
OMEGAAlpes Documentation

Release 0.0.1

B. DELINCHANT, S. HODENCQ, Y. MARECHAL, L. MORRIET, C. PA

Mar 23, 2020

Contents

| | | |
|----------|----------------------------------|-----------|
| 1 | Introduction to OMEGAlpes | 1 |
| 2 | Contents | 3 |
| 3 | Acknowledgments | 51 |
| | Python Module Index | 53 |
| | Index | 55 |

CHAPTER 1

Introduction to OMEGAAlpes

OMEGAAlpes stands for Generation of Optimization Models As Linear Programming for Energy Systems. It aims to be an energy systems modelling tool for linear optimisation (LP, MILP). It is currently based on the LP modeler PuLP.

It is an Open Source project located on GitLab at [OMEGAAlpes Gitlab](#)

2.1 OMEGAlpes Installation

- *Install OMEGAlpes*
- *Other installation requirements*
- *Install OMEGAlpes as a developer*

2.1.1 Install OMEGAlpes

Do not hesitate to listen to a really nice music to be sure... it's going to work!

Python 3.6.0

Please use Python 3.6.0 for the project interpreter: [Python 3.6](#)

pip install omegalpes

Please install OMEGAlpes Lib with pip using on of the following the command prompt:

- **If you are admin on Windows or working on a virtual environment:**

```
pip install omegalpes
```

- **If you want a local installation or you are not admin:**

```
pip install --user omegalpes
```

- **If you are admin on Linux:**

```
sudo pip install omegAlpes
```

Then, you can download (or clone) the OMEGAAlpes Examples folder (repository) at : [OMEGAAlpes Examples](#) Make shure that the name of the examples folder is: “omegalpes_examples”.

Launch the examples (with Pycharm for instance) to understand how the OMEGAAlpes Lib works. Remember that the examples are presented at : [OMEGAAlpes Examples Documentation](#)

Enjoy your time using OMEGAAlpes !

2.1.2 Other installation requirements

If the music was enough catchy, the following libraries should be already installed. If not, increase the volume and install the following libraries with the help below.

- **PuLP >= 1.6.10**

PuLP is an LP modeler written in python. PuLP can generate MPS or LP files and call GLPK, COIN CLP/CBC, CPLEX, and GUROBI to solve linear problems : [PuLP](#)

- **Matplotlib >= 2.2.2**

Matplotlib is a Python 2D plotting library : [Matplotlib](#)

- **Numpy >= 1.14.2**

NumPy is the fundamental package needed for scientific computing with Python. [Numpy](#)

- **Pandas >= 0.22.0**

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. [Pandas](#)

— **Command lover** —

```
pip install <library_name>==version
```

If required, the command to upgrade the library is

```
pip install --upgrade <library_name>
```

— **Pycharm lover** —

Install automatically the library using pip with Pycharm on “File”, “settings...”, “Project Interpreter”, “+”, and choosing the required library

2.1.3 Install OMEGAAlpes as a developer

Installation as a developer and local branch creation

Absolute silence, keep calm and stay focus... you can do it! :<https://www.youtube.com/watch?v=g4mHPeMGTJM>

1. Create a new folder in the suitable path, name it as you wish for instance : OMEGAAlpes
2. Clone the OMEGAAlpes library repository

— **Command lover** —

```
git clone https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/  
↪omegalpes.git
```


— **Pycharm lover** —

Open Pycharm

On the Pycharm window, click on “Check out from version control” then choose “Git”.

A “clone repository” window open.

Copy the following link into the URL corresponding area:

<https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/omegalpes.git>

Copy the path of the new folder created just before.

Test if the connection to the git works and if it works click on “Clone”.

Once OMEGAAlpes is cloned, you must be able to see the full OMEGAAlpes library on Pycharm or on another development environment.

If the connection does not work and if you are working with local protected network, please try again with the wifi.

3. First, choose or change your project interpreter

— **Pycharm lover** —

Click on the yellow warning link or go to “File”, “settings...”, “Project Interpreter”

You can:

- either select the “Python 3.6” project interpreter but you may change the version of some library that you could use for another application.
- either create a virtual environment in order to avoid this problem (recommended).

Click on the star wheel near the project interpreter box.

Click on “add...”.

Select “New environment” if it not selected.

The location is pre-filled, if not fill it with the path of the folder as folder_path/venv

Select “Python 3.6” as your base interpreter

Then click on “Ok”

4. You can install the library on developing mode using the following command in command prompt once your are located it on the former folder. If you are calling OMEGAAlpes library in another project, the following command enables you to refer to the OMEGAAlpes library you are developing:

```
python setup.py develop
```

5. If it is not already done, install the library requirements.

— **Command lover** —

```
pip install <library_name>
```

If required, the command to upgrade the library is

```
pip install --upgrade <library_name>
```

— **Pycharm lover** —

You should still have a yellow warning. You can:

- install automatically the libraries clicking on the yellow bar.
- install automatically the library using pip with Pycharm on “File”, “settings...”, “Project Interpreter”, “+”, and choose the required library as indicated in the Library Installation Requirements part.

6. Finally, you can create your own local development branch.

— **Command lover** —

```
git branch <branch_name>
```

— **Pycharm lover** —

By default you are on a local branch named master.

Click on “Git: master” located on the bottom write of Pycharm

Select “+ New Branch”

Name the branch as you convenience for instance “dev_your_name”

7. Do not forget to “rebase” regularly to update your version of the library.

— **Command lover** —

```
git rebase origin
```

— **Pycharm lover** —

To do so, click on your branch name on the bottom write of the Pycharm window select “Origin/master” and click on “Rebase current onto selected”

If you want to have access to examples and study cases, download (or clone) the OMEGAAlpes Examples folder (repository) from : [OMEGAAlpes Examples](#) . Make shure that the name of the examples folder is: “omegalpes_examples”. Remember that the examples are presented at : [OMEGAAlpes Examples Documentation](#)

Enjoy your time developing OMEGAAlpes!

2.2 OMEGAAlpes structure

The models are bases on **general**, **energy** and **actor** classes. The structure of the library is described on the following class diagram:

The energy units are detailed in the energy package

2.2.1 energy package

The energy package gathers the energy units, poles and nodes modules:

- *Energy_units module*

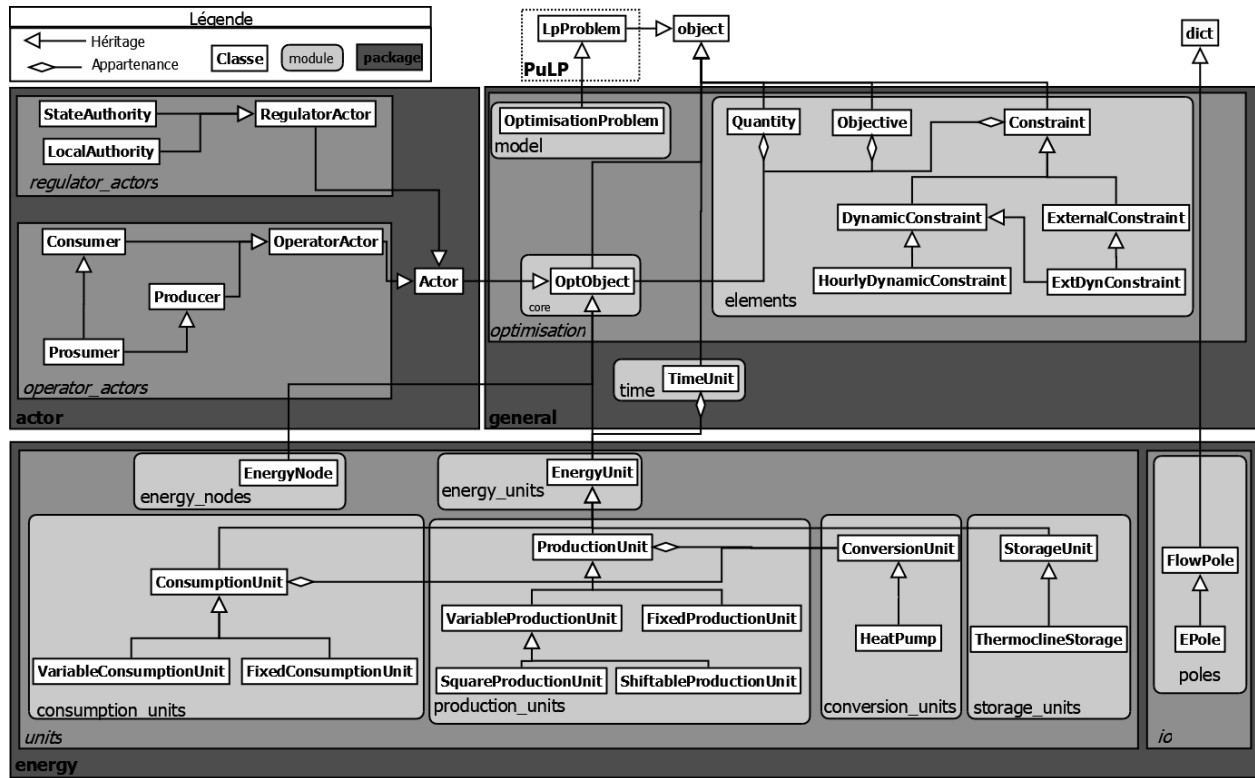


Fig. 1: Figure: OMEGAAlpes class diagram

- *Consumption_units module*
- *Production_units module*
- *Conversion_units module*
- *Storage_units module*
- *Energy_nodes module*
- *Thermal Building module*
- *Poles module*

The energy units inherit from the `EnergyUnit` object which itself inherit from the `Unit` object.

Energy_units module

This module defines the energy units of OMEGAAlpes. The production, consumption and storage unit will inherit from it.

The energy_units module defines the basic attributes and methods of an energy unit in OMEGAAlpes.

It includes the following attributes and quantities:

- `p` : instantaneous power of the energy unit (kW)
- `p_min` : minimal power (kW)
- `p_max` : maximal power (kW)

- `e_tot` : total energy during the time period (kWh)
- `e_min` : minimal energy of the unit (kWh)
- `e_max` : maximal energy of the unit (kWh)
- `u` : binary describing if the unit is operating or not at `t` (delivering or consuming P)

```
class omegalpes.energy.units.energy_units.EnergyUnit (time,                                name,  
                                                    flow_direction='in',  
                                                    p=None, p_min=-10000.0,  
                                                    p_max=10000.0, e_min=-  
                                                    1000000.0, e_max=1000000.0,  
                                                    starting_cost=None,  
                                                    operating_cost=None,  
                                                    min_time_on=None,  
                                                    min_time_off=None,  
                                                    max_ramp_up=None,  
                                                    max_ramp_down=None,  
                                                    co2_out=None,                                avail-  
                                                    ability_hours=None,  
                                                    energy_type=None,  
                                                    no_warn=True,                                ver-  
                                                    bose=True)
```

Bases: `omegalpes.general.optimisation.core.OptObject`

Description

Module dedicated to the parent class (EnergyUnit) of :

- production units
- consumption units
- storage units

`add_availability` (*av_hours*)

`add_co2_emissions` (*co2_out*)

`add_max_ramp_down` (*max_ramp_down*)

`add_max_ramp_up` (*max_ramp_up*)

`add_min_time_off` (*min_time_off*)

`add_min_time_on` (*min_time_on*)

`add_operating_cost` (*operating_cost*)

`add_starting_cost` (*start_cost*)

`minimize_co2_emissions` (*weight=1*)

Parameters **weight** – Weight coefficient for the objective

`minimize_costs` (*weight=1*)

Parameters **weight** – Weight coefficient for the objective

`minimize_energy` (*weight=1*)

Parameters **weight** – Weight coefficient for the objective

`minimize_operating_cost` (*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_starting_cost (*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_time_of_use (*weight=1*)

Parameters **weight** – Weight coefficient for the objective

set_energy_limits_on_time_period (*e_min=0, e_max=None, start='YYYY-MM-DD HH:MM:SS', end='YYYY-MM-DD HH:MM:SS', period_index=None*)

Parameters

- **e_min** – Minimal energy set during the time period (int or float)
- **e_max** – Maximal energy set during the time period (int or float)
- **start** – Date of start of the time period YYYY-MM-DD HH:MM:SS (str)
- **end** – Date of end of the time period YYYY-MM-DD HH:MM:SS (str)

```
class omegalpes.energy.units.energy_units.FixedEnergyUnit (time, name: str, p: list,  
flow_direction='in',  
starting_cost=None,  
operating_cost=None,  
co2_out=None, en-  
ergy_type=None,  
verbose=True)
```

Bases: *omegalpes.energy.units.energy_units.EnergyUnit*

Description

Energy unit with a fixed power profile.

Attributs

- **p** : instantaneous power known by advance (kW)
- **energy_type** : type of energy ('Electrical', 'Heat', ...)

```
class omegalpes.energy.units.energy_units.SawtoothEnergyUnit (time, name,  
flow_direction,  
p_peak, p_low,  
alpha_peak,  
t_triangle,  
t_sawtooth,  
mandatory=True, start-  
ing_cost=None,  
operat-  
ing_cost=None,  
co2_out=None, en-  
ergy_type=None,  
verbose=True)
```

Bases: *omegalpes.energy.units.energy_units.ShiftableEnergyUnit*

```
class omegalpes.energy.units.energy_units.SeveralEnergyUnit (time, name,
                                                             fixed_power,
                                                             pmin=1e-05,
                                                             pmax=100000.0,
                                                             imaginary=False,
                                                             e_min=0,
                                                             e_max=1000000.0,
                                                             nb_unit_min=0,
                                                             nb_unit_max=None,
                                                             flow_direction='in',
                                                             starting_cost=None,
                                                             operat-
                                                             ing_cost=None,
                                                             max_ramp_up=None,
                                                             max_ramp_down=None,
                                                             co2_out=None, en-
                                                             ergy_type=None,
                                                             verbose=True,
                                                             no_warn=True)
```

Bases: *omegalpes.energy.units.energy_units.VariableEnergyUnit*

Description

Energy unit based on a fixed power curve enabling to multiply several times (nb_unit) the same power curve.

Be careful, if imaginary == True, the solution may be imaginary as nb_unit can be continuous. The accurate number of the power unit should be calculated later

Attributs

- fixed_power : fixed power curve

```
class omegalpes.energy.units.energy_units.ShiftableEnergyUnit (time, name: str,
                                                                flow_direction,
                                                                power_values,
                                                                mandatory=True,
                                                                co2_out=None,
                                                                start-
                                                                ing_cost=None,
                                                                operat-
                                                                ing_cost=None,
                                                                en-
                                                                ergy_type=None,
                                                                verbose=True)
```

Bases: *omegalpes.energy.units.energy_units.VariableEnergyUnit*

Description

EnergyUnit with shiftable power profile.

Attributs

- power_values : power profile to shift (kW)
- mandatory : indicates if the power is mandatory (True) or not (False)
- starting_cost : cost of the starting of the EnergyUnit
- operating_cost : cost of the operation (€/kW)

- `energy_type` : type of energy ('Electrical', 'Heat', ...)

```
class omegalpes.energy.units.energy_units.SquareEnergyUnit (time,          name,
                                                             p_square,  n_square,
                                                             t_between_sq,
                                                             t_square=1,
                                                             flow_direction='in',
                                                             starting_cost=None,
                                                             operating_cost=None,
                                                             co2_out=None,  en-
                                                             ergy_type=None,
                                                             verbose=True,
                                                             no_warn=True)

Bases: omegalpes.energy.units.energy_units.VariableEnergyUnit
```

```
class omegalpes.energy.units.energy_units.TriangleEnergyUnit (time,          name,
                                                                flow_direction,
                                                                p_peak,      al-
                                                                pha_peak,
                                                                t_triangle,  manda-
                                                                tory=True,   start-
                                                                ing_cost=None,
                                                                operat-
                                                                ing_cost=None,
                                                                co2_out=None, en-
                                                                ergy_type=None,
                                                                verbose=True)

Bases: omegalpes.energy.units.energy_units.ShiftableEnergyUnit
```

```
class omegalpes.energy.units.energy_units.VariableEnergyUnit (time,          name,
                                                                flow_direction='in',
                                                                p_min=-10000.0,
                                                                p_max=10000.0,
                                                                e_min=-
                                                                1000000.0,
                                                                e_max=1000000.0,
                                                                start-
                                                                ing_cost=None,
                                                                operat-
                                                                ing_cost=None,
                                                                min_time_on=None,
                                                                min_time_off=None,
                                                                max_ramp_up=None,
                                                                max_ramp_down=None,
                                                                co2_out=None,
                                                                availabil-
                                                                ity_hours=None,
                                                                en-
                                                                ergy_type=None,
                                                                verbose=True,
                                                                no_warn=True)

Bases: omegalpes.energy.units.energy_units.EnergyUnit
```

Consumption_units module

This module defines the consumption units

The `consumption_units` module defines various classes of consumption units, from generic to specific ones.

It includes :

- `ConsumptionUnit` : simple consumption unit. It inherits from `EnergyUnit`, its power flow direction is always 'in'.

3 Objectives are also available :

- minimize consumption,
 - maximize consumption,
 - minimize consumption costs.
- `FixedConsumptionUnit` : consumption with a fixed load profile. It inherits from `ConsumptionUnit`.
 - `VariableConsumptionUnit` : consumption unit allowing for a variation of power between `pmin` et `pmax`. It inherits from `ConsumptionUnit`.

```
class omegalpes.energy.units.consumption_units.ConsumptionUnit (time,      name,
                                                                p=None,
                                                                p_min=1e-05,
                                                                p_max=100000.0,
                                                                e_min=0,
                                                                e_max=1000000.0,
                                                                co2_out=None,
                                                                start-
                                                                ing_cost=None,
                                                                consump-
                                                                tion_cost=None,
                                                                min_time_on=None,
                                                                min_time_off=None,
                                                                max_ramp_up=None,
                                                                max_ramp_down=None,
                                                                availabil-
                                                                ity_hours=None,
                                                                en-
                                                                ergy_type=None,
                                                                verbose=True)
```

Bases: `omegalpes.energy.units.energy_units.EnergyUnit`

Description

Simple Consumption unit

Attributes

- `p` : instantaneous power demand (kW)
- `pmax` : maximal instantaneous power demand (kW)
- `pmin` : minimal instantaneous power demand (kW)
- `energy_type` : type of energy ('Electrical', 'Heat', ...)
- `consumption_cost` : cost associated to the energy consumption

maximize_consumption (*weight=1*)

Parameters `weight` – Weight coefficient for the objective

`minimize_consumption (weight=1)`

Parameters **weight** – Weight coefficient for the objective

`minimize_consumption_cost (weight=1)`

Parameters **weight** – Weight coefficient for the objective

```
class omegalpes.energy.units.consumption_units.FixedConsumptionUnit (time,
                                                                    name,
                                                                    p: list
                                                                    = None,
                                                                    co2_out=None,
                                                                    start-
                                                                    ing_cost=None,
                                                                    operat-
                                                                    ing_cost=None,
                                                                    en-
                                                                    ergy_type=None,
                                                                    ver-
                                                                    bose=True)
```

Bases: `omegalpes.energy.units.energy_units.FixedEnergyUnit`, `omegalpes.energy.units.consumption_units.ConsumptionUnit`

Description

Consumption unit with a fixed consumption profile.

Attributes

- `p` : instantaneous power demand known in advance (kW)
- `energy_type` : type of energy ('Electrical', 'Heat', ...)
- `consumption_cost` : cost associated to the energy consumption

```
class omegalpes.energy.units.consumption_units.SeveralConsumptionUnit (time,
                                                                    name,
                                                                    fixed_cons,
                                                                    pmin=1e-
                                                                    05,
                                                                    pmax=100000.0,
                                                                    e_min=0,
                                                                    e_max=1000000.0,
                                                                    nb_unit_min=0,
                                                                    nb_unit_max=None,
                                                                    co2_out=None,
                                                                    start-
                                                                    ing_cost=None,
                                                                    op-
                                                                    erat-
                                                                    ing_cost=None,
                                                                    max_ramp_up=None,
                                                                    max_ramp_down=None,
                                                                    en-
                                                                    ergy_type=None,
                                                                    ver-
                                                                    bose=True,
                                                                    no_warn=True)
```

Bases: `omegalpes.energy.units.consumption_units.VariableConsumptionUnit`,

omegalpes.energy.units.energy_units.SeveralEnergyUnit

Description

Consumption unit based on a fixed consumption curve enabling to multiply several times (nb_unit) the same consumption curve.

Attributes

- fixed_cons : fixed consumption curve

```
class omegalpes.energy.units.consumption_units.SeveralImaginaryConsumptionUnit (time,  
                                         name,  
                                         fixed_cons,  
                                         pmin=1e-05,  
                                         pmax=100000.0,  
                                         e_min=0,  
                                         e_max=1000000,  
                                         nb_unit_min=0,  
                                         nb_unit_max=None,  
                                         co2_out=None,  
                                         start-  
                                         ing_cost=None,  
                                         op-  
                                         er-  
                                         at-  
                                         ing_cost=None,  
                                         max_ramp_up=None,  
                                         max_ramp_down=None,  
                                         en-  
                                         ergy_type=None,  
                                         ver-  
                                         bose=True,  
                                         no_warn=True)
```

Bases: *omegalpes.energy.units.consumption_units.VariableConsumptionUnit*,
omegalpes.energy.units.energy_units.SeveralEnergyUnit

Description

Consumption unit based on a fixed consumption curve enabling to multiply several times (nb_unit) the same consumption curve. Be careful, the solution may be imaginary as nb_unit can be continuous. The accurate number of the consumption unit should be calculated later.

Attributes

- fixed_cons : fixed consumption curve

```

class omegalpes.energy.units.consumption_units.ShiftableConsumptionUnit (time,
                                                                    name:
                                                                    str,
                                                                    power_values,
                                                                    mandatory=True,
                                                                    co2_out=None,
                                                                    starting_cost=None,
                                                                    operating_cost=None,
                                                                    energy_type=None,
                                                                    verbose=True)

Bases:    omegalpes.energy.units.energy_units.ShiftableEnergyUnit, omegalpes.
energy.units.consumption_units.VariableConsumptionUnit

```

Description

Consumption unit with shiftable consumption profile.

Attributes

- power_values : consumption profile to shift (kW)
- mandatory : indicates if the consumption is mandatory (True) or not

(False) * starting_cost : cost of the starting of the consumption * operating_cost : cost of the operation (€/kW) * energy_type : type of energy ('Electrical', 'Heat', ...)

```

class omegalpes.energy.units.consumption_units.SquareConsumptionUnit (time,
                                                                    name,
                                                                    p_square,
                                                                    duration,
                                                                    n_square,
                                                                    t_between_sq,
                                                                    co2_out=None,
                                                                    starting_cost=None,
                                                                    operating_cost=None,
                                                                    energy_type=None,
                                                                    verbose=True,
                                                                    no_warn=False)

Bases:    omegalpes.energy.units.energy_units.SquareEnergyUnit, omegalpes.
energy.units.consumption_units.VariableConsumptionUnit

```

Description

Consumption unit with a fixed value and fixed duration.
Only the time of beginning can be modified
Operation can be mandatory or not

Attributs

- `p` : instantaneous power consumption (kW)
- `duration` : duration of the power delivery (hours)
- `mandatory` : indicates if the power delivery is mandatory or not
- `energy_type` : type of energy ('Electrical', 'Heat', ...)
- `consumption_cost` : cost associated to the energy consumption

```
class omegalpes.energy.units.consumption_units.VariableConsumptionUnit (time,  
                                                                           name,  
                                                                           pmin=1e-  
                                                                           05,  
                                                                           pmax=100000.0,  
                                                                           e_min=0,  
                                                                           e_max=1000000.0,  
                                                                           co2_out=None,  
                                                                           start-  
                                                                           ing_cost=None,  
                                                                           op-  
                                                                           er-  
                                                                           at-  
                                                                           ing_cost=None,  
                                                                           min_time_on=None,  
                                                                           min_time_off=None,  
                                                                           max_ramp_up=None,  
                                                                           max_ramp_down=None,  
                                                                           en-  
                                                                           ergy_type=None,  
                                                                           ver-  
                                                                           bose=True,  
                                                                           no_warn=True)
```

Bases: `omegalpes.energy.units.energy_units.VariableEnergyUnit`, `omegalpes.energy.units.consumption_units.ConsumptionUnit`

Description

Consumption unit with a variation of power between `pmin` et `pmax`.

Attributs

- `pmax` : maximal instantaneous power consumption (kW)
- `pmin` : minimal instantaneous power consumption (kW)
- `energy_type` : type of energy ('Electrical', 'Heat', ...)

Production_units module**This module defines the production units**

The `production_units` module defines various kinds of production units with associated attributes and methods.

It includes :

- `ProductionUnit` : simple production unit inheriting from `EnergyUnit` and with an outer flow direction. The outside `co2` emissions, the starting cost, the operating cost, the minimal operating time, the minimal non-operating time, the maximal increasing ramp and the maximal decreasing ramp can be filled.

Objectives are also available :

- minimize starting cost, operating cost, total cost
- minimize production, co2_emissions, time of use
- maximize production
- FixedProductionUnit : Production unit with a fixed production profile.
- VariableProductionUnit : Production unit with a variation of power between pmin et pmax.

And also :

- SeveralProductionUnit: Production unit based on a fixed production curve enabling to multiply several times (nb_unit) the same production curve
- SeveralImaginaryProductionUnit: Production unit based on a fixed production curve enabling to multiply several times (nb_unit) the same production curve. Be careful, the solution may be imaginary as nb_unit can be continuous. The accurate number of the production unit should be calculated later
- SquareProductionUnit: Production unit with a fixed value and fixed duration.
- ShiftableProductionUnit: Production unit with shiftable production profile.

```
class omegalpes.energy.units.production_units.FixedProductionUnit (time, name:
                                                                    str,      p:
                                                                    list = None,
                                                                    co2_out=None,
                                                                    start-
                                                                    ing_cost=None,
                                                                    operat-
                                                                    ing_cost=None,
                                                                    en-
                                                                    ergy_type=None,
                                                                    ver-
                                                                    bose=True)
```

Bases: *omegalpes.energy.units.energy_units.FixedEnergyUnit, omegalpes.energy.units.production_units.ProductionUnit*

Description

Production unit with a fixed production profile.

Attributes

- p : instantaneous power production known by advance (kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)

```
class omegalpes.energy.units.production_units.ProductionUnit (time, name,
                                                                p=None,
                                                                p_min=1e-05,
                                                                p_max=100000.0,
                                                                e_min=0,
                                                                e_max=1000000.0,
                                                                co2_out=None,
                                                                start-
                                                                ing_cost=None,
                                                                operat-
                                                                ing_cost=None,
                                                                min_time_on=None,
                                                                min_time_off=None,
                                                                max_ramp_up=None,
                                                                max_ramp_down=None,
                                                                availabil-
                                                                ity_hours=None,
                                                                en-
                                                                ergy_type=None,
                                                                verbose=True,
                                                                no_warn=True)
```

Bases: *omegalpes.energy.units.energy_units.EnergyUnit*

Description

Simple Production unit

Attributes

- `co2_out`: outside co2 emissions
- `starting_cost`: the starting cost
- `operating_cost`: the operating cost
- `min_time_on` : the minimal operating time
- `min_time_off` : the minimal non-operating time
- `max_ramp_up` : the maximal increasing ramp
- `max_ramp_down` ; the maximal decreasing ramp

maximize_production (*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_production (*weight=1*)

Parameters **weight** – Weight coefficient for the objective

```
class omegalpes.energy.units.production_units.SeveralImaginaryProductionUnit (time,  
                                                                           name,  
                                                                           fixed_prod,  
                                                                           pmin=1e-  
                                                                           05,  
                                                                           pmax=100000.0,  
                                                                           e_min=0,  
                                                                           e_max=1000000.0,  
                                                                           nb_unit_min=0,  
                                                                           nb_unit_max=None,  
                                                                           co2_out=None,  
                                                                           start-  
                                                                           ing_cost=None,  
                                                                           op-  
                                                                           er-  
                                                                           at-  
                                                                           ing_cost=None,  
                                                                           max_ramp_up=None,  
                                                                           max_ramp_down=None,  
                                                                           en-  
                                                                           ergy_type=None,  
                                                                           ver-  
                                                                           bose=True,  
                                                                           no_warn=True)
```

Bases: `omegalpes.energy.units.production_units.VariableProductionUnit`,
`omegalpes.energy.units.energy_units.SeveralEnergyUnit`

Description

Production unit based on a fixed production curve enabling to multiply several times (nb_unit) the same production curve. Be careful, the solution may be imaginary as nb_unit can be continuous. The accurate number of the production unit should be calculated later.

Attributes

- fixed_prod : fixed production curve

```
class omegalpes.energy.units.production_units.SeveralProductionUnit (time,  
                                                                    name,  
                                                                    fixed_prod,  
                                                                    pmin=1e-  
                                                                    05,  
                                                                    pmax=100000.0,  
                                                                    e_min=0,  
                                                                    e_max=1000000.0,  
                                                                    nb_unit_min=0,  
                                                                    nb_unit_max=None,  
                                                                    co2_out=None,  
                                                                    start-  
                                                                    ing_cost=None,  
                                                                    operat-  
                                                                    ing_cost=None,  
                                                                    max_ramp_up=None,  
                                                                    max_ramp_down=None,  
                                                                    en-  
                                                                    ergy_type=None,  
                                                                    ver-  
                                                                    bose=True,  
                                                                    no_warn=True)
```

Bases: `omegalpes.energy.units.production_units.VariableProductionUnit`,
`omegalpes.energy.units.energy_units.SeveralEnergyUnit`

Description

Production unit based on a fixed production curve enabling to multiply several times (nb_unit) the same production curve. nb_unit is an integer variable.

Attributes

- fixed_prod : fixed production curve

```
class omegalpes.energy.units.production_units.ShiftableProductionUnit (time,  
                                                                    name:  
                                                                    str,  
                                                                    power_values,  
                                                                    manda-  
                                                                    tory=True,  
                                                                    co2_out=None,  
                                                                    start-  
                                                                    ing_cost=None,  
                                                                    op-  
                                                                    erat-  
                                                                    ing_cost=None,  
                                                                    en-  
                                                                    ergy_type=None,  
                                                                    ver-  
                                                                    bose=True)
```

Bases: `omegalpes.energy.units.energy_units.ShiftableEnergyUnit`, `omegalpes.energy.units.production_units.VariableProductionUnit`

Description

Production unit with shiftable production profile.

Attributes

- `power_values` : production profile to shift (kW)
- `mandatory` : indicates if the production is mandatory : True or not : False
- `starting_cost` : cost of the starting of the production
- `operating_cost` : cost of the operation (€/kW)
- `energy_type` : type of energy ('Electrical', 'Heat', ...)

```
class omegalpes.energy.units.production_units.SquareProductionUnit (time,
                                                                    name,
                                                                    p_square,
                                                                    duration,
                                                                    n_square,
                                                                    t_between_sq,
                                                                    co2_out=None,
                                                                    start-
                                                                    ing_cost=None,
                                                                    operat-
                                                                    ing_cost=None,
                                                                    en-
                                                                    ergy_type=None,
                                                                    ver-
                                                                    bose=True,
                                                                    no_warn=True)
```

Bases: `omegalpes.energy.units.energy_units.SquareEnergyUnit`, `omegalpes.energy.units.production_units.VariableProductionUnit`

Description

Production unit with a fixed value and fixed duration.
 Only the time of beginning can be modified.
 Operation can be mandatory or not.

Attributes

- `p` : instantaneous power production (kW)
- `duration` : duration of the power delivery (hours)
- `mandatory` : indicates if the power delivery is mandatory or not
- `energy_type` : type of energy ('Electrical', 'Heat', ...)

```
class omegalpes.energy.units.production_units.VariableProductionUnit (time,
                                                                    name,
                                                                    pmin=1e-
                                                                    05,
                                                                    pmax=100000.0,
                                                                    e_min=0,
                                                                    e_max=1000000.0,
                                                                    co2_out=None,
                                                                    start-
                                                                    ing_cost=None,
                                                                    operat-
                                                                    ing_cost=None,
                                                                    min_time_on=None,
                                                                    min_time_off=None,
                                                                    max_ramp_up=None,
                                                                    max_ramp_down=None,
                                                                    en-
                                                                    ergy_type=None,
                                                                    ver-
                                                                    bose=True,
                                                                    no_warn=True)
                                                                    omegalpes.
```

Bases: `omegalpes.energy.units.energy_units.VariableEnergyUnit`, `omegalpes.energy.units.production_units.ProductionUnit`

Description

Production unit with a variation of power between pmin et pmax.

Attributes

- pmax : maximal instantaneous power production (kW)
- pmin : minimal instantaneous power production (kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)

Conversion_units module

This module defines the conversion units, with at least a production unit and a consumption unit and using one or several energy types

The conversion_units module defines various classes of conversion units, from generic to specific ones.

It includes :

- ConversionUnit : simple conversion unit. It inherits from OptObject.
- ElectricalToThermalConversionUnit : Electrical to thermal Conversion unit with an electricity consumption and a thermal production linked by and electrical to thermal ratio. It inherits from ConversionUnit
- HeatPump : Simple Heat Pump with an electricity consumption, a heat production and a heat consumption. It has a theoretical coefficient of performance COP and inherits from ConversionUnit.

```
class omegalpes.energy.units.conversion_units.ConversionUnit (time,      name,
                                                                prod_units=None,
                                                                cons_units=None,
                                                                verbose=True)
```

Bases: `omegalpes.general.optimisation.core.OptObject`

Description

Simple Conversion unit

Attributes

- `time` : TimeUnit describing the studied time period
- `prod_units` : list of the production units
- `cons_units` : list of the consumption units
- `poles` : dictionary of the poles of the conversion unit

```
class omegalpes.energy.units.conversion_units.ElectricalToThermalConversionUnit (time,
                                                                                   name,
                                                                                   pmin_in_elec=
                                                                                   05,
                                                                                   pmax_in_elec=
                                                                                   p_in_elec=None,
                                                                                   pmin_out_therm=
                                                                                   05,
                                                                                   pmax_out_therm=
                                                                                   p_out_therm=None,
                                                                                   elec_to_therm_
                                                                                   ver-
                                                                                   bose=True)
```

Bases: `omegalpes.energy.units.conversion_units.ConversionUnit`

Description

Electrical to thermal Conversion unit with an electricity consumption and a thermal production

Attributes

- `thermal_production_unit` : thermal production unit (thermal output)
- `elec_consumption_unit` : electricity consumption unit (electrical input)
- `conversion` : Dynamic Constraint linking the electrical input to the thermal output through the electrical to thermal ratio

```
class omegalpes.energy.units.conversion_units.HeatPump (time,
                                                                                   name,
                                                                                   pmin_in_elec=1e-05,
                                                                                   pmax_in_elec=100000.0,
                                                                                   p_in_elec=None,
                                                                                   pmin_in_therm=1e-05,
                                                                                   pmax_in_therm=100000.0,
                                                                                   p_in_therm=None,
                                                                                   pmin_out_therm=1e-05,
                                                                                   pmax_out_therm=100000.0,
                                                                                   p_out_therm=None, cop=3,
                                                                                   losses=0)
```

Bases: `omegalpes.energy.units.conversion_units.ConversionUnit`

Description

Simple Heat Pump with an electricity consumption, a thermal production and a thermal consumption. It has a theoretical coefficient of performance COP and inherits from ConversionUnit.

Attributes

- `thermal_production_unit` : thermal production unit (condenser)
- `elec_consumption_unit` : electricity consumption unit (electrical input)

- `thermal_consumption_unit` : heavy consumption unit (evaporator)
- `COP` : Quantity describing the coefficient of performance of the heat pump
- `conversion` : Dynamic Constraint linking the electrical input to the thermal output through the electrical to thermal ratio
- `power_flow` : Dynamic constraint linking the thermal output to the electrical and thermal inputs in relation to the losses.

Storage_units module

This module defines the storage units

The `storage_units` module defines various kinds of storage units with associated attributes and methods, from simple to specific ones.

It includes :

- `StorageUnit` : simple storage unit inheriting from `EnergyUnit`, with storage specific attributes. It includes the objective “minimize capacity”.
- `Thermocline storage` : a thermal storage that need to cycle (i.e. reach `SOC_max`) every period of `Tcycle`

```
class omegalpes.energy.units.storage_units.StorageUnit (time, name, pc_min=1e-05, pc_max=100000.0, pd_min=1e-05, pd_max=100000.0, capacity=None, e_0=None, e_f=None, soc_min=0, soc_max=1, eff_c=1, eff_d=1, self_disch=0, self_disch_t=0, ef_is_e0=False, cycles=None, energy_type=None)
```

Bases: `omegalpes.energy.units.energy_units.VariableEnergyUnit`

Description

Simple Storage unit

Attributes

- `capacity` (Quantity): maximal energy that can be stored
- `e` (Quantity): energy at time `t` in the storage
- `set_soc_min` (DynamicConstraint): constraining the energy to be above the value : `soc_min*capacity`
- `set_soc_max` (DynamicConstraint): constraining the energy to be below the value : `soc_max*capacity`
- `pc` (Quantity) : charging power
- `pd` (Quantity) : discharging power
- `u_c` (Quantity) : binary variable describing the charge of the storage unit : 0 : Not charging & 1 : charging
- `calc_e` (DynamicConstraint) : energy calculation at time `t` ; relation power/energy
- `calc_p` (DynamicConstraint) : power calculation at time `t` ; power flow equals charging power minus discharging power

- `on_off_stor` (DynamicConstraint) : making $u[t]$ matching with storage modes (on/off)
- `def_max_charging` (DynamicConstraint) : defining the max charging power, avoiding charging and discharging at the same time
- `def_max_discharging` (DynamicConstraint) : defining the max discharging power, avoiding charging and discharging at the same time
- `def_min_charging` (DynamicConstraint) : defining the min charging power, avoiding charging and discharging at the same time
- `def_min_discharging` (DynamicConstraint) : defining the min discharging power, avoiding charging and discharging at the same time
- `set_e_0` (ExternalConstraint) : set the energy state for $t=0$
- `e_f` (Quantity) : energy in the storage at the end of the time horizon, i.e. after the last time step
- `e_f_min` (Constraint) : e_f value is constrained above $soc_min \times capacity$
- `e_f_max` (Constraint) : e_f value is constrained below $soc_max \times capacity$
- `set_e_f` (Constraint) : when e_f is given, it is set in the same way the energy is, but after the last time step
- `calc_e_f` (Constraint) : when e_f is not given, it is calculated in the same way the energy is, but after the last time step
- `ef_is_e0` (ExternalConstraint) : Imposing $ef=e0$ on the time period.
- `cycles` (ExternalDynamicConstraint) : setting a cycle constraint $e[t] = e[t+cycles/dt]$

minimize_capacity (*weight=1*)

Parameters weight – Weight coefficient for the objective

```
class omegalpes.energy.units.storage_units.ThermoclineStorage (time,      name,
                                                                pc_min=1e-05,
                                                                pc_max=100000.0,
                                                                pd_min=1e-05,
                                                                pd_max=100000.0,
                                                                capacity=None,
                                                                e_0=None,
                                                                e_f=None,
                                                                soc_min=0,
                                                                soc_max=1,
                                                                eff_c=1, eff_d=1,
                                                                self_disch=0,
                                                                Tcycl=120,
                                                                ef_is_e0=False)
```

Bases: `omegalpes.energy.units.storage_units.StorageUnit`

Description

Class `ThermoclineStorage` : class defining a thermocline heat storage, inheriting from `StorageUnit`.

Attributes

- `is_soc_max` (Quantity) : indicating if the storage is fully charged 0:No 1:Yes
- `def_is_soc_max_inf` (DynamicConstraint) : setting the right value for `is_soc_max`
- `def_is_soc_max_sup` (DynamicConstraint) : setting the right value for `is_soc_max`
- `force_soc_max` (ExtDynConstraint) : The energy has to be at least once at its maximal value during the period `Tcycl`.

Energy_nodes module

This module defines the energy nodes that will allow energy transmission between the various energy units and conversion units

The energy_node module includes the EnergyNode class for energy transmission between production, consumption, conversion and storage. Defining several energy nodes and exporting/importing energy between them can also allow for a better demarcation of the energy system.

class omegalpes.energy.energy_nodes.**EnergyNode** (*time, name, energy_type=None, operator=None*)

Bases: *omegalpes.general.optimisation.core.OptObject*

This class defines an energy node.

add_connected_energy_unit (*unit*)

Add an EnergyUnit to the connected_units list

add_pole (*pole: omegalpes.energy.io.poles.Epole*) → None

Add an energy pole to the poles_list

Parameters **pole** – Epole

connect_units (**units*)

Connecting all EnergyUnit to the EnergyNode

Parameters **units** (*list*) – EnergyUnits connected to the EnergyNode

create_export (*node, export_min, export_max*)

Create the export from the EnergyNode (self) to the EnergyNode (node)

Parameters

- **node** – EnergyNode to whom power can be exported
- **export_min** – Minimal value of exported power when there is export
- **export_max** – Maximal value of exported power when there is export

Returns Quantity that defines the power exported

export_to_node (*node, export_min=1e-05, export_max=100000.0*)

Add an export of power from the node to another node

Parameters

- **node** – EnergyNode to whom power can be exported
- **export_min** – Minimal value of exported power when there is export
- **export_max** – Maximal value of exported power when there is export

get_connected_energy_units

Return the list of connected EnergyUnits in the EnergyNode

get_exports

Return the list of exports to the EnergyNode

get_flows

Get all the power flows of the energy node :rtype: list :return: list of power flows

get_imports

Return the list of imports to the EnergyNode

get_poles

Return the list of energy poles in the EnergyNode

```
import_from_node (node, import_min=1e-05, import_max=100000.0)
```

Parameters

- **node** – EnergyNode from whom power can be imported
- **import_min** – Minimal value of imported power when there is import
- **import_max** – Maximal value of imported power when there is import

```
is_export_flow (flow)
```

Get if the power flow is an export or not

```
is_import_flow (flow)
```

Get if the power flow is an import or not

```
set_power_balance ()
```

Set the power balance equation for the EnergyNode

Thermal Building module

This module enables to model buildings as thermal loads

```
class omegalpes.energy.buildings.thermal.HeatingLoad (time, name, tz,
                                                    p_max=10000000000000.0,
                                                    T_set=19, temp_margin=1,
                                                    no_cooling=True)
```

Bases: *omegalpes.energy.units.consumption_units.VariableConsumptionUnit*

```
maximize_thermal_comfort (T_op=None, weight=1)
```

Parameters

- **T_op** – Operative temperature wished for the maximal thermal comfort
- **weight** – Weight of the objective

```
class omegalpes.energy.buildings.thermal.RCNetwork_1 (time, name, T_ext,
                                                    theta_ec, theta_em,
                                                    T_int_min=0, T_int_max=50,
                                                    theta_ea=None, theta_c=None,
                                                    theta_m=None, h_ea=0,
                                                    h_ac=0, h_ec=0, h_mc=0,
                                                    h_em=0, c_m=0, f_im=None,
                                                    f_r_l=0.7, f_r_p=0.5,
                                                    f_r_a=0.2, f_sa=0.1,
                                                    f_sm=None, f_sc=None,
                                                    f_ic=None, f_hc_cv=None,
                                                    U_wall=0.2, U_win=1.2,
                                                    U_roof=0.2, e_wall=0.9,
                                                    e_win=0.9, e_roof=0.9,
                                                    a_wall=0.6, a_roof=0.6,
                                                    A_wall=None, A_win=None,
                                                    A_roof=None, owner=None)
```

Bases: *omegalpes.general.optimisation.core.OptObject*

```
class omegalpes.energy.buildings.thermal.ThermalZone (rc_network, phi_i_a,
                                                    phi_i_l, phi_i_p, I_sol_av,
                                                    Fsh_win, T_mean, T_dew,
                                                    sky_cover=1, T_ext=None,
                                                    hvac_prop=None)
```

Bases: *omegalpes.general.optimisation.core.OptObject*

split_heating_and_cooling (*p_max_heating=10000000000000.0*,
p_max_cooling=10000000000000.0)

class *omegalpes.energy.buildings.thermal.ZEA_RCNetwork_1* (*time*, *name*, *T_ext*,
A_f, *A_win*, *Aext_v*,
A_roof, *footprint*, *U_win*,
U_wall, *U_roof*, *U_base*,
floors, *e_wall=0.9*,
e_win=0.9, *e_roof=0.9*,
a_wall=0.6, *a_roof=0.6*,
construction='heavy',
height_bg=0,
perimeter=0,
f_hc_cv=1, *void=0*,
hvac_prop=None,
T_int_min=0,
T_int_max=50,
owner=None)

Bases: *omegalpes.energy.buildings.thermal.RCNetwork_1*

omegalpes.energy.buildings.thermal.calc_Am (*Cm_Af*, *Af*)

omegalpes.energy.buildings.thermal.calc_Aop_bel (*height_bg*, *perimeter*, *footprint*)

omegalpes.energy.buildings.thermal.calc_Aop_sup (*Awall_all*, *void*, *win-*
dow_to_wall_ratio)

omegalpes.energy.buildings.thermal.calc_Hd (*Aop_sup*, *U_wall*, *footprint*, *U_roof*)

omegalpes.energy.buildings.thermal.calc_Hg (*Aop_bel*, *U_base*)

omegalpes.energy.buildings.thermal.calc_Htr_op (*Aop_bel*, *Aop_sup*, *footprint*, *U_base*,
U_wall, *U_roof*)

omegalpes.energy.buildings.thermal.calc_I_rad_linearization_coef (*Tdry*,
Tdew, *Tlin*,
sky_cover=1)

Parameters

- **T_dry** – Dry bulb temperature in Celsius
- **T_dew** – Dew point temperature in Celsius
- **sky_cover** –

Returns list(A), list(B):

omegalpes.energy.buildings.thermal.calc_I_sol (*I_sol_average*, *Aop_sup*, *Aroof*, *Awin*,
a_wall, *a_roof*, *U_wall*, *U_roof*,
Fsh_win)

Parameters

- **I_sol_average** – W/m2
- **Aop_sup** – Opaque wall areas above ground (excluding voids and windows) [m2]
- **Aroof** – Roof area [m2]
- **Awin** – Windows area [m2]
- **a_wall** – Absorption coefficient of the walls [0..1]

- **a_roof** – Absorption coefficient of the roof [0..1]
- **U_wall** –
- **U_roof** –
- **Fsh_win** – Shading factor for windows

Returns I_{sol} [kW]

`omegalpes.energy.buildings.thermal.calc_T_sky(T_dry, T_dew, sky_cover=1)`

Parameters

- **T_dry** – Dry bulb temperature in Celsius
- **T_dew** – Dew point temperature in Celsius
- **sky_cover** – Sky cover

`omegalpes.energy.buildings.thermal.calc_cm(Cm_Af, Af)`

`omegalpes.energy.buildings.thermal.calc_skytemp(Tdrybulb, Tdewpoint, N=1)`

Copyright 2014, Architecture and Building Systems - ETH Zurich

Parameters

- **Tdrybulb** – Drybulb temperature [°C]
- **Tdewpoint** – Dewpoint temperature [°C]
- **N** – Sky cover

Returns Sky temperature in °C

`omegalpes.energy.buildings.thermal.get_Cm_Af(construction)`

Description code Cm_Af Light construction T1 110000 Medium construction T2 165000 Heavy construction T3 300000

`omegalpes.energy.buildings.thermal.lookup_effective_mass_area_factor(cm)`

Look up the factor to multiply the conditioned floor area by to get the effective mass area by building construction type. This is used for the calculation of the effective mass area “Am” in *get_prop_RC_model*. Standard values can be found in the Annex G of ISO EN13790

param cm: The internal heat capacity per unit of area [J/m2].

return Effective mass area factor (0, 2.5 or 3.2 depending on cm value).

`omegalpes.energy.buildings.thermal.write_linerazation_exp(T_dry, T_dew, sky_cover, Tlin, U_win, U_wall, U_roof, e_win, e_wall, e_roof, A_win, A_wall, A_roof, name)`

Poles module

This module defines inputs and outputs of as poles

The poles module includes :

- FlowPole : this class defines a pole with a directed flow (in or out)
- EPole : this class define an energy pole

```
class omegalpes.energy.io.poles.Epole (p, direction, energy_type=None)  
    Bases: omegalpes.energy.io.poles.FlowPole
```

Description

Definition of an energetic pole, power and power flow direction convention 'in' or 'out'

```
class omegalpes.energy.io.poles.FlowPole (flow='flow', direction='in')  
    Bases: dict
```

Description

Interface for basics flux poles

The **actor classes** enables to build the energy model considering pre-defined stakeholders' constraints and objectives

2.2.2 actor package

The actor modelling is based on a main actor class defined on the actor module. Then, the actors are divided in two categories: the "operator actors" who operates energy units and the "regulator actors" who unable to create regulation constraints.

- *Actor module*
- *Operator_actors module*
- *Consumer_actors module*
- *Producer_actors module*
- *Consumer_producer_actors module*
- *Regulator_actors module*

Actor module

This modules define the basic Actor object

Few methods are available:

- `add_external_constraint`
- `add_external_dynamic_constraint`
- `add_objective`

```
class omegalpes.actor.actor.Actor (name, no_warn=True, verbose=True)  
    Bases: omegalpes.general.optimisation.core ОптимаObject
```

Description

Actor class is the basic class to model an actor. The basic actor is defined by its name and description.
An actor is then defined by its constraints and objectives.

Attributes

- `description` : description as an Actor ОптимаObject

```
add_external_constraint (cst_name, exp)  
    Enable to add an external constraint linked with an actor
```

Parameters

- **cst_name** – name of the constraint
- **exp** – expression of the constraint

add_external_dynamic_constraint (*cst_name, exp_t, t_range='for t in time.I'*)

Enable to add an external dynamic constraint linked with an actor. A dynamic constraint changes over time

Parameters

- **cst_name** – name of the constraint
- **exp** – expression of the constraint depending on the time
- **t_range** – expression of time for the constraint

add_objective (*obj_name, exp*)

Enable to add an objective linked with an actor

Parameters

- **obj_name** – name of the objective
- **exp** – expression of the objective

remove_external_constraints (*ext_cst_name_list=None*)

Enable to remove an external constraint linked with an actor

Parameters **ext_cst_name_list** – list of external constraint that would be removed

Operator_actors module

This module defines the **operator_actor** and its scope of responsibility

```
class omegalpes.actor.operator_actors.operator_actors.OperatorActor (name,
                                                                    oper-
                                                                    ated_unit_type_tuple,
                                                                    oper-
                                                                    ated_unit_list=None,
                                                                    oper-
                                                                    ated_node_list=None,
                                                                    ver-
                                                                    bose=True)
```

Bases: *omegalpes.actor.actor.Actor*

Description

OperatorActor class inherits from the the basic class Actor. It enables one to model an actor who operates energy units which is part of its scope of responsibility. An operator actor has objectives and constraints which are linked to the energy units he operates.

Attributes

- **name** : name of the actor
- **operated_unit_list**: list of the energy units operated by the actor or more precisely in its scope of responsibility

Consumer_actors module

This module describes the Consumer actor

Few objectives and constraints are available.

Objectives :

- `maximize_consumption`
- `minimize_consumption`
- `minimize_co2_consumption`
- `minimize_consumption_costs`

Constraints :

- `energy_consumption_minimum`
- `energy_consumption_maximum`
- `power_consumption_minimum`
- `power_consumption_maximum`

```
class omegalpes.actor.operator_actors.consumer_actors.Consumer(name,      oper-  
                                ated_unit_list,  
                                oper-  
                                ated_node_list=None,  
                                verbose=True)
```

Bases: `omegalpes.actor.operator_actors.operator_actors.OperatorActor`

Description

Consumer class inherits from the the class `OperatorActor`. It enables one to model a consumer actor.

energy_consumption_maximum (*max_e_tot, cst_operated_unit_list=None*)

To create the actor constraint of a maximum of energy consumption.

Parameters

- **max_e_tot** – Maximum of the total energy consumption over the study period
- **cst_operated_unit_list** – List of units on which the constraint will be applied.
Might be empty.

energy_consumption_minimum (*min_e_tot, cst_operated_unit_list=None*)

To create the actor constraint of a minimum of energy consumption.

Parameters

- **min_e_tot** – Minimum of the total energy consumption over the study period
- **cst_operated_unit_list** – List of units on which the constraint will be applied.
Might be empty.

maximize_consumption (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to maximize the consumption of the consumer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied.
Might be empty
- **weight** – Weight coefficient for the objective

minimize_co2_consumption (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to minimize the co2 emissions due to the consumer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied.
Might be empty
- **weight** – Weight coefficient for the objective

minimize_consumption (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to minimize the consumption of the consumer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied.
Might be empty
- **weight** – Weight coefficient for the objective

minimize_consumption_cost (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to minimize the expenses due to the consumer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied.
Might be empty
- **weight** – Weight coefficient for the objective

power_consumption_maximum (*max_p, time, cst_operated_unit_list=None*)

To create the actor constraint of a maximum of power consumption.

Parameters

- **max_p** – Maximum of the power consumption. May be an int, float or a list with the size of the period study
- **time** – period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied.
Might be empty.

power_consumption_minimum (*min_p, time, cst_operated_unit_list=None*)

To create the actor constraint of a minimum of power consumption.

Parameters

- **min_p** – Minimum of the power consumption. May be an int, float or a list with the size of the period study
- **time** – period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied.
Might be empty.

Producer_actors module

This module describes the Producer actor

Few objectives and constraints are available.

Objectives :

- maximize_production
- minimize_production
- minimize_time_of_use
- minimize_co2_emissions
- minimize_costs
- minimize_operating_cost
- minimize_starting_cost

Constraints :

- energy_production_minimum
- energy_production_maximum
- power_production_minimum
- power_production_maximum

```
class omegalpes.actor.operator_actors.producer_actors.Producer(name,      oper-
                                                                ated_unit_list,
                                                                oper-
                                                                ated_node_list=None,
                                                                verbose=True)
```

Bases: *omegalpes.actor.operator_actors.operator_actors.OperatorActor*

Description

Producer class inherits from the the class OperatorActor. It enables one to model an energy producer actor.

energy_production_maximum (*max_e_tot, cst_operated_unit_list=None*)

To create the actor constraint of a maximum of energy production.

Parameters

- **max_e_tot** – Maximum of the total energy production over the period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied.
Might be empty.

energy_production_minimum (*min_e_tot, cst_operated_unit_list=None*)

To create the actor constraint of a minimum of energy production.

Parameters

- **min_e_tot** – Minimum of the total energy production over the period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied.
Might be empty.

maximize_production (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to maximize the production of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied.
Might be empty.
- **weight** – Weight coefficient for the objective

minimize_co2_emissions (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to minimize the co2 emissions of the producer's units (all or part of them). based on the quantity "co2_emission"

Parameters

- **obj_operated_unit_list** – list of the operated energy units on which the objective will be applied
- **weight** – weight of the objective

minimize_costs (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to minimize the cost of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – list of the operated energy units on which the objective will be applied
- **weight** – weight of the objective

minimize_operating_cost (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to minimize the operating costs of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – list of the operated energy units on which the objective will be applied
- **weight** – weight of the objective

minimize_production (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to minimize the production of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

minimize_starting_cost (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to minimize the starting costs of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – list of the operated energy units on which the objective will be applied
- **weight** – weight of the objective

minimize_time_of_use (*obj_operated_unit_list=None, weight=1*)

To create the objective in order to minimize the time of use of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

power_production_maximum (*max_p, time, cst_operated_unit_list=None*)

To create the actor constraint of a maximum of power production.

Parameters

- **max_p** – Minimum of the power production. May be an int, float or a list with the size of the period study
- **time** – period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

power_production_minimum (*min_p, time, cst_operated_unit_list=None*)

To create the constraint of a minimum of power production.

Parameters

- **min_p** – Minimum of the power production. May be an int, float or a list with the size of the period study
- **time** – period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

Consumer_producer_actors module

This module describes the Prosumer (producer and consumer) actor

```
class omegalpes.actor.operator_actors.consumer_producer_actors.Prosumer (name,  
                                                                    op-  
                                                                    er-  
                                                                    ated_consumption_unit_list,  
                                                                    op-  
                                                                    er-  
                                                                    ated_production_unit_list,  
                                                                    op-  
                                                                    er-  
                                                                    ated_node_list=None,  
                                                                    ver-  
                                                                    bose=True)
```

Bases: *omegalpes.actor.operator_actors.consumer_actors.Consumer, omegalpes.actor.operator_actors.producer_actors.Producer*

Description

Prosumer class inherits from the the class OperatorActor, Consumer and Producer. It enables one to model an actor which is at the same time an energy producer and consumer

maximize_conso_prod_match (*time,* *obj_operated_consumption_unit_list=None,*
 obj_operated_production_unit_list=None, weight=1)

To create the objective in order to match at each time the consumption with the local production of the prosumer's units (all or part of them).

Parameters

- **obj_operated_consumption_unit_list** – List of consumption units on which the objective will be applied. Might be empty.
- **obj_operated_production_unit_list** – List of production units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective


```
maximize_selfconsumption_rate (time, obj_operated_production_unit_list=None,
                                obj_operated_consumption_unit_list=None,
                                obj_operated_selfconsummed_production_export_list=None,
                                obj_operated_selfconsummed_production_unit_list=None,
                                weight=1)
```

To create the objective in order to maximize the selfconsumption rate of the prosumer's units (all or part of them) WHILE maximizing the load matching selfconsummed production is calculated with the export nodes.

Selfconsumption rate = selfconsummed production / total production

Parameters

- **obj_operated_production_unit_list** – List of production units on which the objective will be applied. Might be empty.
- **obj_operated_selfconsummed_production_export_list** – List of production exports from nodes on which the objective will be applied. Might be empty.
- **obj_operated_selfconsummed_production_unit_list** – List of production units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

```
maximize_selfproduction_rate (time, obj_operated_consumption_unit_list=None,
                                obj_operated_production_unit_list=None,
                                obj_operated_selfproduced_consumption_export_list=None,
                                obj_operated_selfproduced_consumption_unit_list=None,
                                weight=1)
```

To create the objective in order to maximize the selfproduction rate of the prosumer's units (all or part of them) WHILE maximizing the load matching selfproduced consumption may required export nodes

Selfproduction rate = selfproduced consumption / total consumption

Parameters

- **obj_operated_consumption_unit_list** – List of consumption units on which the objective will be applied. Might be empty.
- **obj_operated_selfproduced_consumption_export_list** – List of production exports from nodes on which the objective will be applied. Might be empty.
- **obj_operated_selfproduced_consumption_unit_list** – List of production units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

```
class omegalpes.actor.operator_actors.consumer_producer_actors.Supplier (name,
                                                                    op-
                                                                    er-
                                                                    ated_consumption_unit_list,
                                                                    op-
                                                                    er-
                                                                    ated_production_unit_list,
                                                                    ver-
                                                                    bose=True)
```

Bases: `omegalpes.actor.operator_actors.consumer_actors.Consumer`, `omegalpes.actor.operator_actors.producer_actors.Producer`

Description

Supplier class inherits from the the class OperatorActor, Consumer and Producer. It enables one to model a supplier.

Regulator_actors module

This module defines the `operator_actor` and its scope of responsibility

One constraint is available :

- `co2_emission_maximum`

class `omegalpes.actor.regulator_actors.regulator_actors.LocalAuthorities` (*name*)
Bases: `omegalpes.actor.regulator_actors.regulator_actors.RegulatorActor`

Description

`LocalAuthorities` class inherits from the basic class `RegulatorActor`. It focuses on local Authorities constraints

class `omegalpes.actor.regulator_actors.regulator_actors.PublicAuthorities` (*name*)
Bases: `omegalpes.actor.regulator_actors.regulator_actors.RegulatorActor`

Description

`PublicAuthorities` class inherits from the basic class `RegulatorActor`. It focuses on local Authorities constraints

class `omegalpes.actor.regulator_actors.regulator_actors.RegulatorActor` (*name*,
ver-
bose=True)
Bases: `omegalpes.actor.actor.Actor`

Description

`RegulatorActor` class inherits from the the basic class `Actor`. It enables one to model an actor who can add constraints on all energy units of the study case.

Attributes

- `name` : name of the actor

`co2_emission_maximum` (*max_co2*, *time*, *cst_production_list*)
To create the actor constraint of a maximum of CO2 emission.

Parameters

- **`max_co2`** – Minimum of the CO2 emission. May be an int, float or a list with the size of the period study
- **`time`** – period of the study
- **`cst_unit_list`** – List of units on which the constraint will be applied.

The **general classes** helps to build the units, the model and to plot the results

2.2.3 general package

Please, have a look to the following general modules

- *Elements module*
- *Core module*
- *Time module*

- *Model module*
- *Input Data module*
- *Output Data module*
- *Plots module*
- *Maths module*

Firstly, the main modules to build an energy optimisation model are presented :

Elements module

This module includes the optimization elements (quantities, constraints and objectives) formulated in LP or MILP

- Quantity : related to the decision variable or parameter
- Constraint : related to the optimization problem constraints
- Objective : related to the objective function

```
class omegalpes.general.optimisation.elements.Constraint (exp,          name='CST0',
                                                         description="",          ac-
                                                         tive=True,              par-
                                                         ent=None)
```

Bases: object

Description

Class that defines a constraint object

Attributes

- name: name of the constraint
- description: a description of the constraint
- active: False = non-active constraint; True = active constraint
- exp: (str) : expression of the constraint
- parent: (unit) : this constraint belongs to this unit

Note: Make sure that all the modifications on Constraints are made before adding the unit to the Model (OptimisationModel.addUnit()).

```
class omegalpes.general.optimisation.elements.DynamicConstraint (exp_t,
                                                                    t_range='for
                                                                    t in time.I',
                                                                    name='DCST0',
                                                                    descrip-
                                                                    tion='dynamic
                                                                    constraint',
                                                                    active=True,
                                                                    parent=None)
```

Bases: *omegalpes.general.optimisation.elements.Constraint*

Description

Defining a constraint depending on the time. NB : Mandatory for PuLP

```
class omegalpes.general.optimisation.elements.ExtDynConstraint (exp_t,  
                                                             t_range='for  
                                                             t in time.I',  
                                                             name='EDCST0',  
                                                             active=True,  
                                                             description='Non-  
                                                             physical and  
                                                             dynamic con-  
                                                             straint', par-  
                                                             ent=None)  
  
Bases: omegalpes.general.optimisation.elements.DynamicConstraint, omegalpes.  
general.optimisation.elements.ExternalConstraint
```

Description

Defining a constraint both external and dynamic (see: `DynamicConstraint`, `ExternalConstraint`)

```
class omegalpes.general.optimisation.elements.ExternalConstraint (exp,  
                                                             name='ExCST0',  
                                                             descrip-  
                                                             tion='',  
                                                             active=True,  
                                                             par-  
                                                             ent=None)  
  
Bases: omegalpes.general.optimisation.elements.Constraint
```

Description

Defining a special type of constraint: the external constraint

- This constraint does not translate a physical constraint
- This constraint defines an external constraint, which could be relaxed

`deactivate_constraint()`

An external constraint can be deactivated : - To compare scenarios - To try a less constrained problem

```
class omegalpes.general.optimisation.elements.HourlyDynamicConstraint (exp_t,  
                                                             time,  
                                                             init_h:  
                                                             int =  
                                                             0, fi-  
                                                             nal_h:  
                                                             int =  
                                                             24,  
                                                             name='HDCST0',  
                                                             de-  
                                                             scrip-  
                                                             tion='hourly  
                                                             dy-  
                                                             nam-  
                                                             ic con-  
                                                             straint',  
                                                             ac-  
                                                             tive=True,  
                                                             par-  
                                                             ent=None)  
  
Bases: omegalpes.general.optimisation.elements.DynamicConstraint
```

Description

Class that defines an dynamic constraint for a time range

Ex: Constraint applying between 7am and 10pm

```
ex_cst = HourlyDynamicConstraint(exp_t, time, init_h=7, final_h=22, name='ex_cst')
```

Attributes

- name (str) : name of the constraint
- exp_t (str) : expression of the constraint
- init_h (int) : hour of beginning of the constraint [0-23]
- final_h (int) : hour of end of the constraint [1-24]
- description (str) : description of the constraint
- active (bool) : defines if the constraint is active or not
- parent (OptObject) : parent of the constraint

```
class omegalpes.general.optimisation.elements.Objective(exp, name='OBJ0', description="", active=True, weight=1, unit='s.u.', parent=None)
```

Bases: object

Description

Class that defines an optimisation objective

Attributes

- name (str) :
- description (str) :
- active (bool) :
- exp (str) :
- weight (float) : weighted factor of the objective
- parent (unit)
- unit (str) : unit of the cost expression

Note: Make sure that all the modifications on Objectives are made before adding the unit to the Optimisation-Model, otherwise, it won't be taken into account

```
class omegalpes.general.optimisation.elements.Quantity(name='var0', opt=True, unit='s.u', vlen=None, value=None, description="", vtype='Continuous', lb=None, ub=None, parent=None)
```

Bases: object

Description

Class that defines what is a quantity. A quantity can wether be a decision variable or a parameter, depending on the opt parameter

Attributes

- name (str) : the name of the quantity
- description (str) : a description of the meaning of the quantity
- vtype (PuLP) : the variable type, depending on PuLP
 - LpBinary (binary variable)
 - LpInteger (integer variable)
 - LpContinuous (continuous variable)
- vlen (int) : size of the variable
- unit (str) : unit of the quantity
- opt (binary)
 - True: this is an optimization variable
 - False: this is a constant - a parameter
- value (float, list, dict) : value (unused if opt=True)
- ub, lb : upper and lower bounds
- parent (OptObject) : the quantity belongs to this unit

Note: Make sure that all the modifications on Quantity are made before adding the unit to the Model

Core module

This module define the main Core object on which the energy units and actors will be based

```
class omegalpes.general.optimisation.core.OptObject (name='U0',          descrip-  
                                                    tion='Optimization    object',  
                                                    verbose=True)
```

Bases: object

Description

OptObject class is used as an “abstract class”, i.e. it defines some general attributes and methods but doesn’t contain variable, constraint nor objective. In the OMEGAAlpes package, all the subsystem models are represented by a unit. A model is then generated adding OptObject to it. Variable, objective and constraints declarations are usually done using the `__init__` method of the OptObject class.

Attributes

- name
- description
- _quantities_list : list of the quantities of the OptObject (active or not)
- _constraints_list : list of the constraints of the OptObject (active or not)
- _external_constraints_list : list of the constraints of the OptObject (active or not)
- _objectives_list : list of the objectives of the OptObject (active or not)

Methods

- `__str__`: defines the
- `__repr__`: defines the unit with its name
- `get_constraints_list`
- `get_constraints_name_list`
- `get_external_constraints_list`
- `get_external_constraints_name_list`
- `get_objectives_list`
- `get_objectives_name_list`
- `get_quantities_list`
- `get_quantities_name_list`
- `deactivate_optobject_external_constraints`

Note: The `OptObject` class shouldn't be instantiated in a python script, except if you want to create your own model from the beginning. In this case, one should consider creating a new class `NewModel(OptObject)`.

`deactivate_optobject_external_constraints` (*ext_cst_name_list=None*)

Enable to remove an external constraint linked with an `OptObject`

Parameters `ext_cst_name_list` – list of external constraint that would be removed

`get_constraints_list` ()

Get the constraints associated with the unit as a dictionary shape [`'constraint_name'` , constraint]

`get_constraints_name_list` ()

Get the names of the constraints associated with the unit

`get_external_constraints_list` ()

Get the external constraints associated with the unit as a dictionary shape [`'constraint_name'` , constraint]

`get_external_constraints_name_list` ()

Get the names of the external constraints associated with the unit

`get_objectives_list` ()

Get objectives associated with the unit as a dictionary shape [`'objective_name'` , objective]

`get_objectives_name_list` ()

Get the names of the objectives associated with the unit

`get_quantities_list` ()

Get the quantities associated with the unit as a dictionary shape [`'quantity_name'` , quantity]

`get_quantities_name_list` ()

Get the names of the quantities associated with the unit

Time module

This module creates the `Time` object

class `omegalpes.general.time.TimeUnit` (*start='01/01/2018', end=None, periods=None, dt=1*)

Bases: `object`

Description

Class defining the studied time period.

Attributes

- **DATES** : dated list of simulation steps
- **DT** : delta t between values in hours (int or float), i.e. 1/6 will be 10 minutes.
- **LEN** : number of simulation steps (length of DATES)
- **I** : index of time ([0 : LEN])

get_date_for_index (*index*)

Getting a date for a given index

Parameters **index** – int value for the index of the wanted dated, between 0 and LEN (it must be in the studied period)

get_days

Getting days for the studied period

Return **all_days** list of days of the studied period

get_index_for_date (*date*=*'YYYY-MM-DD HH:MM:SS'*)

Getting the index associated with a date

Parameters **date** – date the index of is wanted. Format YYYY-MM-DD HH:MM:SS, must be within the studied period and consistent with the timestep value

get_index_for_date_range (*starting_date*=*'YYYY-MM-DD HH:MM:SS'*, *end*=*None*, *periods*=*None*)

Getting a list of index for a date range

Parameters

- **starting_date** – starting date of the wanted index
- **end** – ending date of the wanted index
- **periods** – number of periods from the starting_date of the wanted index

Return **index_list** list of indexes for the given dates

get_non_working_dates (*month_range*=*range(0, 12)*, *hour_range*=*range(0, 24)*, *country*=*'France'*)

get_non_working_days (*country*=*'France'*)

get_working_dates (*month_range*=*range(0, 12)*, *hour_range*=*range(0, 24)*, *country*=*'France'*)

get_working_days (*country*=*'France'*)

print_studied_period ()

omegalpes.general.time.convert_european_format (*date*)

Converting a date with an european format DD/MM/YYYY into a datetime format YYYY-MM-DD or return

Parameters **date** – date in european format

Returns date in format datetime

Model module

This module enables to fill the optimization model and formulate it in LP or MILP based on the package PuLP (LpProblem)

```
class omegalpes.general.optimisation.model.OptimisationModel (time,  
                                                             name='optimisation_model')
```

Bases: `pulp.pulp.LpProblem`

Description

This class includes the optimization model formulated in LP or MILP based on the package PuLP (LpProblem)

add_nodes (**nodes*)

Add nodes and all connected units to the model Check that the time is the same for the model and all the units

Parameters *nodes* – EnergyNode

add_nodes_and_actors (**nodes_or_actors*)

Add nodes, actors and all connected units to the model Check that the time is the same for the model and all the units

Parameters *nodes_or_actors* – EnergyNode or Actor type

get_model_constraints_list ()

Gets constraints of the model

get_model_constraints_name_list ()

Gets the names of the constraints of the model

get_model_objectives_list ()

Gets objectives of the model

get_model_objectives_name_list ()

Gets the names of the objectives of the model

get_model_quantities_list ()

Gets quantities of the model

get_model_quantities_name_list ()

Gets the names of the quantities of the model

solve_and_update (*solver: pulp.solvers.LpSolver = None*) → None

Solves the optimization model and updates all variables values.

Parameters *solver* (*LpSolver*) – Optimization solver

update_units ()

Updates all units values with optimization results

`omegalpes.general.optimisation.model.check_if_unit_could_have_parent` (*unit*)

Checks if the unit has an associated parent

Parameters *unit* – unit which parents will be checked

```
omegalpes.general.optimisation.model.compute_gurobi_IIS (gurobi_exe_path='C:\gurobi800\win64\bin',  
                                                         opt_model=None,  
                                                         MPS_model=None)
```

Identifies the constraints in a .ilp file

Parameters

- **gurobi_exe_path** – Path to the gurobi solver “gurobi_cl.exe”

- **opt_model** – OptimisationModel to whom compute IIS
- **MPS_model** – name of the mps model

Then, utils methods are developed and are presented in the following module:

Input Data module

This module includes the following utils for input data management

It contains the following methods:

- **select_csv_file_between_dates()** : select data in a .csv file between two dates
- **read_enedis_data_csv_file()** : select and rearrange the data in a .csv file of Enedis (the French Distribution System Operator company), possibly between two dates

```
omegalpes.general.utils.input_data.read_enedis_data_csv_file(file_path=None,  
                                                             start=None,  
                                                             end=None)
```

Rearrange the Enedis data in cvs file in order to have a Dataframe of the following form

DD MM YYYY HH:00 ; a DD MM YYYY HH:30 ; b ...

Parameters

- **file_path** – path of the file to rearrange
- **start** – DD MM YYYY HH:00 : first date which should be considered
- **end** – DD MM YYYY HH:00 : last date which should be considered

Returns df_list: the data as a list

```
omegalpes.general.utils.input_data.select_csv_file_between_dates(file_path=None,  
                                                                  start='DD/MM/YYYY  
HH:MM',  
                                                                  end='DD/MM/YYYY  
HH:MM',  
                                                                  v_cols=[],  
                                                                  sep=';')
```

Select data in a .csv file between two dates

Parameters

- **file_path** – path of the file to rearrange
- **start** – DD MM YYYY HH:00 : first date which should be considered
- **end** – DD MM YYYY HH:00 : last date which should be considered
- **v_cols** – columns which should be considered

Returns df: a dataframe considering the dates

Output Data module

This module includes the following utils for output data management

It contains the following methods:

- **save_energy_flows()** : Save the optimisation results in a .csv file

```
omegalpes.general.utils.output_data.save_energy_flows(*nodes, file_name=None,
                                                    sep='\\', decimal_sep='.')
```

Save the optimisation results in a .csv file

Parameters

- **nodes** – list of the nodes from which should be collected the data
- **file_name** – name of the file to save the data
- **sep** – separator for the data
- **decimal_sep** – separator for the decimals of the data

Plots module

This module includes the following display utils:

- `plot_node_energetic_flows()` : enables one to plot the energy flows through an EnergyNode
- `plot_energy_mix()` : enables one to plot the energy flows connected to a node
- `plot_quantity()` : enables one to plot easily a Quantity
- `plot_quantity_bar()` : enables one to plot easily a Quantity as a bar
- `sum_quantities_in_quantity()` : enables one to plot several quantities in one once the optimisation is done

```
omegalpes.general.utils.plots.plot_energy_mix(node)
```

```
omegalpes.general.utils.plots.plot_node_energetic_flows(node)
```

Description

This function allows to plot the energy flows through an EnergyNode

The display is realized :

- with histograms for production and storage flow
- with dashed curve for consumption flow

Parameters **node** – EnergyNode

```
omegalpes.general.utils.plots.plot_quantity(time, quantity, fig=None, ax=None,
                                           color=None, label=None)
```

Description

Function that plots a OMEGALPES.general.optimisation.elements.Quantity

Attributes

- **quantity** is the OMEGALPES.general.optimisation.elements.Quantity
- **fig** could be None, a matplotlib.pyplot.Figure or Axes for multiple plots

Returns

- **arg1** the matplotlib.pyplot.Figure handle object
- **arg2** the matplotlib.pyplot.Axes handle object
- **arg3** the matplotlib.pyplot.Line2D handle object

```
omegalpes.general.utils.plots.plot_quantity_bar(time, quantity, fig=None, ax=None,
                                                color=None, label=None)
```

Description

Function that plots a `OMEGALPES.general.optimisation.elements.Quantity` as a bar

Attributes

- `quantity` is the `OMEGALPES.general.optimisation.elements.Quantity`
- `fig` could be `None`, a `matplotlib.pyplot.Figure` or `Axes` for multiple plots

Returns

- `arg1` the `matplotlib.pyplot.Figure` handle object
- `arg2` the `matplotlib.pyplot.Axes` handle object
- `arg3` the `matplotlib.pyplot.Line2D` handle object

```
omegalpes.general.utils.plots.sum_quantities_in_quantity (quantities_list=[],  
                                                         tot_quantity_name='sum_quantity')
```

Description

Function that creates a new quantity gathering several values of quantities Should be used in order to plot several quantities in one once the optimisation is done

Attributes

- `quantities` : a list of `Quantities` (`OMEGALPES.general.optimisation.elements.Quantity`)
- `tot_quantity_name` : string : name of the new quantity

Returns

- `tot_quantity` : the new quantity created and filled

Maths module

This module includes the following maths utils

It contains the following methods:

- `def_abs_value()` : Define easily absolute value for quantity

```
omegalpes.general.utils.maths.def_abs_value (quantity, q_min, q_max)
```

Parameters

- **quantity** – Quantity whose absolute value is wanted
- **q_min** – Minimal value of the quantity (negative value)
- **q_max** – Maximal value of the quantity (positive value)

Returns A new `Quantity` whose values equal absolute values of the initial `Quantity`

2.3 OMEGALpes Graphical Representation

The energy systems developed in OMEGALpes are describes following a specified graph representation.

| Unspecified | Fixed power (input of optimisation) | Variable power (output of optimisation) |
|-------------|--|--|
| Production | Fixed Production | Variable Production |
| Consumption | Fixed Consumption | Variable Consumption |

Fig. 2: Figure: Energy unit representation

| Gas | Heat | Electricity |
|-------------|-------------|-------------|
| Production | Production | Production |
| Consumption | Consumption | Consumption |
| Storage | Storage | Storage |

Fig. 3: Figure: Specified carrier energy unit representation

2.3.1 Energy Unit representation

The energy units are describes as rectangles as presented just below. The graph is adapted considering if the unit is a variable or a fixed energy unit.

A colour may be added to specify the carrier of the energy unit like gas (yellow), thermal (red), electricity (blue).

Conversion units use the former representation inside a green box as multi-carrier energy units. Have a look to the representation for ElectricalToHeat and HeatPump energy units

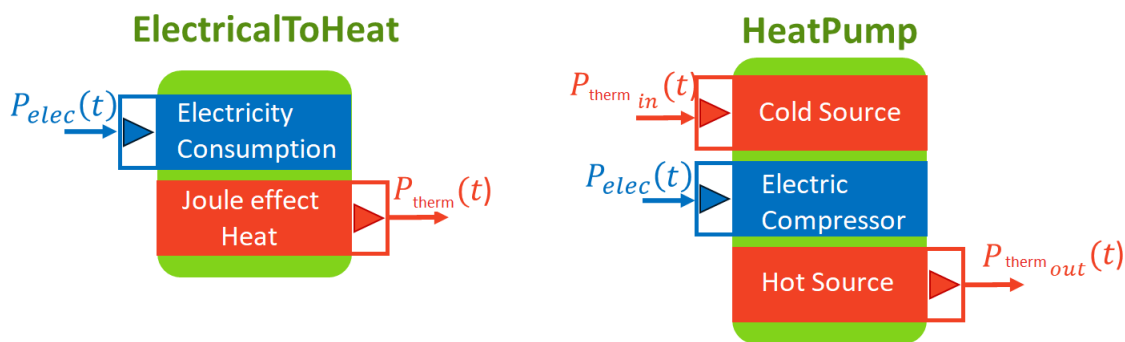


Fig. 4: Figure: Conversion units representation

2.3.2 Representing all the energy unit

In order to link the energy units, energy nodes and arrows should be used. It is possible to highlight the variable which may be optimised by the system or which is interesting for the user.










| | Gas | Heat | Electricity |
|----------------------|---|---|---|
| Power flow |  |  |  |
| Energy Node |  |  |  |
| Variable of interest |  |  |  |

Fig. 5: Figure: Energy Nodes, Flows and Variable of interest representation

Finally, constraints and objectives can be added on the model with the following representation



Fig. 6: Figure: Objectives and Constraints representation

Please, have a look to the examples to see if applications of this graphical representation [OMEGAAlpes Examples Documentation](#)

Note: This graph representation is not used yet as a graphical user interface but we hope that it will be in the near future.

2.4 OMEGAAlpes Examples

Please click on the following link to have a look to OMEGAAlpes examples and study cases: [OMEGAAlpes Examples Documentation](#)

CHAPTER 3

Acknowledgments

Vincent Reinbold - Library For Linear Modeling of Energetic Systems : <https://github.com/ReinboldV>

Mathieu Brugeron

This work has been partially supported by the [CDP Eco-SESA](#) receiving fund from the French National Research Agency in the framework of the “Investissements d’avenir” program (ANR-15-IDEX-02) and the VALocal project (CNRS Interdisciplinary Mission and INSIS)

O

`omegalpes.actor.actor`, [30](#)
`omegalpes.actor.operator_actors.consumer_actors`,
 [32](#)
`omegalpes.actor.operator_actors.consumer_producer_actors`,
 [36](#)
`omegalpes.actor.operator_actors.operator_actors`,
 [31](#)
`omegalpes.actor.operator_actors.producer_actors`,
 [33](#)
`omegalpes.actor.regulator_actors.regulator_actors`,
 [38](#)
`omegalpes.energy.buildings.thermal`, [27](#)
`omegalpes.energy.energy_nodes`, [26](#)
`omegalpes.energy.io.poles`, [29](#)
`omegalpes.energy.units.consumption_units`,
 [11](#)
`omegalpes.energy.units.conversion_units`,
 [22](#)
`omegalpes.energy.units.energy_units`, [7](#)
`omegalpes.energy.units.production_units`,
 [16](#)
`omegalpes.energy.units.storage_units`,
 [24](#)
`omegalpes.general.optimisation.core`, [42](#)
`omegalpes.general.optimisation.elements`,
 [39](#)
`omegalpes.general.optimisation.model`,
 [45](#)
`omegalpes.general.time`, [43](#)
`omegalpes.general.utils.input_data`, [46](#)
`omegalpes.general.utils.maths`, [48](#)
`omegalpes.general.utils.output_data`, [46](#)
`omegalpes.general.utils.plots`, [47](#)

A

Actor (class in *omegalpes.actor.actor*), 30

add_availability()
(*omegalpes.energy.units.energy_units.EnergyUnit*
method), 8

add_co2_emissions()
(*omegalpes.energy.units.energy_units.EnergyUnit*
method), 8

add_connected_energy_unit()
(*omegalpes.energy.energy_nodes.EnergyNode*
method), 26

add_external_constraint()
(*omegalpes.actor.actor.Actor* method), 30

add_external_dynamic_constraint()
(*omegalpes.actor.actor.Actor* method), 31

add_max_ramp_down()
(*omegalpes.energy.units.energy_units.EnergyUnit*
method), 8

add_max_ramp_up()
(*omegalpes.energy.units.energy_units.EnergyUnit*
method), 8

add_min_time_off()
(*omegalpes.energy.units.energy_units.EnergyUnit*
method), 8

add_min_time_on()
(*omegalpes.energy.units.energy_units.EnergyUnit*
method), 8

add_nodes() (*omegalpes.general.optimisation.model.OptimisationModel*
method), 45

add_nodes_and_actors()
(*omegalpes.general.optimisation.model.OptimisationModel*
method), 45

add_objective() (*omegalpes.actor.actor.Actor*
method), 31

add_operating_cost()
(*omegalpes.energy.units.energy_units.EnergyUnit*
method), 8

add_pole() (*omegalpes.energy.energy_nodes.EnergyNode*
method), 26

add_starting_cost()
(*omegalpes.energy.units.energy_units.EnergyUnit*
method), 8

C

calc_Am() (in module
omegalpes.energy.buildings.thermal), 28

calc_Aop_bel() (in module
omegalpes.energy.buildings.thermal), 28

calc_Aop_sup() (in module
omegalpes.energy.buildings.thermal), 28

calc_cm() (in module
omegalpes.energy.buildings.thermal), 29

calc_Hd() (in module
omegalpes.energy.buildings.thermal), 28

calc_Hg() (in module
omegalpes.energy.buildings.thermal), 28

calc_Htr_op() (in module
omegalpes.energy.buildings.thermal), 28

calc_I_rad_linearization_coef() (in module
omegalpes.energy.buildings.thermal), 28

calc_I_sol() (in module
omegalpes.energy.buildings.thermal), 28

calc_skytemp() (in module
omegalpes.energy.buildings.thermal), 29

calc_T_sky() (in module
omegalpes.energy.buildings.thermal), 29

check_if_unit_could_have_parent() (in
module *omegalpes.general.optimisation.model*),
45

co2_emission_maximum()
(*omegalpes.actor.regulator_actors.regulator_actors.RegulatorAct*
method), 38

compute_gurobi_IIS() (in module
omegalpes.general.optimisation.model), 45

connect_units() (*omegalpes.energy.energy_nodes.EnergyNode*
method), 26

Constraint (class in
omegalpes.general.optimisation.elements),
39

Consumer (class in *omegalpes.actor.operator_actors.consumer_actors*),
32

ConsumptionUnit (class in *omegalpes.energy.units.consumption_units*), 12

ConversionUnit (class in *omegalpes.energy.units.conversion_units*),
22

convert_european_format() (in module *omegalpes.general.time*), 44

create_export() (*omegalpes.energy.energy_nodes.EnergyNode*
method), 26

D

deactivate_constraint() (*omegalpes.general.optimisation.elements.ExternalConstraint*
method), 40

deactivate_optobject_external_constraints() (*omegalpes.general.optimisation.core.OptObject*
method), 43

def_abs_value() (in module *omegalpes.general.utils.maths*), 48

DynamicConstraint (class in *omegalpes.general.optimisation.elements*),
39

E

ElectricalToThermalConversionUnit (class in *omegalpes.energy.units.conversion_units*),
23

energy_consumption_maximum() (*omegalpes.actor.operator_actors.consumer_actors.Consumer*
method), 32

energy_consumption_minimum() (*omegalpes.actor.operator_actors.consumer_actors.Consumer*
method), 32

energy_production_maximum() (*omegalpes.actor.operator_actors.producer_actors.Producer*
method), 34

energy_production_minimum() (*omegalpes.actor.operator_actors.producer_actors.Producer*
method), 34

EnergyNode (class in *omegalpes.energy.energy_nodes*), 26

EnergyUnit (class in *omegalpes.energy.units.energy_units*), 8

Epole (class in *omegalpes.energy.io.poles*), 29

export_to_node() (*omegalpes.energy.energy_nodes.EnergyNode*
method), 26

ExtDynConstraint (class in *omegalpes.general.optimisation.elements*),
40

ExternalConstraint (class in *omegalpes.general.optimisation.elements*),
40

FixedConsumptionUnit (class in *omegalpes.energy.units.consumption_units*), 13

FixedEnergyUnit (class in *omegalpes.energy.units.energy_units*), 9

FixedProductionUnit (class in *omegalpes.energy.units.production_units*),
17

FlowPole (class in *omegalpes.energy.io.poles*), 30

G

get_Cm_Af() (in module *omegalpes.energy.buildings.thermal*), 29

get_connected_energy_units(*omegalpes.energy.energy_nodes.EnergyNode*
attribute), 26

get_constraints_list() (*omegalpes.general.optimisation.core.OptObject*
method), 43

get_constraints_name_list() (*omegalpes.general.optimisation.core.OptObject*
method), 43

get_date_for_index() (*omegalpes.general.time.TimeUnit* method), 44

get_days(*omegalpes.general.time.TimeUnit* attribute),
44

get_exports(*omegalpes.energy.energy_nodes.EnergyNode*
attribute), 26

get_external_constraints_list() (*omegalpes.general.optimisation.core.OptObject*
method), 43

get_external_constraints_name_list() (*omegalpes.general.optimisation.core.OptObject*
method), 43

get_flows(*omegalpes.energy.energy_nodes.EnergyNode*
attribute), 26

get_imports(*omegalpes.energy.energy_nodes.EnergyNode*
attribute), 26

get_index_for_date() (*omegalpes.general.time.TimeUnit* method), 44

get_index_for_date_range() (*omegalpes.general.time.TimeUnit* method), 44

get_model_constraints_list() (*omegalpes.general.optimisation.model.OptimisationModel*
method), 45

get_model_constraints_name_list() (*omegalpes.general.optimisation.model.OptimisationModel*
method), 45

get_model_objectives_list() (*omegalpes.general.optimisation.model.OptimisationModel*
method), 45

get_model_objectives_name_list() (*omegalpes.general.optimisation.model.OptimisationModel*
method), 45

get_model_quantities_list() (omegalpes.general.optimisation.model.OptimisationModel method), 45

get_model_quantities_name_list() (omegalpes.general.optimisation.model.OptimisationModel method), 45

get_non_working_dates() (omegalpes.general.time.TimeUnit method), 44

get_non_working_days() (omegalpes.general.time.TimeUnit method), 44

get_objectives_list() (omegalpes.general.optimisation.core.OptObject method), 43

get_objectives_name_list() (omegalpes.general.optimisation.core.OptObject method), 43

get_poles (omegalpes.energy.energy_nodes.EnergyNode attribute), 26

get_quantities_list() (omegalpes.general.optimisation.core.OptObject method), 43

get_quantities_name_list() (omegalpes.general.optimisation.core.OptObject method), 43

get_working_dates() (omegalpes.general.time.TimeUnit method), 44

get_working_days() (omegalpes.general.time.TimeUnit method), 44

H

HeatingLoad (class in omegalpes.energy.buildings.thermal), 27

HeatPump (class in omegalpes.energy.units.conversion_units), 23

HourlyDynamicConstraint (class in omegalpes.general.optimisation.elements), 40

I

import_from_node() (omegalpes.energy.energy_nodes.EnergyNode method), 26

is_export_flow() (omegalpes.energy.energy_nodes.EnergyNode method), 27

is_import_flow() (omegalpes.energy.energy_nodes.EnergyNode method), 27

L

LocalAuthorities (class in omegalpes.actor.regulator_actors.regulator_actors), 38

lookup_effective_mass_area_factor() (in module omegalpes.energy.buildings.thermal), 29

M

maximize_conso_prod_match() (omegalpes.actor.operator_actors.consumer_producer_actors.Producer method), 36

maximize_consumption() (omegalpes.actor.operator_actors.consumer_actors.Consumer method), 32

maximize_consumption() (omegalpes.energy.units.consumption_units.ConsumptionUnit method), 12

maximize_production() (omegalpes.actor.operator_actors.producer_actors.Producer method), 34

maximize_production() (omegalpes.energy.units.production_units.ProductionUnit method), 18

maximize_selfconsumption_rate() (omegalpes.actor.operator_actors.consumer_producer_actors.Producer method), 36

maximize_selfproduction_rate() (omegalpes.actor.operator_actors.consumer_producer_actors.Producer method), 37

maximize_thermal_comfort() (omegalpes.energy.buildings.thermal.HeatingLoad method), 27

minimize_capacity() (omegalpes.energy.units.storage_units.StorageUnit method), 25

minimize_co2_consumption() (omegalpes.actor.operator_actors.consumer_actors.Consumer method), 32

minimize_co2_emissions() (omegalpes.actor.operator_actors.producer_actors.Producer method), 34

minimize_co2_emissions() (omegalpes.energy.units.energy_units.EnergyUnit method), 8

minimize_consumption() (omegalpes.actor.operator_actors.consumer_actors.Consumer method), 33

minimize_consumption() (omegalpes.energy.units.consumption_units.ConsumptionUnit method), 12

minimize_consumption_cost() (omegalpes.actor.operator_actors.consumer_actors.Consumer method), 33

minimize_consumption_cost() (omegalpes.energy.units.consumption_units.ConsumptionUnit method), 13

minimize_costs() (omegalpes.actor.operator_actors.producer_actors.Producer method), 35

minimize_costs() (omegalpes.energy.units.energy_units.EnergyUnit method), 8

minimize_energy() (omegalpes.energy.units.energy_units.EnergyUnit method), 8

`(omegalpes.energy.units.energy_units.EnergyUnit`
`method), 8`
`minimize_operating_cost()`
`(omegalpes.actor.operator_actors.producer_actors.Producer`
`method), 35`
`minimize_operating_cost()`
`(omegalpes.energy.units.energy_units.EnergyUnit`
`method), 8`
`minimize_production()`
`(omegalpes.actor.operator_actors.producer_actors.Producer`
`method), 35`
`minimize_production()`
`(omegalpes.energy.units.production_units.ProductionUnit`
`method), 18`
`minimize_starting_cost()`
`(omegalpes.actor.operator_actors.producer_actors.Producer`
`method), 35`
`minimize_starting_cost()`
`(omegalpes.energy.units.energy_units.EnergyUnit`
`method), 9`
`minimize_time_of_use()`
`(omegalpes.actor.operator_actors.producer_actors.Producer`
`method), 35`
`minimize_time_of_use()`
`(omegalpes.energy.units.energy_units.EnergyUnit`
`method), 9`
O
`Objective` (class in `omegalpes.general.optimisation.elements`),
41
`omegalpes.actor.actor` (module), 30
`omegalpes.actor.operator_actors.consumer_actors`
`(module), 32`
`omegalpes.actor.operator_actors.consumer_producer_actors`
`(module), 36`
`omegalpes.actor.operator_actors.operator_actors`
`(module), 31`
`omegalpes.actor.operator_actors.producer_actors`
`(module), 33`
`omegalpes.actor.regulator_actors.regulator_actors`
`(module), 38`
`omegalpes.energy.buildings.thermal` (mod-
`ule), 27`
`omegalpes.energy.energy_nodes` (module), 26
`omegalpes.energy.io.poles` (module), 29
`omegalpes.energy.units.consumption_units`
`(module), 11`
`omegalpes.energy.units.conversion_units`
`(module), 22`
`omegalpes.energy.units.energy_units`
`(module), 7`
`omegalpes.energy.units.production_units`
`(module), 16`
`omegalpes.energy.units.storage_units`
`(module), 24`
`omegalpes.general.optimisation.core`
`(module), 42`
`omegalpes.general.optimisation.elements`
`(module), 39`
`omegalpes.general.optimisation.model`
`(module), 45`
`omegalpes.general.time` (module), 43
`omegalpes.general.utils.input_data` (mod-
`ule), 46`
`omegalpes.general.utils.maths` (module), 48
`omegalpes.general.utils.output_data`
`(module), 46`
`omegalpes.general.utils.plots` (module), 47
`OptActor` (class in
`omegalpes.actor.operator_actors.operator_actors`),
31
`OptimisationModel` (class in
`omegalpes.general.optimisation.model`), 45
`OptObject` (class in
`omegalpes.general.optimisation.core`), 42
P
`plot_energy_mix()` (in module
`omegalpes.general.utils.plots`), 47
`plot_node_energetic_flows()` (in module
`omegalpes.general.utils.plots`), 47
`plot_quantity()` (in module
`omegalpes.general.utils.plots`), 47
`plot_quantity_bar()` (in module
`omegalpes.general.utils.plots`), 47
`power_consumption_maximum()`
`(omegalpes.actor.operator_actors.consumer_actors.Consumer`
`method), 33`
`power_consumption_minimum()`
`(omegalpes.actor.operator_actors.consumer_actors.Consumer`
`method), 33`
`power_production_maximum()`
`(omegalpes.actor.operator_actors.producer_actors.Producer`
`method), 35`
`power_production_minimum()`
`(omegalpes.actor.operator_actors.producer_actors.Producer`
`method), 36`
`print_studied_period()`
`(omegalpes.general.time.TimeUnit method), 44`
`Producer` (class in `omegalpes.actor.operator_actors.producer_actors`),
34
`ProductionUnit` (class in
`omegalpes.energy.units.production_units`),
17
`Prosumer` (class in `omegalpes.actor.operator_actors.consumer_producer_`
36
`PublicAuthorities` (class in

- omegalpes.actor.regulator_actors.regulator_actors*), 38
- Q**
- Quantity (class in *omegalpes.general.optimisation.elements*), 41
- R**
- RCNetwork_1 (class in *omegalpes.energy.buildings.thermal*), 27
- read_enedis_data_csv_file() (in module *omegalpes.general.utils.input_data*), 46
- RegulatorActor (class in *omegalpes.actor.regulator_actors.regulator_actors*), 38
- remove_external_constraints() (*omegalpes.actor.actor.Actor* method), 31
- S**
- save_energy_flows() (in module *omegalpes.general.utils.output_data*), 46
- SawtoothEnergyUnit (class in *omegalpes.energy.units.energy_units*), 9
- select_csv_file_between_dates() (in module *omegalpes.general.utils.input_data*), 46
- set_energy_limits_on_time_period() (*omegalpes.energy.units.energy_units.EnergyUnit* method), 9
- set_power_balance() (*omegalpes.energy.energy_nodes.EnergyNode* method), 27
- SeveralConsumptionUnit (class in *omegalpes.energy.units.consumption_units*), 13
- SeveralEnergyUnit (class in *omegalpes.energy.units.energy_units*), 9
- SeveralImaginaryConsumptionUnit (class in *omegalpes.energy.units.consumption_units*), 14
- SeveralImaginaryProductionUnit (class in *omegalpes.energy.units.production_units*), 18
- SeveralProductionUnit (class in *omegalpes.energy.units.production_units*), 19
- ShiftableConsumptionUnit (class in *omegalpes.energy.units.consumption_units*), 14
- ShiftableEnergyUnit (class in *omegalpes.energy.units.energy_units*), 10
- ShiftableProductionUnit (class in *omegalpes.energy.units.production_units*), 20
- solve_and_update() (*omegalpes.general.optimisation.model.OptimisationModel* method), 45
- split_heating_and_cooling() (*omegalpes.energy.buildings.thermal.ThermalZone* method), 28
- SquareConsumptionUnit (class in *omegalpes.energy.units.consumption_units*), 15
- SquareEnergyUnit (class in *omegalpes.energy.units.energy_units*), 11
- SquareProductionUnit (class in *omegalpes.energy.units.production_units*), 21
- StorageUnit (class in *omegalpes.energy.units.storage_units*), 24
- sum_quantities_in_quantity() (in module *omegalpes.general.utils.plots*), 48
- Supplier (class in *omegalpes.actor.operator_actors.consumer_producer*), 37
- T**
- ThermalZone (class in *omegalpes.energy.buildings.thermal*), 27
- ThermoclineStorage (class in *omegalpes.energy.units.storage_units*), 25
- TimeUnit (class in *omegalpes.general.time*), 43
- TriangleEnergyUnit (class in *omegalpes.energy.units.energy_units*), 11
- U**
- update_units() (*omegalpes.general.optimisation.model.OptimisationModel* method), 45
- V**
- VariableConsumptionUnit (class in *omegalpes.energy.units.consumption_units*), 16
- VariableEnergyUnit (class in *omegalpes.energy.units.energy_units*), 11
- VariableProductionUnit (class in *omegalpes.energy.units.production_units*), 21
- W**
- write_linerazation_exp() (in module *omegalpes.energy.buildings.thermal*), 29
- Z**
- ZEA_RCNetwork_1 (class in *omegalpes.energy.buildings.thermal*), 28