# SOFIRpy

*Release 0.1.0*

**Daniele Inturri**

**Jan 24, 2023**

# CONTENTS

CHAPTER

ONE

# OVERVIEW

Co- **S**imulation **O**f **F**unctional Mock-up Units (FMU) with **I**ntegrated **R**esearch Data Management (SOFIRpy) is a python package that lets you co-simulate FMUs with custom models written in python.

The package provides functionalities to:

- Export Dymola and OpenModelica models as a FMU
- Co-simulate FMUs with custom written models in python
- Store data and meta data of the simulation inside a hdf5 file

## 1.1 Use Cases

**Control of physical systems.**

Tools like OpenModelica and Dymola are excellent tools for modelling complex physical systems. These models can be exported as FMUs and co-simulated with custom written controllers in python. Thereby Python offers extensive tools for the development of control systems, such as machine learning toolboxes, so that complex control strategies can be implemented.

**Automation of Simulation workflow**

Easy-to-use functionalities for storing data and meta data in a hdf5 file allows to implement custom automated simulation workflows. (Example illustrating possible workflow coming in the future).

CHAPTER

TWO

# INSTALLATION

sofirpy can be installed from PyPI using the following command.

```
pip install sofirpy
```

---

**Note:** The most recent version of OMPython on PyPI has a bug whereby the model export can fail. To get the most recent version of OMPython enter the following commands in the command window.

First uninstall the current version of OMPython:

```
pip uninstall OMPython
```

Then install the most recent version from GitHub.

```
pip install git+https://git@github.com/OpenModelica/OMPython.git
```

---

CHAPTER

THREE

# GETTING STARTED

To start using sofirpy, import the package:

```
import sofirpy
```

The following 3 examples demonstrate how to export a Modelica model as a FMU, simulate a FMU and a controller and how to initialize a project.

## 3.1 Exporting a modelica model

sofirpy allows to export a OpenModelica and Dymola model as a FMU.

**Exporting a OpenModelica model**

```
from pathlib import Path
from sofirpy import export_open_modelica_model

dir_path = Path(__file__).parent.parent
model_path = dir_path / "DC_Motor.mo"
model_name = "DC_Motor"
output_direcotry = dir_path

export_open_modelica_model(model_path, model_name, dir_path)
```

**Exporting a Dymola model and importing parameters**

```
from pathlib import Path
from sofirpy import export_dymola_model

dir_path = Path(__file__).parent
model_path = dir_path.parent / "DC_Motor.mo"
output_direcotry = dir_path
dymola_exe_path = r"C:\Program Files\Dymola 2018 FD01\bin64\Dymola.exe"
model_name = "DC_Motor"
# To import paramater define the parameters and their values in a dictionary as
# follows:

parameters = {"damper.d": 0.1, "damper.useHeatPort": False}

# Note: The values can also be strings, but then they must correspond to the
```

(continues on next page)

(continued from previous page)

```
# Modelica syntax:
# >>> parameters = {"damper.d": "0.1", "damper.useHeatPort": "false"}

export_dymola_model(
    dymola_exe_path, model_path, model_name, output_direcotry, parameters=parameters,␣
↪keep_mos=False
)
```

## 3.2 Simulating a FMU and a Controller

```
from pathlib import Path
from sofirpy import simulate
from discrete_pid import PID # custom implemented pid controller

fmu_path = Path(__file__).parent / "DC_Motor.fmu"

fmu_info = [
    {
        "name": "DC_Motor",
        "path": str(fmu_path),
        "connections": [
            {
                "parameter name": "u",
                "connect to system": "pid",
                "connect to external parameter": "u",
            }
        ],
    }
]


control_infos = [
    {
        "name": "pid",
        "connections": [
            {
                "parameter name": "speed",
                "connect to system": "DC_Motor",
                "connect to external parameter": "y",
            }
        ],
    }
]
pid = PID(1e-3, 3, 20, 0.1, set_point=100, u_max=100, u_min=0)
control_class = {"pid": pid}
parameters_to_log = {"DC_Motor": ["y", "MotorTorque.tau"], "pid": ["u"]}

results, units = simulate(
    stop_time=10,
```

(continues on next page)

(continued from previous page)

```
    step_size=1e-3,
    fmu_infos=fmu_info,
    model_infos=control_infos,
    model_classes=control_class,
    parameters_to_log=parameters_to_log,
    get_units=True,
)
```

The custom implemented pid controller is shown below.

```
from sofirpy import SimulationEntity


class PID(SimulationEntity):
    """Simple implementation of a discrete pid controller"""

    def __init__(
        self, step_size, K_p=1, K_i=0, K_d=0, set_point=0, u_max=1000, u_min=-1000
    ):

        self.T_a = step_size
        self.K_p = K_p
        self.K_i = K_i
        self.K_d = K_d
        self.set_point = set_point
        self.inputs = {"speed": 0}
        self.outputs = {"u": 0}
        self.d_0 = K_p * (
            1 + (self.T_a * self.K_i / self.K_p) + self.K_d / (self.K_p * self.T_a)
        )
        self.d_1 = K_p * (-1 - 2 * self.K_d / (self.K_p * self.T_a))
        self.d_2 = K_p * self.K_d / (self.K_p * self.T_a)
        self.error = [0, 0, 0]
        self.u_max = u_max
        self.u_min = u_min

    def compute_error(self):

        self.error[2] = self.error[1]
        self.error[1] = self.error[0]
        self.error[0] = self.set_point - self.inputs["speed"]

    def set_input(self, input_name, input_value):  # mandatory methode

        self.inputs[input_name] = input_value

    def do_step(self, time):  # mandatory methode

        self.compute_error()
        u = (
            self.outputs["u"]
            + self.d_0 * self.error[0]
```

(continues on next page)

(continued from previous page)

```
            + self.d_1 * self.error[1]
            + self.d_2 * self.error[2]
        )
        if u > self.u_max or u < self.u_min:
            if u > self.u_max:
                u = self.u_max
            if u < self.u_min:
                u = self.u_min

        self.outputs["u"] = u

    def get_parameter_value(self, output_name):  # mandatory methode
        return self.outputs[output_name]
```

## 3.3 Initializing a project and storing data

```
from pathlib import Path
from sofirpy import Project

# Initialize a project by giving it the path to a project directory and the
# path to a hdf5. If the given paths do not exist, they will be created.

project_dir = Path(__file__).parent
hdf5_name = "example_project"
hdf5_path = Path(project_dir) / f"{hdf5_name}.hdf5"
project = Project(hdf5_path, project_dir)

# actions can be taken only on the hdf5, only on the project directory and
# simultaneously on both the hdf5 and the project directory.

# storing data in the hdf5
data = [
    [15, 8, 1, 24, 17],
    [16, 14, 7, 5, 23],
    [22, 20, 13, 6, 4],
    [3, 21, 19, 12, 10],
    [9, 2, 25, 18, 11],
]
data_name = "magic"
data_attributes = {"created on": "01.01.01"}

# specify in which folder the matrix should be stored, if the data should be
# stored at the top level, folder_name must be set to None.

folder_name = "magic_matrices"
project.hdf5.store_data(
    data_name, data, group_path=folder_name, attributes=data_attributes
)
```

(continues on next page)

(continued from previous page)

```
# reading data
data, attr = project.hdf5.read_data(data_name, folder_name, get_attributes=True)

# create a folder in the project directory
folder_name = "magic_matrices"
project.project_dir.create_folder(folder_name)

# create a folder in the project directory and the hdf5 simultaneously
folder_name = "magic_matrices/examples"
project.create_folder(folder_name)

# delete a folder in the project directory and the hdf5 simultaneously
project.delete_folder(folder_name)
```

## 3.4 Additional examples

Additional examples can be found here.

CHAPTER

# FOUR

# CONNECT SYSTEMS

The following example demonstrates how multiple systems can be connected for the simulation.

Consider the following systems.

For each input of a system it must be defined with which output of another system the input is to be connected. For the shown system the variables 'fmu_infos' and 'model_infos' must be defined as follows. The variables 'fmu_infos' and 'model_infos' are inputs for the *simulate()* function.

```
>>> fmu_infos = [
... {"name": "FMU1",
...  "path": "<path to the fmu>",
...  "connections":
...     [
...     {
...         "parameter name": "fmu1_input1",
...         "connect to system": "CustomSystem1",
...         "connect to external parameter": "custom_system1_output1"
...         }
...     ]
...     },
... {"name": "FMU2",
...  "path": "<path to the fmu>",
...  "connections":
...     [
...     {
...         "parameter name": "fmu2_input1",
...         "connect to system": "FMU1",
...         "connect to external parameter": "fmu1_output2"
...        }
... }
... ]
>>> model_infos = [
... {"name": "CustomSystem1",
...  "connections":
...     [
...     {
...         "parameter name": "custom_system1_input1",
...         "connect to system": "FMU1",
...         "connect to external parameter": "fmu1_output1"
...         },
```

(continues on next page)

(continued from previous page)

```
...     {
...         "parameter name": "custom_system1_input2",
...         "connect to system": "CustomSystem2",
...         "connect to external parameter": "custom_system2_output1"
...         }
...     ]
...     },
... {"name": "CustomSystem2"} # CustomSystem2 has no input so not connections need to be␣
↪defined
... ]
```

CHAPTER

FIVE

# LIMITATIONS

## 5.1 Model export

- OpenModelica model export does not allow a parameter import

## 5.2 Simulation

- rather slow simulation
- no variable step size for the simulation
- start values can not be set (coming in the future)

## 5.3 Project

- basic functionalities for storing and reading data → RDM principles have to be implemented (more functionalities coming in the future)

CHAPTER

SIX

# RELEASE NOTES

## 6.1 v0.1.0

The first release of SOFIRpy. SOFIRpy provides functionalities to:

- export Dymola and OpenModelica models as FMUs
- co-simulate FMUs with custom python models
- store data and meta data of the simulation

CHAPTER

SEVEN

# PACKAGES DOCUMENTATION

## 7.1 Subpackages

### 7.1.1 sofirpy.fmu_export package

#### Submodules

#### sofirpy.fmu_export.fmu_export module

This module contains the base class for a fmu export.

**class** sofirpy.fmu_export.fmu_export.**FmuExport**(*model_path: Path*, *fmu_path: Path*)

Bases: object

Object that sets the paths for the fmu export.

**property fmu_path: Path**

Path the exported fmu is going to have.

**Returns**
Path the exported fmu is going to have.

**Return type**
Path

**property model_path: Path**

Path to the modelica model.

**Returns**
Path to the modelica model.

**Return type**
Path

**move_fmu**(*target_directory: Path*) → None

Move the log fmu to a target directory.

**Parameters**
**target_directory** (*Path*) – Path to the target directory.

### sofirpy.fmu_export.dymola_fmu_export module

This module allows to export a Dymola model as a fmu.

**class** sofirpy.fmu_export.dymola_fmu_export.**DymolaFmuExport**(*dymola_exe_path: Path*, *model_path: Path*, *model_name: str*, *parameters: dict[str, Union[str, int, float, list, bool]] | None = None*, *model_modifiers: list[str] | None = None*, *packages: list[Union[str, pathlib.Path]] | None = None*)

Bases: *FmuExport*

Object that performs the Dymola fmu export.

**create_mos_file**(*mos_script: str*) → None

Create the mos file with the specified content.

**Parameters**
**mos_script** (*str*) – The content for the mos file.

**property dymola_exe_path: Path**

Path to the dymola executable.

**Returns**
Path to the dymola executable

**Return type**
Path

**export_fmu**(*export_simulator_log: bool | None = True*, *export_error_log: bool | None = True*) → None

Execute commands to export a fmu.

**Parameters**

- **export_simulator_log** (*bool, optional*) – If True a simulator log file will be generated. Defaults to True.
- **export_error_log** (*bool, optional*) – If True a error log file will be generated. Defaults to True.

**format_parameters**() → list[str]

Format parameter values.

Format parameter values to adjust to dymola scripting syntax and stores the parameter names and values in a list in the following format.

```
>>> parameter_declaration = ['"Resistor.R" =  "1"',
...                          '"Resistor.useHeatPort" = "true"']
```

**Returns**
List of parameters.

**Return type**
list[str]

**property model_modifiers: list[str]**

List of model modifiers.

**Returns**
List of model modifiers.

**Return type**
list[str]

**move_log_file**(*target_directory: Path*) → None

Move the log file to a target directory.

**Parameters**
**target_directory** (*Path*) – Path to the target directory.

**move_mos_script**(*target_directory: Path*) → None

Move the mos script to a target directory.

**Parameters**
**target_directory** (*Path*) – Path to the target directory.

**property parameters: dict[str, Union[str, int, float, list, bool]]**

Dictionary of parameter names and values.

**Returns**
Dictionary of parameter names and values

**Return type**
dict[str, Union[str, int, float, list, bool]]

**write_mos_script**(*export_simulator_log: bool | None = True*, *export_error_log: bool | None = True*) → str

Write the content for the mos file/script.

The script contains the necessary instructions to import the specified parameters and model modifiers and export the model as a fmu.

**Parameters**

- **export_simulator_log** (*bool, optional*) – If True a simulator log file will be generated. Defaults to True.
- **export_error_log** (*bool, optional*) – If True a error log file will be generated. Defaults to True.

**Returns**
content for the mos script

**Return type**
str

sofirpy.fmu_export.dymola_fmu_export.**check_not_valid_parameters**(*imported_parameters: list[str]*, *parameters_in_model: list[str]*) → list[str]

Return parameters that were tried to be imported but were not part of the model.

**Parameters**

- **imported_parameters** (*list[str]*) – Parameter names that were imported.
- **parameters_in_model** (*list[str]*) – Parameters names in the model.

**Returns**
List of parameters names that are were tried to be imported but were not part of the model

**Return type**
list[str]

sofirpy.fmu_export.dymola_fmu_export.**export_dymola_model**(*dymola_exe_path: Path | str*, *model_path: Path | str*, *model_name: str*, *output_directory: Path | str*, *parameters: dict[str, Union[str, int, float, list, bool]] | None = None*, *model_modifiers: list[str] | None = None*, *packages: list[Union[str, pathlib.Path]] | None = None*, *keep_log: bool | None = True*, *keep_mos: bool | None = True*) → *DymolaFmuExport*

Export a dymola model as a fmu.

The following steps are performed: 1. Initializes the DymolaFmuExport object. 2. Tries to export a dymola model as an fmu. 3. When exporting, multiple unnecessary files will be generated. These files will be deleted. 4. If the export was successful, all generated files that are to be kept are moved to the specified output directory. 5. If the export was not successful the model is exported without imported parameters. 6. If the export without imported parameters was successful a list of parameters is genereted that were tried to be imported but are not part of the model.

**Parameters**

- **dymola_exe_path** (*Union[Path, str]*) – Path to the dymola executable.
- **model_path** (*Union[Path, str]*) – Path to the dymola model that should be exported.
- **model_name** (*str*) – Name of the model that should be exported. If the model that should be exported is inside a package, separate the package name and the model name with a '.'.
- **output_directory** (*Union[Path, str]*) – Path to the output directory.
- **parameters** (*dict[str, Union[str, int, float, list, bool]], optional*) – Dictionary of parameter names and values. Example:

```
>>> paremeters = {"Resistor.R" : "1", "Resistor.useHeatPort": True}
```

  Defaults to None.

- **model_modifiers** (*list[str]], optional*) – List of model modifiers. Example

```
>>> model_modifiers = ["redeclare package Medium ="
...     "Modelica.Media.Water.ConstantPropertyLiquidWater"]
```

  Defaults to None.

- **packages** (*Optional[list[Union[str, Path]]], optional*) – List of model/package paths that need to be loaded as dependencies for the model.
- **keep_log** (*Optional[bool], optional*) – If True the simulator log is kept else it will be deleted. Defaults to True.
- **keep_mos** (*Optional[bool], optional*) – If True the mos script is kept else it will be deleted. Defaults to True.

**Returns**

DymolaFmuExport object.

**Return type**

*DymolaFmuExport*

sofirpy.fmu_export.dymola_fmu_export.**read_model_parameters**(*fmu_path: Path*) → list[str]

Read the models parameters of the given fmu.

**Parameters**
**fmu_path** (*Path*) – Path to a fmu.

**Returns**
List of parameters is the model.

**Return type**
list[str]

#### sofirpy.fmu_export.open_modelica_fmu_export module

This module allows to export a OpenModelica model as a fmu.

**class** sofirpy.fmu_export.open_modelica_fmu_export.**OpenModelicaFmuExport**(*model_path: Path*, *model_name: str*)

Bases: *FmuExport*

Object that performs the OpenModelica fmu export

**export_fmu**()

Exports the model as an fmu.

sofirpy.fmu_export.open_modelica_fmu_export.**export_open_modelica_model**(*model_path: Path | str*, *model_name: str*, *output_directory: Path | str*) → *OpenModelicaFmuExport* | None

Exports a modelica model as an fmu and moves the fmu to the output directory

**Parameters**

- **model_path** (*Union[Path, str]*) – Path to the modelica model that should be exported
- **model_name** (*str*) – Name of the model.
- **output_directory** (*Union[Path, str]*) – Path to the output directory.

**Returns**
OpenModelicaFmuExport object

**Return type**
*OpenModelicaFmuExport*

### 7.1.2 sofirpy.simulation package

#### UML Diagram of the simulation package

## Submodules

### sofirpy.simulation.simulation module

This module allows to co-simulate multiple fmus and models written in python.

**class** sofirpy.simulation.simulation.**Connection**(*input_point: ConnectionPoint*, *output_point: ConnectionPoint*)

Bases: `object`

Object representing a connection between two systems.

**Parameters**

- **input_point** (ConnectionPoint) – ConnectionPoint object that represents an input of a system
- **output_point** (ConnectionPoint) – ConnectionPoint object that represents an output of a system

**class** sofirpy.simulation.simulation.**ConnectionPoint**(*system: System*, *name: str*)

Bases: *SystemParameter*

ConnectionPoint object representing a parameter in a system that is an input our output.

**class** sofirpy.simulation.simulation.**LoggedParameter**(*system: System*, *name: str*)

Bases: *SystemParameter*

LoggedParameter object representing a parameter in a system that is logged.

**class** sofirpy.simulation.simulation.**Simulation**(*systems: list[sofirpy.simulation.simulation.System]*, *connections: list[sofirpy.simulation.simulation.Connection]*, *parameters_to_log: list[sofirpy.simulation.simulation.LoggedParameter]* | *None = None*)

Bases: `object`

Object that performs the simulation.

**create_result_df**(*parameters_to_log: list[sofirpy.simulation.simulation.LoggedParameter]*) → DataFrame

Initialise the result dataframe. By default the first column contains the time.

**Parameters**

**parameters_to_log** (*list[LoggedParameter]*) – list of parameters that should be logged

**Returns**

dataframe with only the column names

**Return type**

pd.DataFrame

**do_step**(*time: float*) → None

Perform a calculation in all systems.

**Parameters**

- **time** (*float*) – current simulation time

- **step_size** (*float*) – step size of the simulation

**get_units**() → dict[str, str]

Get a dictionary with all logged parameters as keys and their units as values.

**Returns**
keys: parameter name, values: unit. If the unit can not be obtained it is set to None.

**Return type**
dict[str, str]

**log_values**(*time: float*, *time_step: int*) → None

Log parameter values that are set o be logged.

**Parameters**

- **time** (*float*) – current simulation time
- **time_step** (*int*) – current time step

**set_systems_inputs**() → None

Set inputs for all systems.

**simulate**(*stop_time: float*, *step_size: float*, *start_time: float = 0.0*) → DataFrame

Simulate the systems.

**Parameters**

- **stop_time** (*float*) – stop time for the simulation
- **step_size** (*float*) – step size for the simulation
- **start_time** (*float, optional*) – start time of the simulation. Defaults to 0.0.

**Returns**
results dataframe with times series of logged parameters

**Return type**
pd.DataFrame

**class** sofirpy.simulation.simulation.**System**(*simulation_entity:* SimulationEntity, *name: str*)

Bases: object

System object representing a simulation entity.

**Parameters**

- **simulation_entity** (SimulationEntity) – fmu or python model
- **name** (*str*) – name of the system

**class** sofirpy.simulation.simulation.**SystemParameter**(*system:* System, *name: str*)

Bases: object

SystemParameter object representing a parameter in a system.

**Parameters**

- **system** (System) – System object
- **name** (*str*) – name of the paremeter

sofirpy.simulation.simulation.**init_connections**(*system_infos: list[dict[str, Union[str, list[dict[str, str]]]]]*, *systems: dict[str, sofirpy.simulation.simulation.System]*) → list[*sofirpy.simulation.simulation.Connection*]

Initialize all the connections.

**Parameters**

- **system_infos** (*list[dict[str, Union[str, list[dict[str, str]]]]]*) – Defines how all systems are connected.
- **systems** (*dict[str, System]*) – Dictrionary with system names as keys and the corresponding System instance as values.

**Returns**
List of Connections.

**Return type**
list[*Connection*]

sofirpy.simulation.simulation.**init_parameter_list**(*parameters_to_log: dict[str, list[str]] | None*, *systems: dict[str, sofirpy.simulation.simulation.System]*) → list[*sofirpy.simulation.simulation.LoggedParameter*]

Initialize all parameters that should be logged.

**Parameters**

- **parameters_to_log** (*Optional[dict[str, list[str]]]*) – Defines which paramters should be logged.
- **systems** (*dict[str, System]*) – Dictrionary with system names as keys and the corresponding System instance as values.

**Returns**
List of system parameters that should be logged.

**Return type**
list[*LoggedParameter*]

sofirpy.simulation.simulation.**init_systems**(*fmu_infos: list[dict[str, Union[str, list[dict[str, str]]]]]*, *model_infos: list[dict[str, Union[str, list[dict[str, str]]]]]*, *model_classes: dict[str, sofirpy.simulation.simulation_entity.SimulationEntity]*, *step_size: float*) → dict[str, *sofirpy.simulation.simulation.System*]

Initialize all System object and stores them in a dictionary.

**Parameters**

- **fmu_infos** (*list[dict[str, Union[str, list[dict[str, str]]]]]*) – Defines which fmus should be simulated and how they are connected to other systems.
- **model_infos** (*list[dict[str, Union[str, list[dict[str, str]]]]]*) – Defines which python models should be simulated and how they are connected to other systems.
- **model_classes** (*dict[str, SimulationEntity]*) – Dictionary with the name of the models as keys and a instance of the model class as values.
- **step_size** (*float*) – step size of the simulation

**Returns**

**Dictrionary with system names as keys and the**
corresponding System instance as values.

**Return type**
dict[str, *System*]

sofirpy.simulation.simulation.**simulate**(*stop_time: float | int*, *step_size: float*, *fmu_infos: list[dict[str, Union[str, list[dict[str, str]]]]] | None = None*, *model_infos: list[dict[str, Union[str, list[dict[str, str]]]]] | None = None*, *model_classes: dict[str, sofirpy.simulation.simulation_entity.SimulationEntity] | None = None*, *parameters_to_log: dict[str, list[str]] | None = None*, *get_units: bool | None = False*) → DataFrame | tuple[pandas.core.frame.DataFrame, dict[str, str]]

Simulate fmus and models written in python.

Any number of python models and fmus can be simulated, but at least one python model or fmu has to be simulated.

**Parameters**

- **stop_time** (*Union[float,int]*) – stop time for the simulation
- **step_size** (*float*) – step size for the simulation
- **fmu_infos** (*Optional[list[dict[str, Union[str, list[dict[str, str]]]]], optional*) – Defines which fmus should be simulated and how they are connected to other systems. It needs to have the following formart:

```
>>> fmu_infos = [
... {"name": "<name of the fmu>",
...  "path": "<path to the fmu>",
...  "connections":
...     [
...     {
...         "parameter name":       "<name of the input"
...                                 "parameter of the fmu>",
...         "connect to system":    "<name of the system the input"
...                                 "parameter should be connected␣
↪to>",
...         "connect to external parameter":    "<name of the output"
...                                             "parameter in the"
...                                             "connected system the
↪"
...                                             "input parameter␣
↪should"
...                                             "be connected to>"
...         },
...         {
...         "parameter name":       "<name of the input"
...                                 "parameter of the fmu>",
...         "connect to system":    "<name of the system the input"
...                                  "parameter should be connected␣
↪to>",
...         "connect to external parameter":    "<name of the output"
...                                             "parameter in the"
```

(continues on next page)

(continued from previous page)

```
...                                                 "connected system the
↪"
...                                                 "input parameter␣
↪should"
...                                                 "be connected to>"
...         }
...     ]
...     },
... {"name": "<name of the fmu>",
...  "path": "<path to the fmu>",
...  "connections":
...     [
...     {
...         "parameter name":       "<name of the input"
...                                 "parameter of the fmu>",
...         "connect to system":    "<name of the system the input"
...                                  "parameter should be connected␣
↪to>",
...         "connect to external parameter":    "<name of the output"
...                                             "parameter in the"
...                                             "connected system the
↪"
...                                             "input parameter␣
↪should"
...                                             "be connected to>"
...         }
... }
... ]
```

Note: The name of the fmus can be chosen arbitrarily, but each name in 'fmu_infos' and 'model_infos' must occur only once. Defaults to None.

- **model_infos** (*Optional[list[dict[str, Union[str, list[dict[str, str]]]]]], optional*) – Defines which python models should be simulated and how they are connected to other systems. It needs to have the same format as 'fmu_infos' with the difference that the key "path" is not part of the dictionaries. Note: The name of the models can be chosen arbitrarily, but each name in 'fmu_infos' and 'model_infos' must occur only once. Defaults to None.
- **model_classes** (*Optional[dict[str, SimulationEntity]], optional*) – Dictionary with the name of the python model as keys and a instance of the model class as values. The name in the dictionary must be chosen according to the names specified in 'model_infos'. Defaults to None.
- **parameters_to_log** (*Optional[dict[str, list[str]]], optional*) – Dictionary that defines which parameters should be logged. It needs to have the following format:

```
>>> parameters_to_log =
... {
...     "<name of system 1 (corresponding to the names specified in"
...     "'model_infos' or 'fmu_infos')>":
...     [
...         "<name of parameter 1>",
```

(continues on next page)

(continued from previous page)

```
...         "<name of parameter 2>",
...     ],
...     "<name of system 2>":
...     [
...         "<name of parameter 1>",
...         "<name of parameter 2>",
...     ]
... }
```

Defaults to None.

- **get_units** (*Optional[bool], optional*) – Determines whether the units of the logged parameter should be returned. Defaults to False.

**Raises**

- **TypeError** – start_time type was invalid
- **TypeError** – step_size type was invalid
- **TypeError** – fmu_infos type was invalid
- **TypeError** – model_infos type was invalid
- **ValueError** – fmu_infos and model_infos were 'None'
- **TypeError** – model_classes type was invalid
- **ValueError** – start_time value was invalid
- **ValueError** – step_size value was invalid

**Returns**

Result Dataframe with times series of logged parameters, units of logged parameters.

**Return type**

Union[pd.DataFrame, tuple[pd.DataFrame, dict[str,str]]]

### sofirpy.simulation.fmu module

**class** sofirpy.simulation.fmu.**Fmu**(*fmu_path: Path*, *step_size: float*)

Bases: *SimulationEntity*

Class representing a fmu.

**conclude_simulation_process**() → None

Conclude the simulation process of the fmu.

**create_model_vars_dict**() → None

Create a dictionary for the variables of the fmu.

The keys of this dictionary are the names of the variables and the values are the coresponding reference numbers.

**create_unit_vars_dict**() → None

Create a dictionary for the units of the fmu variables.

The keys of this dictionary are the names of the variables and the values are the coresponding units.

**do_step**(*time: float*) → None

Perform a simulation step.

**Parameters**
**time** (*float*) – current time

**property fmu_path: Path**

Path to the fmu.

**Returns**
Path to the fmu.

**Return type**
Path

**get_parameter_value**(*parameter_name: str*) → float

Return the value of a parameter.

**Parameters**
**parameter_name** (*str*) – name of parameter whose value is to be obtained

**Returns**
value of the parameter

**Return type**
Union[int, float]

**get_unit**(*parameter_name: str*) → str

Return the unit of a variable.

**Parameters**
**parameter_name** (*str*) – Name of the variable.

**Returns**
The unit of the variable.

**Return type**
str

**initialize_fmu**(*start_time: float | None = 0*) → None

Initialize the fmu.

**Parameters**
**start_time** (*float, optional*) – start time. Defaults to 0.

**set_input**(*input_name: str*, *input_value: float | int*) → None

Set the value of an input parameter.

**Parameters**

- **input_name** (*str*) – name of the parameter that should be set
- **input_value** (*Union[float, int]*) – value to which the parameter is to be set

### sofirpy.simulation.simulation_entity module

**class** sofirpy.simulation.simulation_entity.**SimulationEntity**

Bases: ABC

Abstract object representing a simulation entity.

**abstract do_step**(*time: float*)

Perform simulation entity calculation and set parameters accordingly.

**Parameters**
**time** (*float*) – current simulation time

**abstract get_parameter_value**(*parameter_name: str*) → int | float

Return the value of a parameter.

**Parameters**
**parameter_name** (*str*) – name of parameter whose value is to be obtained

**Returns**
value of the parameter

**Return type**
Union[int, float]

**abstract set_input**(*input_name: str*, *input_value: float | int*)

Set the value of an input parameter.

**Parameters**

- **input_name** (*str*) – name of the parameter that should be set
- **input_value** (*Union[float, int]*) – value to which the parameter is to be set

### sofirpy.simulation.plot module

This module provides a easy function for plotting the simulation results.

sofirpy.simulation.plot.**plot_results**(*results: DataFrame*, *x_name: str*, *y_name: str | list[str]*, *x_label: str | None = None*, *y_label: str | None = None*, *title: str | None = None*, *legend: str | list | None = None*, *style_sheet_path: Path | str | None = None*) → Axes

Plot the simulation results.

**Parameters**

- **results** (*pd.DataFrame*) – Simulation results.
- **x_name** (*str*) – Name of data that should be on the x-axis.
- **y_name** (*Union[str, list[str]]*) – Name of data that should be on the y-axis. For multiple plots, give a list with names as the argument.
- **x_label** (*str, optional*) – X-label for the plot. Defaults to None.
- **y_label** (*str, optional*) – Y-label for the plot. Defaults to None.
- **title** (*str, optional*) – Title for the plot. Defaults to None.
- **legend** (*Union[str, list], optional*) – Legend for the plot. For multiple plots give a list of strings as the argument. Defaults to None.

- **style_sheet_path** (*Union[str, Path], optional*) – Path to a matplotlib style sheet. Defaults to None.

**Returns**
Matplotlib Axes object.

**Return type**
Axes

### 7.1.3 sofirpy.project package

#### Submodules

#### sofirpy.project.hdf5 module

This module allows to easily store and read data from a hdf5 file.

**class** sofirpy.project.hdf5.**HDF5**(*hdf5_path: Path | str*)

Bases: object

Object representing a HDF5.

**append_attributes**(*path: str*, *attributes: dict[str, Any]*) → None

Append attributes to a hdf5 Dataset or a Group.

**Parameters**

- **path** (*str*) – hdf5 path to the dataset or group.
- **attributes** (*dict[str, Any]*) – Attributes dictionary with attribute names as keys and the attributes as values

**create_group**(*group_path: str*) → None

Creates a group in the hdf5 file.

**Parameters**
**group_path** (*str*) – The name of the group. Subgroups can be created by seperating the group names with '/'. Example:

```
>>> # create a group at the top level with the name "group1"
>>> group_path = "group1"
>>> # create a group with the name "subgroup1" in "group1"
>>> group_path = "group1/subgroup1"
```

The parant groups does not need to exist to create a subgroup. They will be created automatically. Example:

```
>>> group_path = "group2/subgroup1/subsubgroup1"
```

**delete_attribute**(*path: str*, *attribute_name: str*) → None

Deletes a attribute of a hdf5 Dataset or Group.

**Parameters**

- **path** (*str*) – hdf5 path to the dataset or group.
- **attribute_name** (*str*) – Name of the attribute.

**delete_data**(*group_path: str*, *data_name: str*) → None

Deletes a hdf5 Dataset.

**Parameters**

- **group_path** (*str*) – Path to the hdf5 group the data is in.
- **data_name** (*str*) – Name of the data.

**Raises**

- **KeyError** – If the hdf5 path to the data doesn't exists.
- **ValueError** – If the hdf5 path to the data does not lead to hdf5 Dataset.

**delete_group**(*group_path: str*) → None

Deletes hdf5 group.

**Parameters**
**group_path** (*str*) – Path to the hdf5 group.

**Raises**

- **KeyError** – If the hdf5 path doesn't exists.
- **ValueError** – If the group_path does not lead to hdf5 Group.

**property hdf5_path: Path**

Path to a hdf5 file.

**Returns**
Path to a hdf5 file.

**Return type**
Path

**read_data**(*data_name: str*, *group_path: str*, *get_attributes: bool | None = False*) → Any | tuple[Any, dict[str, Any]]

Reads the data of at a given data path.

**Parameters**

- **data_name** (*str*) – Name of the data.
- **group_path** (*str*) – Path to the hdf5 group.
- **get_attributes** (*Optional[bool], optional*) – If True attributes will be returned as well. Defaults to False.

**Raises**
**ValueError** – If data path does not lead to a hdf5 Dataset.

**Returns**
Data and/or attributes of the Dataset.

**Return type**
Union[Any, tuple[Any, dict[str, Any]]]

**read_entire_group_data**(*group_path: str | None = None*) → dict[str, Any]

Reads all data inside a hdf5 group.

The data will be returned in a nested dictionary representing the file structure. If no path to a group is given all the data in the hdf5 will returned.

**Parameters**
**group_path** (*Optional[str], optional*) – Path to a hdf5 group. Defaults to None.

**Raises**
**ValueError** – If the group_path does not lead to a hdf5 group.

**Returns**
Dictionary with the data of the given group.

**Return type**
dict[str, Any]

**read_hdf5_structure**(*group_path: str | None = None*) → None | dict[str, Any]

Reads the structure of a hdf5 group.

The file structure is represented by a nested dictionary. If group_path is not given the structure of the wholen hdf5 will be returned.

**Parameters**
**group_path** (*Optional[str], optional*) – Path to the hdf5 group which file structure should be returned. Defaults to None.

**Returns**
Returns a dictionary with the structure corresponding to the structure of the hdf5 group.

**Return type**
Union[None, dict[str, Any]]

**store_data**(*data_name: str*, *data: Any*, *group_path: str*, *attributes: dict | None = None*) → None

Stores data in a hdf5 group. If the group doesn't exist it will be created.

**Parameters**

- **data_name** (*str*) – Name of the data.
- **data** (*Any*) – Data that should be stored.
- **group_path** (*str*) – Path to the hdf5 group.
- **attributes** (*Optional[dict], optional*) – Data attributes dictionary with attribute names as keys and the attributes as values. Defaults to None.

**Raises**
**ValueError** – If data path already exists.

### sofirpy.project.project_dir module

This module allows to apply different actions within a directory.

**class** sofirpy.project.project_dir.**ProjectDir**(*project_directory: Path | str*)

Bases: object

Object representing the Project Directory.

**copy_and_rename_file**(*source_path: Path | str*, *target_dir: Path | str*, *new_name: str*) → Path

Copy and rename a file.

**Parameters**

- **source_path** (*Union[Path, str]*) – Source path of the file that should be copied and renamed.
- **target_dir** (*Union[Path, str]*) – Target directory the file should be copied to.

- **new_name** (*str*) – New file name.

**Returns**
Path to the copied file.

**Return type**
Path

**copy_file**(*source_path: Path | str*, *target_directory: Path | str | None = None*) → Path

Copy a file from a source path to a target directory while keeping the file name.

**Parameters**

- **source_path** (*Union[Path, str]*) – Source path of the file that should be copied.
- **target_directory** (*Optional[Union[Path, str]], optional*) – Target directory the file should be moved to. If not specified the file is moved to the current folder. Defaults to None.

**Returns**
Path to the copied file.

**Return type**
Path

**create_folder**(*folder_name: str*) → None

Create a folder in the project directory.

**Parameters**
**folder_name** (*str*) – Name of the folder. Subfolders can be created by seperating the folder names with '/'.

**property current_folder: Path**

Path to the current folder.

**Returns**
Path to the current folder.

**Return type**
Path

**delete_element**(*name: str*) → None

Delete file or directory in the current folder.

**Parameters**
**name** (*str*) – Name of the file/directory that should be deleted.

**delete_files**(*file_names: list[str]*) → None

Delete multiple files in the current folder.

**Parameters**
**file_names** (*list[str]*) – List with file names.

**delete_folder**(*folder_name: str*) → None

Delete a folder in the project directory.

**Parameters**
**folder_name** (*str*) – Name of the folder that should be deleted. Subfolders can be deleted by seperating the folder names with '/'.

**move_and_rename_file**(*source_path: Path | str*, *target_dir: Path | str*, *new_name: str*) → Path

Move and rename a file.s

**Parameters**

- **source_path** (*Union[Path, str]*) – Source path of the file that should be moved and renamed.
- **target_dir** (*Union[Path, str]*) – Target directory the file should be moved to.
- **new_name** (*str*) – New file name.

**Returns**
Path to the moved file.

**Return type**
Path

**move_file**(*source_path: Path | str*, *target_directory: Path | str | None = None*) → Path

Move a file from a source path to a target directory.

**Parameters**

- **source_path** (*Union[Path, str]*) – Source path of the file that should be moved.
- **target_directory** (*Union[Path, str], optional*) – Target Directory the file should be moved to. If not specified the file is moved to the current folder. Defaults to None.

**Returns**
Path to the moved file.

**Return type**
Path

**move_files**(*source_paths: list[Union[pathlib.Path, str]]*, *target_directory: Path | str | None = None*) → None

Move multiple files to a target directory.

**Parameters**

- **source_paths** (*list[Union[Path, str]]*) – List of file paths that should be moved.
- **target_directory** (*Optional[Union[Path, str]], optional*) – Target Directory the file should be moved to. If not specified the file is moved to the current folder. Defaults to None.

**property project_directory: Path**

Path to the project directory.

**Returns**
Path to the project directory.

**Return type**
Path

**rename_file**(*file_path: Path | str*, *new_name: str*) → Path

Rename a file.

**Parameters**

- **file_path** (*Union[Path, str]*) – Path to the filed that should be renamed.

- **new_name** (*str*) – New file name without the suffix. The suffix of the file will stay the same.

**Returns**
Path to the renamed file.

**Return type**
Path

**set_current_folder**(*folder_name: str*) → None

Set the current folder.

**Parameters**
**folder_name** (*str*) – Name of the folder.

### sofirpy.project.project module

This module allows to take actions on a given hdf5 file and directory simultaneously.

**class** sofirpy.project.project.**Project**(*hdf5_path: Path | str*, *project_dir_path: Path | str*)

Bases: object

Object representing a project.

**create_folder**(*folder_name: str*) → None

Create a folder in the hdf5 and in the project directory.

**Parameters**
**folder_name** (*str*) – Name of the folder. Subfolders can be created by seperating the folder names with '/'.

**Raises**
**error** – If error occurs while creating the folder in the project directory.

**delete_folder**(*folder_name: str*) → None

Delete a folder in the hdf5 and in the project directory.

**Parameters**
**folder_name** (*str*) – Name of the folder. Subfolders can be deleted by seperating the folder names with '/'.

**store_file**(*source_path: str | Path*, *folder_name: str*, *copy: bool | None = True*, *new_file_name: str | None = None*) → None

Store a file in the project directory and a reference this file in the hdf5.

**Parameters**

- **source_path** (*Union[str, Path]*) – Path to the file.
- **folder_name** (*str*) – Name of the folder the file should be stored in.
- **copy** (*Optional[bool], optional*) – If true the file will be copied from it's source path else it will be moved. Defaults to True.
- **new_file_name** (*Optional[str], optional*) – If specified the file will be renamed accordingly. Defaults to None.

### sofirpy.project.store_input_arguments module

sofirpy.project.store_input_arguments.**store_input_arguments**(*func: Callable*)

Decorator that lets you store the input arguments of the __init__ class methode.

The input arguments will be stored as a dictrionay with the variable names as keys and the value of the variables as values. The dictrionay will be stored as a class attribute with the name '__input_arguments__'.

**Parameters**
**func** (*Callable*) – Function to decorate

**Returns**
Decorated function

**Return type**
Callable

CHAPTER

# EIGHT

# CONTRIBUTIONS

**Jonas Boder**: Conceptualization, Methodology **Julius Breuer**: Validation **Lara Doß**: Conceptualization, Methodology **Hendrick Gockel**: Conceptualization, Methodology **Daniele Inturri**: Conceptualization, Methodology, Software, Validation **Michaela Lestáková**: Supervision **Kevin Logan**: Supervision **Tim Müller**: Supervision **Tim Schimkat**: Conceptualization, Methodology **Johanna Spegg**: Conceptualization, Methodology

CHAPTER

NINE

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## S

# INDEX

## H



## I



## L



## M



## O



## P



## R



## S

## W