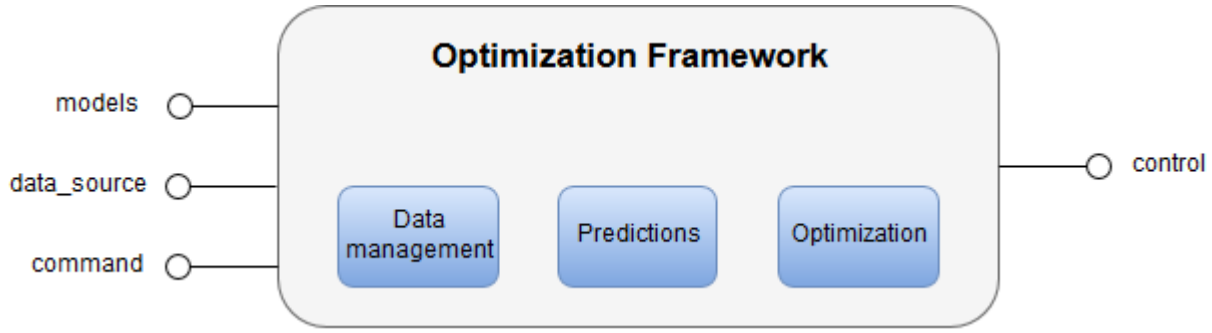
Optimization Framework

LinkSmart® Optimization Framework (OFW) is a frameworkthat enables continous optimal control in diverse application domains The architecture includes modules for management and signal processing of sensor data , linking of predictive algorithms to deliver inputs to the optimization model, optimization modeling, linking of a solver, an optimization controller and a post-processer module for formatting the results or creating control events. The framework offers an API that describes the insertion of new optimization models, allows the registration of data sources and control outputs and presents a set of commands to control the start of the framework. All implementations support SenML data model and may offer additional formats using HTTP negotiation.



A full-implementation of OFW offers four APIs:

- **data_source**: Registry of input data sources (MQTT or HTTP).
- **control:** Registry of outputs of the optimization problem.
- **command:** Commands for starting and stopping the framework, while entering optimization parameters.
- **models:** Registry of a new optimization model, which is used for the optimal control and is based in Pyomo

Getting Started

Examples

Working with Optimization Models (/models)

Inserting an optimization model

The first step is to have an optimization model written as documented in pyomo for python. You can use the following link to beautify your code or to find some errors in the nomenclature and indentation.

You can find here an example of how an optimization model should look like.

After having your optimization model, you can introduce it into the framework, by using the following command:

```
curl --request POST \
 --url http://ip_address:8080/v1/models/upload/custom_model \
 --header 'Cache-Control: no-cache' \
 --header 'Content-Type: text/plain' \
 --header 'Postman-Token: 747de622-89a9-428a-b6d0-b75c574bed77' \
 --data 'model = AbstractModel()\r\n \r\nmodel.T = Set() # Index Set for time steps of optimization
horizon\r\nmodel.T_SoC = Set() # SoC of the ESSs at the end of optimization horizon are also taken into
account\r\n \r\n################################## PARAMETERS ##################################\r\n \r\nmodel.
dT = Param(within=PositiveIntegers) # Number of seconds in one time step\r\n \r\nmodel.P_PV = Param(model.T,
within=NonNegativeReals) # PV PMPP forecast\r\n \r\nmodel.ESS_Min_SoC = Param(within=PositiveReals) # Minimum
SoC of ESSs\r\nmodel.ESS_Max_SoC = Param(within=PositiveReals) # Maximum SoC of ESSs\r\nmodel.SoC_Value = Param
(within=PositiveReals)\r\nmodel.ESS_Capacity = Param(within=PositiveReals) # Storage Capacity of ESSs\r\nmodel.
ESS_Max_Charge_Power = Param(within=PositiveReals) # Max Charge Power of ESSs\r\nmodel.ESS_Max_Discharge_Power
= Param(within=PositiveReals) # Max Discharge Power of ESSs\r\n \r\nmodel.P_Grid_Max_Export_Power = Param
(within=NonNegativeReals) # Max active power export\r\nmodel.Q_Grid_Max_Export_Power = Param
(within=NonNegativeReals) # Max reactive power export\r\n \r\nmodel.PV_Inv_Max_Power = Param
(within=PositiveReals) # PV inverter capacity\r\n \r\n################################## VARIABLES
##################################\r\n \r\nmodel.P_Grid_Output = Var(model.T, within=Reals)\r\nmodel.
Q_Grid_Output = Var(model.T, within=Reals)\r\nmodel.P_PV_Output = Var(model.T, within=NonNegativeReals, bounds=
(0, model.PV_Inv_Max_Power))\r\nmodel.P_ESS_Output = Var(model.T, within=Reals, bounds=(-model.
ESS_Max_Charge_Power, model.ESS_Max_Discharge_Power))\r\nmodel.SoC_ESS = Var(model.T_SoC,
within=NonNegativeReals, bounds=(model.ESS_Min_SoC, model.ESS_Max_SoC))\r\n
\r\n################################ CONSTRAINTS ################################\r\n \r\n# PV
constraints\r\ndef con_rule_pv_potential(model, t):\r\n return model.P_PV_Output[t] <= model.P_PV[t]\r\n \r\n#
Import/Export constraints for Active power\r\ndef con_rule_grid_P_inv(model, t):\r\n return model.P_Grid_Output
[t] >= -model.P_Grid_Max_Export_Power\r\n \r\n# Import/Export constraints for Active power\r\ndef
con_rule_grid_Q_inv(model, t):\r\n return model.Q_Grid_Output[t] >= -model.Q_Grid_Max_Export_Power\r\n \r\n#
ESS SoC balance\r\ndef con_rule_socBalance(model, t):\r\n return model.SoC_ESS[t + 1] == model.SoC_ESS[t] -
model.P_ESS_Output[t] * model.dT / model.ESS_Capacity / 3600\r\n \r\n# Initializing SoC\r\ndef con_rule_iniSoC
(model):\r\n return model.SoC_ESS[0] == model.SoC_Value\r\n \r\n# Generation-feed in balance\r\ndef
con_rule_generation_feedin(model, t):\r\n return model.P_Grid_Output[t] * model.P_Grid_Output[t] + model.
Q_Grid_Output[t] * model.Q_Grid_Output[t] == (\r\n model.P_PV_Output[t] + model.P_ESS_Output[t]) * (model.
P_PV_Output[t] + model.P_ESS_Output[t])\r\n \r\n \r\n# Gnerating constraints\r\nmodel.con_pv_pmax = Constraint
(model.T, rule=con_rule_pv_potential)\r\nmodel.con_grid_inv_P = Constraint(model.T, rule=con_rule_grid_P_inv)
\r\nmodel.con_grid_inv_Q = Constraint(model.T, rule=con_rule_grid_Q_inv)\r\nmodel.con_ess_soc = Constraint
(model.T, rule=con_rule_socBalance)\r\nmodel.con_ess_Inisoc = Constraint(rule=con_rule_iniSoC)\r\nmodel.
con_gen_feedin = Constraint(model.T, rule=con_rule_generation_feedin)\r\n \r\n
\r\n############################## OBJECTIVE ##############################################\r\n \r\ndef obj_rule
(model):\r\n return sum(model.P_PV[t] - model.P_PV_Output[t] for t in model.T)\r\n \r\nmodel.obj = Objective
(rule=obj_rule, sense=minimize)'
```

or through the Postman tool which simplifies the send of HTTP requests. You can install Postman from here.

Obtaining the names of the optimization models saved into the repository

For checking which optimization models are present in the repository, use the following command:

```
curl --request GET --url http://ip_address:8080/v1/models --header 'Cache-Control: no-cache'
```

Designing your optimization model

- OFW is based on abstract models written in Pyomo. Its documentation can give insights and examples of how to do it.
- While designing your optimization model, don't forget that you can use reserved words for defining your parameters and variables. If you do it, then you can them easily register them as inputs or outputs as in this example.
- Don't forget that you have to use an **abstract model**. This platform doesn't support discrete models.
- If there is a problem in your optimization, you can see it in the logs while running OFW.

Defining Parameters

If you want to insert the Photovoltaic power as a parameter of your model, you could follow these two ways:

1. Choosing a reserved word

   model.**P_PV** = Param(model.T, within=NonNegativeReals)

2. Choosing a custom word

   model.**Power_Photovoltaic** = Param(model.T, within=NonNegativeReals)

The difference is how you have to registry these two inputs through the API /data_source.

1. Considering you choose the **first option** and you have input data through MQTT (Go to the example)
2. Considering you choose the **first option** and you have input data through HTTP (Go to the example)
3. Considering you choose the **second option** and you have input data through MQTT (Go to the example)
4. Considering you choose the **second option** and you have input data through HTTP (Go to the example)

Optimization Model example

```
model = AbstractModel()

model.T = Set()  # Index Set for time steps of optimization horizon
model.T_SoC = Set()  # SoC of the ESSs at the end of optimization horizon are also taken into account

################################### PARAMETERS ###################################

model.dT = Param(within=PositiveIntegers)  # Number of seconds in one time step

model.P_PV = Param(model.T, within=NonNegativeReals)  # PV PMPP forecast

model.ESS_Min_SoC = Param(within=PositiveReals)  # Minimum SoC of ESSs
model.ESS_Max_SoC = Param(within=PositiveReals)  # Maximum SoC of ESSs
model.SoC_Value = Param(within=PositiveReals)
model.ESS_Capacity = Param(within=PositiveReals)  # Storage Capacity of ESSs
model.ESS_Max_Charge_Power = Param(within=PositiveReals)  # Max Charge Power of ESSs
model.ESS_Max_Discharge_Power = Param(within=PositiveReals)  # Max Discharge Power of ESSs

model.P_Grid_Max_Export_Power = Param(within=NonNegativeReals)  # Max active power export
model.Q_Grid_Max_Export_Power = Param(within=NonNegativeReals)  # Max reactive power export

model.PV_Inv_Max_Power = Param(within=PositiveReals)  # PV inverter capacity

################################### VARIABLES ###################################

model.P_Grid_Output = Var(model.T, within=Reals)
model.Q_Grid_Output = Var(model.T, within=Reals)
model.P_PV_Output = Var(model.T, within=NonNegativeReals, bounds=(0, model.PV_Inv_Max_Power))
model.P_ESS_Output = Var(model.T, within=Reals, bounds=(-model.ESS_Max_Charge_Power, model.
ESS_Max_Discharge_Power))
model.SoC_ESS = Var(model.T_SoC, within=NonNegativeReals, bounds=(model.ESS_Min_SoC, model.ESS_Max_SoC))

################################### CONSTRAINTS ###################################

# PV constraints
def con_rule_pv_potential(model, t):
    return model.P_PV_Output[t] <= model.P_PV[t]

# Import/Export constraints for Active power
def con_rule_grid_P_inv(model, t):
    return model.P_Grid_Output[t] >= -model.P_Grid_Max_Export_Power

# Import/Export constraints for Active power
def con_rule_grid_Q_inv(model, t):
    return model.Q_Grid_Output[t] >= -model.Q_Grid_Max_Export_Power

# ESS SoC balance
def con_rule_socBalance(model, t):
    return model.SoC_ESS[t + 1] == model.SoC_ESS[t] - model.P_ESS_Output[t] * model.dT / model.ESS_Capacity /
3600

# Initializing SoC
def con_rule_iniSoC(model):
    return model.SoC_ESS[0] == model.SoC_Value

# Generation-feed in balance
def con_rule_generation_feedin(model, t):
    return model.P_Grid_Output[t] * model.P_Grid_Output[t] + model.Q_Grid_Output[t] * model.Q_Grid_Output[t] ==
(
            model.P_PV_Output[t] + model.P_ESS_Output[t]) * (model.P_PV_Output[t] + model.P_ESS_Output[t])


# Gnerating constraints
model.con_pv_pmax = Constraint(model.T, rule=con_rule_pv_potential)
```

```
model.con_grid_inv_P = Constraint(model.T, rule=con_rule_grid_P_inv)
model.con_grid_inv_Q = Constraint(model.T, rule=con_rule_grid_Q_inv)
model.con_ess_soc = Constraint(model.T, rule=con_rule_socBalance)
model.con_ess_Inisoc = Constraint(rule=con_rule_iniSoC)
model.con_gen_feedin = Constraint(model.T, rule=con_rule_generation_feedin)


############################## OBJECTIVE ##########################################

def obj_rule(model):
    return sum(model.P_PV[t] - model.P_PV_Output[t] for t in model.T)

model.obj = Objective(rule=obj_rule, sense=minimize)
```

Defining inputs (/data_source)

You can enter data into the framework through MQTT or through HTTP requests.

In the case that you want to introduce data continously through MQTT, you should read Defining Inputs through MQTT.

In the case that you want to introduce data through HTTP, you should read Defining Inputs through HTTP.

Defining Inputs through HTTP

Using POST /data_source/file

The first step to define an input is to use the POST request of the endpoint /data_source/mqtt.

Registering a data_source through HTTP

Using a reserved word

If you chose a reserved word in your optimization model because you are working with energy applications, you can register your input data in the following way.

Remember that you can send the input data directly. It is entered as a list oj JSON values.

- From the API, you find out which is the data model for entering a photovoltaic registry. Inside of it you choose the one that you need. If you are interested in registering an input for the total PV power for example, you choose P_PV. Because we want to enter meta data at the same time, we choose "meta" as well.

```
{
    "photovoltaic":{
        "P_PV":[
            700,
                        600,
                        300
                ],
        "meta":{
            "PV_Inv_Max_Power":1000
        }
    }
}
```

- Then send this data to the /data_source/file endpoint

```
curl --request POST \
  --url http://ip_address:8080/v1/data_source/file \
  --header 'Cache-Control: no-cache' \
  --header 'Content-Type: application/json' \
  --data '{\r\n    "photovoltaic":{\r\n        "P_PV":[\r\n            700,\r\n            600,\r\n
300\r\n         ],\r\n        "meta":{\r\n            "PV_Inv_Max_Power":1000    \r\n        }\r\n    }\r\n}'
```

Using a custom word

If you don't want to use any reserved word, you can register your input data in the following way.

Remember that you can send the input data directly. It is entered as a list oj JSON values.

- From the API, you find out which is the data model for entering a custom registry. Inside of it you choose the one that you need. If you are interested in registering an input for the total PV power with the name "Power_Photovoltaic", you can enter this information. The example shows a list of input values, if you have more than one custom inputs.

```
{
    "generic":[
    {
        "name":"Power_Photovoltaic",
        "file":[
            700,
                        600,
                        300
        ]
    },
    {
        "name":"custom_name",
        "file":[
            700,
                        600,
                        300
        ]
    }

    ]
}
```

- Then send this data to the /data_source/file endpoint

```
curl --request POST \
  --url http://ip_address:8080/v1/data_source/file \
  --header 'Cache-Control: no-cache' \
  --header 'Content-Type: application/json' \
  --data '{\r\n    "generic":[\r\n    {\r\n        "name":"Power_Photovoltaic",\r\n        "file":
[\r\n            700,\r\n            600,\r\n            300\r\n        ]\r\n    },\r\n    {\r\n        "name":"
custom_name",\r\n        "file":[\r\n            700,\r\n            600,\r\n            300\r\n        ]
\r\n    }\r\n\r\n    ]\r\n}'
```

Defining Inputs through MQTT

Using POST /data_source/mqtt

The first step to define an input is to use the POST request of the endpoint /data_source/mqtt

Registering a data_source through MQTT

Using a reserved word

If you chose a reserved word in your optimization model because you are working with energy applications, you can register your input data in the following way:

- From the API, you find out which is the data model for entering a photovoltaic registry. Inside of it you choose the one that you need. If you are interested in registering an input for the total PV power for example, you choose P_PV. Because we want to enter meta data at the same time, we choose "meta" as well.

```
{
    "photovoltaic":{
        "P_PV":{
            "mqtt":{
                "host":"ip_address or name_host",
                "topic":"name_topic_input_data",
                "qos":1
            }
        },
        "meta":{
            "PV_Inv_Max_Power":1000     # Entered in Watt [W]
        }
    }
}
```

- Then send this data to the /data_source/mqtt endpoint

```
curl --request POST \
  --url http://ip_address:8080/v1/data_source/mqtt \
  --header 'Cache-Control: no-cache' \
  --header 'Content-Type: application/json' \
  --data '{\r\n    "photovoltaic":{\r\n        "P_PV":{\r\n            "mqtt":{\r\n                "host":"
ip_address or name_host",\r\n                "topic":"name_topic_input_data",\r\n                "qos":
1\r\n            }\r\n        },\r\n        "meta":{\r\n            "PV_Inv_Max_Power":1000     \r\n        }
\r\n    }\r\n}'
```

Using a custom word

If you don't want to use any reserved word, you can register your input data in the following way:

- From the API, you find out which is the data model for entering a custom registry. Inside of it you choose the one that you need. If you are interested in registering an input for the total PV power with the name "Power_Photovoltaic", you can enter this information. The example shows a list of input values, if you have more than one custom inputs.

```
{
    "generic":[
    {
        "name":"Power_Photovoltaic",
        "mqtt":{
            "host":"ip_address or name_host",
            "topic":"name_topic_input_data",
            "qos":1
        }
    },
    {
        "name":"custom_name",
        "mqtt":{
            "host":"string",
            "topic":"string",
            "qos":1
        }
    }

    ]
}
```

- Then send this data to the /data_source/mqtt endpoint

```
curl --request POST \
  --url http://ip_address:8080/v1/data_source/mqtt \
  --header 'Cache-Control: no-cache' \
  --header 'Content-Type: application/json' \
  --data '{\r\n    "generic":[\r\n    {\r\n        "name":"Power_Photovoltaic",\r\n        "mqtt":
{\r\n            "host":"ip_address or name_host",\r\n            "topic":"name_topic_input_data",
\r\n            "qos":1\r\n        }\r\n    },\r\n    {\r\n        "name":"custom_name",\r\n        "mqtt":
{\r\n            "host":"string",\r\n            "topic":"string",\r\n            "qos":1\r\n        }\r\n    }
\r\n\r\n    ]\r\n}'
```

Sending data through MQTT

For sending data in MQTT, you have to use SenML standard.

An example of it is:

```
[{"n":"P_PV", "v":1000.0, "u":"W", "t": 1540811392.0 }]
```

where the time is expressed in seconds.

Defining outputs (/control)

Starting/Stopping OFW (/command)

Tutorial

Installation

The easiest way to deploy OFW is to use docker engine and docker compose. You can deploy it at the moment on a raspberry pi 3 or on an amd-based machine (e.g. for windows or server)

Installation with docker compose on a raspberry pi 3

Installation with docker compose on a server or windows computer

Installation with docker compose on a raspberry pi 3

If you don't already have Docker Engine and Docker Compose, you need to install them before following these steps.

For installing docker engine and docker compose on a Raspberry Pi 3, you can use the following script:

https://code.linksmart.eu/projects/SPF/repos/optimization-framework/browse/scripts/install_docker.sh

1. Create and enter a directory

Create a directory with the name of your election.

```
mkdir NameOfFile
```

2. Get the docker-compose script

Download the docker compose script and copy it to the created directory.

Download from here and copy to the current directory. It can also be downloaded with the following command:

```
curl -O https://code.linksmart.eu/projects/SPF/repos/optimization-framework/raw/profess/docker-compose-arm.yml
```

3. Run docker-compose script

The `docker-compose` script creates and runs containers for OFW, predictions and redisdb. By default, the containers store their data in a directory named `docker` in the current directory.
Enter the following command from the current directory:

```
docker-compose -f docker-compose-arm.yml up
```

You should now be able to access the API of the OFW with http://raspberry_ip_address:8080/v1/ui.

To start the containers in background (detached mode), stop the containers with Ctrl+C (or `docker-compose down` if that was aborted) and run the following instead:

```
docker-compose -f docker-compose-arm.yml up -d
```

4. Reading the logs

If you want to read the logs from OFW, just enter the following command:

```
docker logs ofw
```

5. Stopping the containers

To stop all containers:

```
docker-compose -f docker-compose-arm.yml down
```

5. Getting started with OFW

The next steps are related to the registry of inputs and outputs and the start of the framework. You can find information in API description or in Tutorial.

Installation with docker compose on a server or windows computer

If you don't already have Docker Engine and Docker Compose, you need to install them before following these steps.

For installing docker engine and docker compose on a windows computer, you can use the following script:

Install Docker for Windows

1. Create and enter a directory

Create a directory with the name of your election.

```
mkdir NameOfFile
```

2. Get the docker-compose script

Download the docker compose script and copy it to the created directory.

Download from here and copy to the current directory. It can also be downloaded with the following command:

```
curl -O https://code.linksmart.eu/projects/SPF/repos/optimization-framework/raw/profess/docker-compose-amd.yml
```

3. Run docker-compose script

The `docker-compose` script creates and runs containers for OFW, predictions and redisdb. By default, the containers store their data in a directory named `docker` in the current directory.
Enter the following command from the current directory:

```
docker-compose -f docker-compose-amd.yml up
```

You should now be able to access the API of the OFW with http://ip_address:8080/v1/ui.

To start the containers in background (detached mode), stop the containers with Ctrl+C (or `docker-compose down` if that was aborted) and run the following instead:

```
docker-compose -f docker-compose-amd.yml up -d
```

4. Reading the logs

If you want to read the logs from OFW, just enter the following command:

```
docker logs ofw
```

5. Stopping the containers

To stop all containers:

```
docker-compose -f docker-compose-amd.yml down
```

5. Getting started with OFW

The next steps are related to the registry of inputs and outputs and the start of the framework. You can find information in API description or in Tutorial.

Getting started

Checking the API

Once you have installed the Optimization Framework and it is running, you can check its API by calling the following command in a browser:

```
http://raspberry_ip_address:8080/v1/ui
```

As a result you are to going to see the API interface of the OFW. It should look like as here.

Registering inputs

Inputs can be understood as being the "Parameters" in the optimization model, in which the inputs of the optimization model will be received.

1. **POST request to register the inputs to OFW**

You should start using the POST request.

In the case OFW is going to receive data via MQTT use POST /data_source/mqt. You can find examples of how to do it here.

In the case OFW is going to receive data via HTTP use POST /data_source/file. You can find examples of to do it here.

Once you send this command, you are going to receive an id, which will identify the OFW object you are creating.

WARNING:

Copy this id or save it into memory for the nex steps. You cannot do anything else, if you lose this id.

2. PUT request to add extra inputs

If you forgot to register something with the POST request. Or you just need to add another input because your system expanded. Use then the PUT request to easily create new registries for these new inputs.

The PUT request allows also the use of MQTT input data together with HTTP input data. In this case, if you registered some inputs with MQTT you can add extra inputs using HTPP or viceversa.

You just need the id you received while you used the POST request.

In case of using MQTT, you can find examples of how to do it here.

In case of using HTTP, you can find examples of how to do it here.

3. Deleting input registries

If you want to delete registries that you entered for any reason, use the DELETE request.

In case of using MQTT, you can find examples of how to do it here.

In case of using HTTP, you can find examples of how to do it here.

Registering control outputs

Outputs can be understood as being the "Variables" in the optimization model, in which the results of the optimization will be stored.

The outputs are either setpoints that a driver or other software mechanism can use for accomplishing a control task or files that can be accessed through HTTP requests.

1. PUT request to add extra MQTT outputs

If you want to register an output or different outputs of the optimization problem to be published in an MQTT Topic, use the PUT request of the /control endpoint.

You just need the id you received while you used the POST request.

You can find examples of how to do it here.

2. Deleting MQTT output registries

If you want to delete registries that you entered for any reason, use the DELETE request.

You just need the id you received while you used the POST request.

You can find examples of how to do it here.

3. Obtaining results through HTTP

If you want to obtain all results from the optimization through HTTP, use the GET /control/file/{id}

{id} is the id you received while you sent the POST request.

You can find examples of how to do it here.

Starting/Stopping the OFW

If you want to start the OFW, you need to enter some optimization parameters:

- Optimization horizon: Defined time window to be used by the optimization
- Model name: Name of the optimization model to be loaded
- Repetitions: Number of repetitions that the software will solve the optimization problem
- Time step: Frequency of calculating the optimization problem
- Solver: Name of the solver to be chosen for solving the opitmization problem. At the moment there are three solvers present:
    - GLPK: Linear problems
    - Ipopt: Quadratic problems
    - Bonmin: Non linear problems

Don't forget to use the id for starting the optimization.

You can find examples of how to do it here.

For stopping OFW, you just need the id.

You can find examples of how to do it here.

Introducing custom optimization models

1. Obtaining the names of the optimization models saved into the internal repository

If you want to obtain the names of the optimization models saved into the OFW's repository, use the GET /models

You can find examples of how to do it here.

2. Introducing a custom optimization model

If you want to introduce a new custom optimization model into the OFW's repository, use the PUT /models/upload/{name}

where:

- {name}: Custom name of the optimization model. This is the new that you have to choose later, while starting the OFW.

OFW is based on abstract models written in Pyomo. For this reason you can use all your abstract models written in pyomo and convert them in an online optimal control.

In the model you specify:

- Paremeters: Parameters names have to coincide with the input registry names in /data_source

- Variables: Variables names have to coincide with the output registry names in /control. (If you don't want to have an output, you don't have to register anything. You can just read the results)
- Constrains
- Optimization objectives

You can find examples of how to do it here.

API Description

The Ootimization Framework REST API exposes the following top-level endpoints:

- /data_source: Registry of data sources used by the optimization model. The data sources can be registered as MQTT or HTTP inputs.
- /control: Registry of the outputs of the optimization framework, which will work as control set-points for other devices (if registered as MQTT) or as output files (if registered as file)
- /command: Start and stop commands for the framework, which allow the entry of some necessary optimization parameters.
- /models: Insertion of custom optimization models that are the base for the optimal control.

OFW was developed for the Storage4Grid project, which is related to the Energy domain. Consequently, data models present in the API are oriented to this domain. Nevertheless, the user is also able to enter custom nomenclature, depending on the application for which the OFW is going to be used.

command

control

data_source

models

Reserved words

Reserved words for parameters

ESS:

- SoC_Value
- ESS_Capacity
- ESS_Charging_Eff
- ESS_Discharging_Eff
- ESS_Max_Charge_Power
- ESS_Max_Discharge_Power
- ESS_Max_SoC
- ESS_Min_SoC

Global Control:

- ESS_Control

Grid:

- Max_Voltage_Drop
- Min_Voltage_Drop
- P_Grid_Max_Export_Power
- Q_Grid_Max_Export_Power

Load:

- P_Load
- P_Load_R
- P_Load_S
- P_Load_T
- Q_Load
- Q_Load_R
- Q_Load_S
- Q_Load_T
- pf_Load

Photovoltaic:

- P_PV
- P_PV_R
- P_PV_S
- P_PV_T
- Q_PV
- Q_PV_R
- Q_PV_S
- Q_PV_T
- PV_Inv_Max_Power

Reserved words for variables

ESS:

- P_ESS_Output

Grid:

- P_Grid_Output
- P_Grid_R_Output
- P_Grid_S_Output
- P_Grid_T_Output
- Q_Grid_Output
- Q_Grid_R_Output
- Q_Grid_S_Output
- Q_Grid_T_Output

Photovoltaic:

- P_PV_Output
- P_PV_R_Output
- P_PV_S_Output
- P_PV_T_Output
- Q_PV_Output
- Q_PV_R_Output
- Q_PV_S_Output
- Q_PV_T_Output