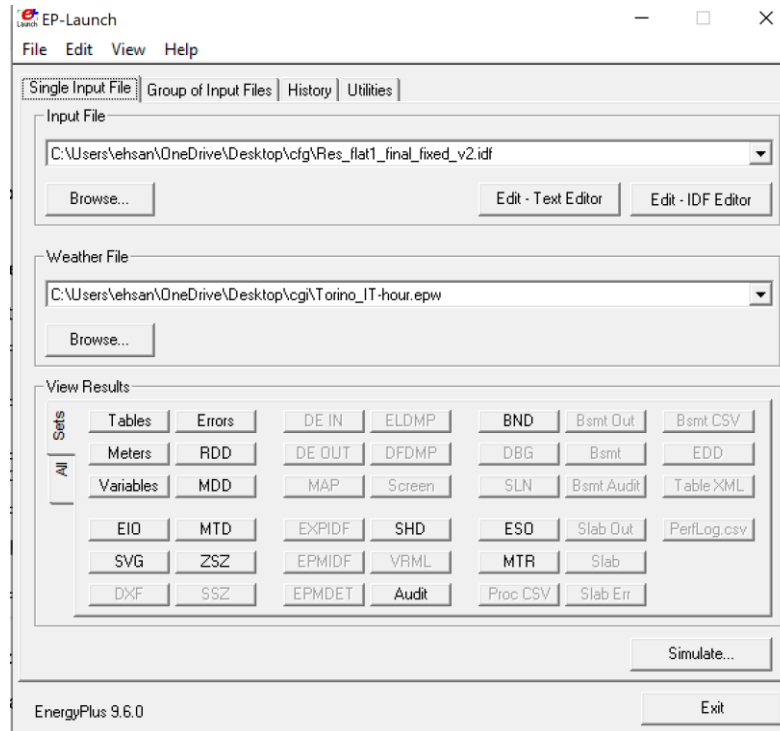


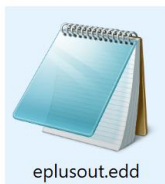
How to create cfg file:

Step 1: Generate .rdd, .mdd, and .edd files



After your first EnergyPlus simulation, these three files appear in the output directory:

File	Description
eplusout.rdd	All available output variables (Output:Variable)
eplusmtr.mdd	All available meter variables (Output:Meter)
eplusout.edd	Lists all EMS-compatible actuators with component name/type/control type



eplusout.edd



eplusout.mdd



eplusout.rdd



flat1-optimal-5.9.
idf



sensorsactuators
_complete.cfg



simulation_outp
uts.csv



Torino_IT-hour.e
pw

Step 2: Pick sensors from .rdd and .mdd

Example sensor variables:

- Zone Mean Air Temperature
- Zone Operative Temperature
- Zone Air Relative Humidity
- Electricity:Facility (from .mdd, meter = true)

Step 3: Pick actuators from .edd

You will find entries like:

Component Name = BLOCCO1:ZONA2

Component Type = Zone Temperature Control

Control Type = Cooling Setpoint

Use these three to define actuators in your config.

Step 4: Write the JSON config (cfg) file

Based on the .rdd, .edd, and .mdd findings, structure your config like this:

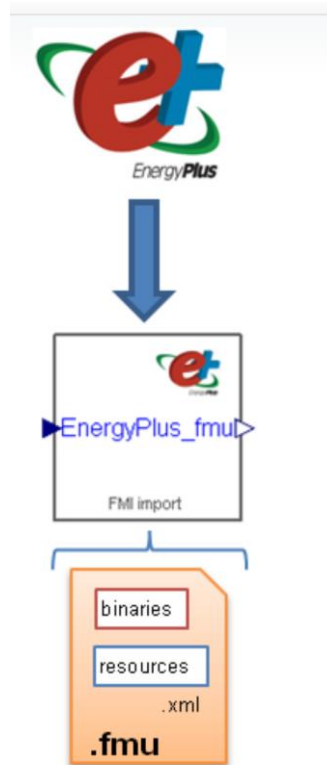
```
json
CopyEdit
{
  "sensors": [...],
  "actuators": [...]
}
```

Generating an FMU from an EnergyPlus Model Using Docker on Windows

1. Introduction

Energy modeling is a crucial step in analyzing the thermal and energy performance of buildings. EnergyPlus, a powerful simulation engine developed by the U.S. Department of Energy, allows for detailed simulation of building systems and performance. In order to integrate an EnergyPlus model with external controllers (e.g., for co-simulation with Python, MATLAB, or reinforcement learning agents), the model must be exported as a **Functional Mock-up Unit (FMU)** that adheres to the **FMI 2.0** standard.

Since the official EnergyPlusToFMU tool only supports Linux, this project utilizes **Docker** in Windows to create a Linux container environment in which the conversion from IDF to FMU can be performed seamlessly.



2. Tools and Required Files

Software Components

- Docker Desktop (on Windows)
- EnergyPlus 9.6.0 for Linux (installer)
- EnergyPlusToFMU v3.1.0
- Python 3.x (inside the container)

Required Files

File Name	Description
flat1.idf	EnergyPlus input model (building definition)
weather.epw	EPW weather file (for Torino, Italy)
sensorsactuators_complete.cfg	JSON configuration of sensors and actuators
fix_idf.py	Python script to modify IDF for FMU compatibility
EnergyPlus-9.6.0-Linux-Ubuntu.sh	Linux installer for EnergyPlus 9.6
EnergyPlusToFMU-v3.1.0.zip	Official FMU export tool from LBNL

3. Procedure

3.1 Project Directory Setup

A working directory named `energyplus_fmu` was created at:

`makefile`

```
C:\Users\ehsan\OneDrive\Documents\ICT\Building\project\our project\energyplus_fmu
```

All required files listed above were placed in this folder.

3.2 Creating the Dockerfile

A custom Dockerfile was written to build a Linux-based container with all dependencies pre-installed. This includes EnergyPlus 9.6.0 and the EnergyPlusToFMU tool. The working directory inside the container was set to `/workspace`.

3.3 Building the Docker Image

The image was built from the terminal using:

`powershell`

```
docker build -t eplus-fmu
```

This command generated a Docker image named `eplus-fmu`.

3.4 Running the Container

The container was started with access to the local project folder:

powershell

```
docker run -it --rm -v "${PWD}:/workspace" eplus-fmu
```

Inside the container, the /workspace directory mirrors the Windows folder.

3.5 Preparing the IDF File

A custom Python script fix_idf.py was executed to enhance the original IDF file with the following components:

- ExternalInterface block
- Output:Variable objects for sensors
- ExternalInterface:From for sensor export
- ExternalInterface:To for actuator import

The modified file was saved as flat1-fmu.idf.

3.6 FMU Conversion

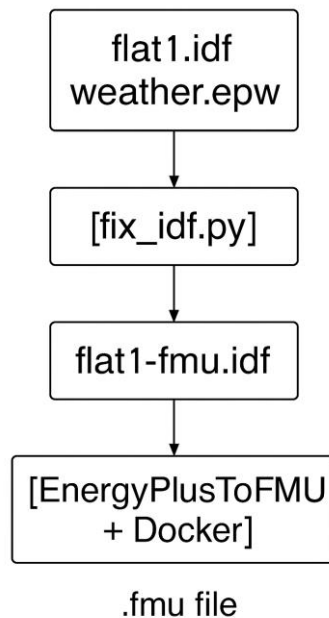
The FMU was created using the following command from the tool's directory:

bash

```
cd /opt/EnergyPlusToFMU
python Scripts/EnergyPlusToFMU.py \
-i /usr/local/EnergyPlus-9-6-0/Energy+.idd \
-w /workspace/weather.epw \
-a 2 \
```

/workspace/flat1-fmu.idf

The result was a .fmu file containing the model and interface configuration, ready for co-simulation.



FMU Validation Using FMPy

After generating the .fmu file, we validated its structure and contents using the FMPy Python library. The FMU was extracted and the modelDescription.xml file was parsed to inspect variable names, causality, and types.

Python Code Snippet:

```
from fmpy import extract, read_model_description
import os

# مسیر فایل FMU
fmu_path = '/content/drive/MyDrive/flat1_optimal_5_9_fmu_fixed.fmu' # اگر روی گوگل
# درایو هست، مسیر کامل بده

# در پوشه موقت FMU مرحله 1: استخراج
```

```

extracted_dir = extract(fmu_path)

print("✅ FMU extracted to:", extracted_dir)

# از مسیر درست modelDescription.xml: خواندن فایل
model_desc_path = os.path.join(extracted_dir, 'modelDescription.xml')

desc = read_model_description(model_desc_path)

print(f"\n📄 Model name: {desc.modelName}")

print("🔍 Variable List (Name | Causality | Type):\n")

for v in desc.modelVariables:

    print(f"- {v.name:45} | {v.causality:6} | {v.type}")

```

Output:

BLOCCO1:ZONA1_HeatingSetpoint	input Real
- BLOCCO1:ZONA1_CoolingSetpoint	input Real
- BLOCCO1:ZONA2_HeatingSetpoint	input Real
- BLOCCO1:ZONA2_CoolingSetpoint	input Real
- BLOCCO1:ZONA3_HeatingSetpoint	input Real
- BLOCCO1:ZONA3_CoolingSetpoint	input Real
- BLOCCO1:ZONA4_HeatingSetpoint	input Real
- BLOCCO1:ZONA4_CoolingSetpoint	input Real
- BLOCCO1:ZONA5_HeatingSetpoint	input Real
- BLOCCO1:ZONA5_CoolingSetpoint	input Real
- BLOCCO1:ZONA6_HeatingSetpoint	input Real
- BLOCCO1:ZONA6_CoolingSetpoint	input Real
- BLOCCO1:ZONA7_HeatingSetpoint	input Real
- BLOCCO1:ZONA7_CoolingSetpoint	input Real
- BLOCCO1:ZONA1_MeanAirTemp	output Real
- BLOCCO1:ZONA1_OperativeTemp	output Real
- BLOCCO1:ZONA1_RelHumidity	output Real
- BLOCCO1:ZONA1_HeatingEnergy	output Real
- BLOCCO1:ZONA1_CoolingEnergy	output Real
- BLOCCO1:ZONA1_PeopleCount	output Real
- BLOCCO1:ZONA2_MeanAirTemp	output Real
- BLOCCO1:ZONA2_OperativeTemp	output Real
- BLOCCO1:ZONA2_RelHumidity	output Real

- BLOCCO1:ZONA2_HeatingEnergy	output Real
- BLOCCO1:ZONA2_CoolingEnergy	output Real
- BLOCCO1:ZONA2_PeopleCount	output Real
- BLOCCO1:ZONA3_MeanAirTemp	output Real
- BLOCCO1:ZONA3_OperativeTemp	output Real
- BLOCCO1:ZONA3_RelHumidity	output Real
- BLOCCO1:ZONA3_HeatingEnergy	output Real
- BLOCCO1:ZONA3_CoolingEnergy	output Real
- BLOCCO1:ZONA3_PeopleCount	output Real
- BLOCCO1:ZONA4_MeanAirTemp	output Real
- BLOCCO1:ZONA4_OperativeTemp	output Real
- BLOCCO1:ZONA4_RelHumidity	output Real
- BLOCCO1:ZONA4_HeatingEnergy	output Real
- BLOCCO1:ZONA4_CoolingEnergy	output Real
- BLOCCO1:ZONA4_PeopleCount	output Real
- BLOCCO1:ZONA5_MeanAirTemp	output Real
- BLOCCO1:ZONA5_OperativeTemp	output Real
- BLOCCO1:ZONA5_RelHumidity	output Real
- BLOCCO1:ZONA5_HeatingEnergy	output Real
- BLOCCO1:ZONA5_CoolingEnergy	output Real
- BLOCCO1:ZONA5_PeopleCount	output Real
- BLOCCO1:ZONA6_MeanAirTemp	output Real
- BLOCCO1:ZONA6_OperativeTemp	output Real
- BLOCCO1:ZONA6_RelHumidity	output Real
- BLOCCO1:ZONA6_HeatingEnergy	output Real
- BLOCCO1:ZONA6_CoolingEnergy	output Real
- BLOCCO1:ZONA6_PeopleCount	output Real
- BLOCCO1:ZONA7_MeanAirTemp	output Real
- BLOCCO1:ZONA7_OperativeTemp	output Real
- BLOCCO1:ZONA7_RelHumidity	output Real
- BLOCCO1:ZONA7_HeatingEnergy	output Real
- BLOCCO1:ZONA7_CoolingEnergy	output Real
- BLOCCO1:ZONA7_PeopleCount	output Real

Result:

- Model successfully extracted
- Variables correctly listed with proper input/output causality
- Confirmed alignment with the configured sensors and actuators

This ensures the FMU is correctly structured and ready for co-simulation.

4. Conclusion

By using Docker, we successfully generated an FMU from an EnergyPlus model within a Windows environment, bypassing the native Linux-only restrictions of the EnergyPlusToFMU tool. This method enables integration with advanced controllers, allows for scalable deployment, and supports real-time simulation frameworks. The FMU can now be used for smart control experiments, such as reinforcement learning-based HVAC regulation or model predictive control (MPC).

This workflow is portable, reproducible, and ideal for research in building energy management and smart grid co-simulation.