# How to ZEN-garden

## Table of Contents

# 1. Setup

## 1.1. Needed Installations

- PyCharm (IDE, you can use other IDEs as well, but most users of ZEN-garden use PyCharm) [Install PyCharm](#)
- Anaconda (Needed for ZEN-garden environment creation) [Install Anaconda](#)
- Gurobi (Optimization Software) [Install Gurobi](#)
- (GitHub Desktop) [Install GitHub Desktop](#)

## 1.2. Steps

1. GitHub registration: If you don't have a GitHub account yet register at: [GitHub](#)

2. <u>Join ZEN-garden repository:</u> If you didn't receive a GitHub invitation, ask your supervisor to invite you to the repository (write them your GitHub email address) <u>ZEN-Garden Repository</u>

3. <u>Create your own branch:</u> In the ZEN-garden repository click on "branches" and then "new branch", choose "main" as the branch source and "development_ZENx_NS" (NS= name, surname) as its name

| | main ▾ | ⑂ 15 branches | ⊘ 4 tags | | Go to file | Add file ▾ | <> Code ▾ |
|---|---|---|---|---|---|---|---|

| | jacob-mannhardt Merge pull request #249 from RRE-ETH/development ... | | f5a76b6 last week | ⏱ 1,090 commits |
|---|---|---|---|---|

| 📁 | .github/workflows | Update auto-PR | 2 weeks ago |
|---|---|---|---|
| 📁 | data | insert gitkeep | last year |
| 📁 | euler | change order in which gurobi is loaded | 2 months ago |
| 📁 | notebooks | adapt to pep 8 | 3 months ago |
| 📁 | tests/testcases | update tests to new capex calculation | last week |
| 📁 | zen_garden | bug fix in yearly time index handling | last week |
| 📄 | .gitignore | adapt to pep 8 | 3 months ago |
| 📄 | README.md | Update README.md | 2 months ago |
| 📄 | setup.py | dummy commit | 4 months ago |
| 📄 | zen_garden_env.yml | Merge branch 'development_ZENpower' into development_ZENx_JM | last month |

| 🔍 Search branches... | | Overview | Yours | Active | Stale | All branches | New branch |
|---|---|---|---|---|---|---|---|

**Default branch**

| 🛡 main 🗐 Updated last week by jacob-mannhardt | | Default |
|---|---|---|

**Your branches**

| development_ZENx_LK 🗐 Updated 1 hour ago by Lukas-Kunz | 0 | 28 | ↥ New pull request | ✎ 🗑 |
|---|---|---|---|---|

**Active branches**

| development_ZENx_LK 🗐 Updated 1 hour ago by Lukas-Kunz | 0 | 28 | ↥ New pull request | ✎ 🗑 |
|---|---|---|---|---|
| development_ZENhydrogen 🗐 Updated 14 hours ago by aganter | 3 | 56 | ↥ New pull request | ✎ 🗑 |
| development_ZENpower 🗐 Updated yesterday by jacob-mannhardt | 7 | 21 | ↥ New pull request | ✎ 🗑 |
| development_linopy 🗐 Updated 5 days ago by jafluri | 8 | 8 | ↥ New pull request | ✎ 🗑 |
| development_ZENx_JM 🗐 Updated last week by jacob-mannhardt | 1 | 1 | ↥ New pull request | ✎ 🗑 |

View more active branches ›

4. <u>Cloning the repository:</u> To create a local copy of your branch on your computer, you must clone the remote repository from GitHub. <span style="color:red">It is important that you clone the repository to a path which doesn't contain any spaces!</span> (Don't clone to e.g. ./Users/Name Surname, otherwise you'll have issues while executing the framework)To clone your branch there's a more beginner-friendly way using GitHub Desktop and a more advanced way using Git Bash for example.
GitHub Desktop: [Clone Reposiotry with GitHub Desktop](#)

   To clone the repositry by using Git Bash, two methods are available: [HTTPS](#) or [SSH](#)

5. <u>ZEN-garden environment creation:</u> Open PyCharm to view the *zen_garden_env.yml* file (contained in the ZEN-garden folder), copy "*conda env create -f zen_garden_env.yml*" and run the command in your Anaconda prompt (takes several minutes), if the installation was successful, you can see the environment at *C:\Users\username\anaconda3\envs* or wherever Anaconda is installed

6. <u>Gurobi license:</u> To use all of Gurobi's functionalities, you need to obtain a free academic license: [Get your Gurobi license](#)
   Following these instructions, you'll get a Gurobi license key which you have to run in your command prompt to activate the license for your computer

7. <u>Create PyCharm Configurations:</u> To execute ZEN-garden's different functionalities configurations are used. To add them, follow the steps at "PyCharm Setup": [Create Configurations](#)

# 2. ZEN-garden configurations

## 2.1. Run ZEN-garden module

The ZEN-garden module can be executed in several ways as well as on ETH's EULER cluster. To check if the setup was successful, you can run one of the standardized test cases. To do so,

- open *ZEN-garden\tests\testcases* and copy paste all the tests to *ZEN-garden\data*
- copy *config.py* to the *data* directory
- choose "test_1a" as the dataset and execute it using one of the following methods

### 2.1.1. Run ZEN-garden using the "Run Module" configuration

Executing ZEN-garden with the created configuration "Run Module" (created in [setup step 7](#)) is the most forward way if you use PyCharm. Simply adjust the path in the analysis attribute "dataset" in the config.py file to one of the desired datasets and click the green run-button (have a look at the [config options](#) to get an overview of all the config settings).

```
1    from zen_garden.model import Config
2    import os
3
4    # create a config
5    config = Config()
6
7    # Analysis - Default dictionary
8    analysis = config.analysis
9    # Solver - Default dictionary
10   solver = config.solver
11   # Scenarios - Default scenario dictionary
12   scenarios = config.scenarios
13
14   # Analysis - settings update compared to default values
15   analysis["dataset"]                           = os.path.join(os.path.dirname(__file__),
16                                                     "test_1a")
17   analysis["objective"]              = "total_cost"              # choose from "TotalCost", "TotalCarbonEmissions", "Risk"
18   analysis["use_existing_capacities"]        = True             # use greenfield or brownfield approach
19   analysis["output_format"] = "h5"
20   # Solver - settings update compared to default values
21   #
22   solver["name"] = "gurobi_persistent" # free solver
23   # solver["solver_options"]["method"]        = 2
24   # solver["solver_options"]["node_method"]   = 2
25   solver["solver_options"]["bar_homogeneous"]   = 1
26   # solver["solver_options"]["presolve"]      = -1
27   solver["solver_options"]["threads"]        = 46
28   # solver["solver_options"]["crossover_basis"]   = 0
29   # solver["solver_options"]["crossover"]     = 0
30   solver["solver_options"]["scale_flag"]     = 2
31   solver["analyze_numerics"]                 = True
32   solver["immutable_unit"]                   = ["hour","km"]
```

### 2.1.2. Run ZEN-garden using a terminal

How to run the ZEN-garden package in a terminal is described in ZEN-garden as a package. Depending on the terminal you want to use, the procedure differs slightly. Before entering the module's execution command, ensure that the *data* folder is your working directory. To change the working directory from, e.g., *ZEN-garden* to *ZEN-garden/data*, simply run *cd data*.

PyCharm Power Shell (Terminal in PyCharm): As the zen-garden conda environment is activated by default, you can simply enter the following command followed by a chosen dataset name:

*(zen-garden) PS C:\Users\Lukas Kunz\ETH\ZEN_garden\ZEN-garden\data> python -m zen_garden --dataset="test_1b"*

Anaconda Prompt: The only difference when using the Anaconda prompt is that you have to activate the zen-garden environment manually before you can run the package execution command. This can be done by running *conda activate zen-garden.*

If your console looks something like the screenshot below, the ZEN-garden module works fine on your computer, and you can run all the data sets located in the *data* folder by choosing one of the two methods. Otherwise, revisit the setup steps according to the occurred error.
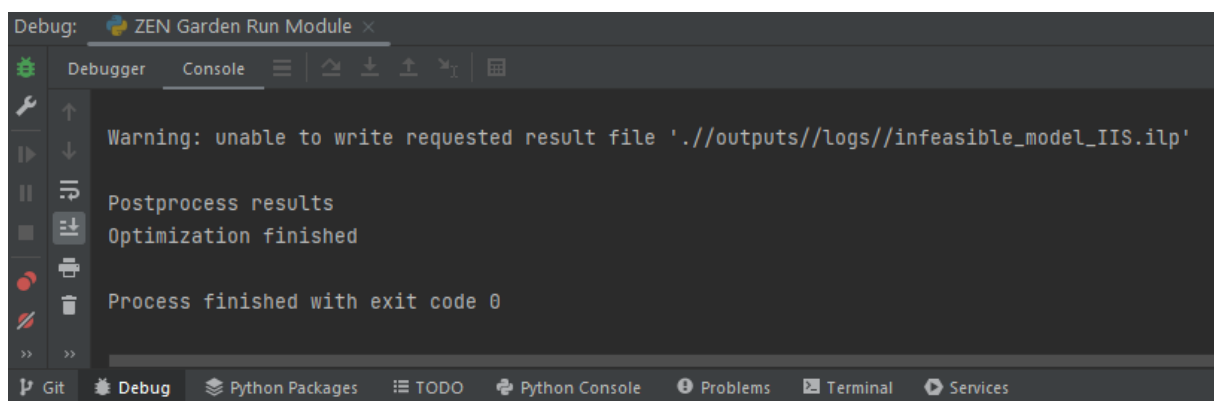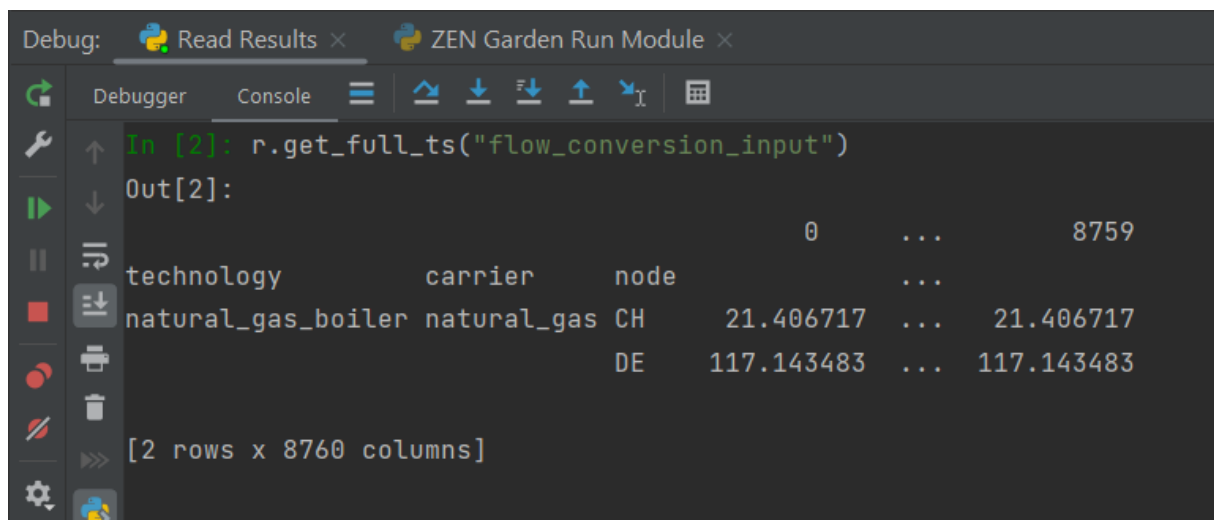
### 2.1.3.   Run ZEN-garden on EULER

To run computational more expensive optimization problems, ETH's EULER cluster can be accessed as described at [ZEN-garden on EULER](#).

## 2.2. Read Results

After a dataset's optimization problem has been executed, its results can be accessed and visualized with help of the *results.py* script. To get a first impression of the available results processing functionalities, the Jupyter Notebook *postprocess_results.ipynb* can help a lot. It can be found in ZEN-garden's *notebooks* directory.

Another way to access your results is to use the "Read Results" configuration. By running the *results.py* script, the different member functions of the contained "Results" class can be applied to the "Results" object to extract and plot the data of your optimization problem. Since the "Read Results" configuration creates an instance of the "Results" class, the object can be accessed by "self". By setting a break point at the end of the file, the debugger console can be used to apply the class's functions to the "Results" instance.



### 2.2.1.   How to plot your results

The class "Results" contains three member functions to plot the simulated data. Please have a look at [the plot discussion entry](#) to get further explanations.

### 2.2.2.   Accessing your results data

To access the data frames containing the raw optimization results of the variables and parameters, the following member functions of the "Results" class can be used:

1.   r.get_total(),
2.   r.get_full_ts(),
3.   r.get_df(),

where "r" is an instance of the "Results" class. r.get_total() returns the aggregated values of the variable and parameter values for each year. For hourly resolved variables, such as "flow_conversion_input", this is the sum over all hours of the year for each year. Yearly resolved variables, such as "capacity", remain unaltered because they are already yearly aggregates.

r.get_full_ts() returns the hourly evolution of hourly resolved variables. This is especially useful when using the time series aggregation, where the hours of the year are aggregated by representative time steps. r.get_full_ts() disaggregates the time series back to full hourly representation. Yearly values

remain in yearly resolution, thus for these components r.get_total() and r.get_full_ts() return the same result.

Under the hood of r.get_total() and r.get_full_ts(), we use the r.get_df() function to extract the raw variable and parameter values. If these are of interest, you can use r.get_df(), otherwise r.get_total() and r.get_full_ts() will be more useful.

### 2.2.2.1.    self.get_df()

The most fundamental function to access the data of a specific variable such as, e.g., "flow_conversion_input" is *self.get_df("flow_conversion_input")*. It returns a Pandas series containing all the "flow_conversion_input" values of the different technologies at the individual nodes at every time step.

| | |
|---|---|
| hard_coal_plant/hard_coal/FR/144 | 0.00000 |
| hard_coal_plant/hard_coal/FR/145 | 0.00000 |
| hard_coal_plant/hard_coal/FR/146 | 0.00000 |
| hard_coal_plant/hard_coal/FR/147 | 0.00000 |
| hard_coal_plant/hard_coal/FR/148 | 0.00000 |
| hard_coal_plant/hard_coal/FR/149 | 0.00000 |
| nuclear/uranium/CH/0 | 12.54420 |
| nuclear/uranium/CH/1 | 12.54420 |
| nuclear/uranium/CH/2 | 8.05693 |
| nuclear/uranium/CH/3 | 12.54420 |
| nuclear/uranium/CH/4 | 12.54420 |
| nuclear/uranium/CH/5 | 12.54420 |
| nuclear/uranium/CH/6 | 12.54420 |
| nuclear/uranium/CH/7 | 12.54420 |

### 2.2.2.2.    self.get_full_ts()

A more convenient way to access the same data is offered by *self.get_full_ts("flow_conversion_input")*, a function which creates a data frame of the variable's full time series.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| biomass_plant/biomass/FR | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 | 3.03632 |
| electrode_boiler/electricity/CH | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| electrode_boiler/electricity/DE | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| electrode_boiler/electricity/FR | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| hard_coal_boiler/hard_coal/CH | 0.15540 | 0.15540 | 0.15540 | 0.18068 | 0.18068 | 0.22886 | 0.22886 | 0.22886 | 0.18068 | 0.18068 | 0.18068 | 0.18068 | 0.18068 | 0.18068 | 0.18068 | 0.18068 |
| hard_coal_boiler/hard_coal/DE | 3.28069 | 3.28069 | 3.28069 | 0.00000 | 0.00000 | 4.78539 | 4.78539 | 4.78539 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| hard_coal_boiler/hard_coal/FR | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.38259 | 0.38259 | 0.38259 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| hard_coal_plant/hard_coal/CH | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| hard_coal_plant/hard_coal/DE | 0.00000 | 0.00000 | 0.00000 | 17.43686 | 17.43686 | 12.65147 | 12.65147 | 12.65147 | 17.43686 | 17.43686 | 17.43686 | 17.43686 | 17.43686 | 17.43686 | 17.43686 | 17.43686 |
| hard_coal_plant/hard_coal/FR | 1.68525 | 1.68525 | 1.68525 | 1.68525 | 1.68525 | 1.30265 | 1.30265 | 1.30265 | 1.68525 | 1.68525 | 1.68525 | 1.68525 | 1.68525 | 1.68525 | 1.68525 | 1.68525 |
| heat_pump/electricity/CH | 2.91542 | 2.91542 | 2.91542 | 3.58026 | 3.58026 | 4.84721 | 4.84721 | 4.84721 | 3.58026 | 3.58026 | 3.58026 | 3.58026 | 3.58026 | 3.58026 | 3.58026 | 3.58026 |
| heat_pump/electricity/DE | 18.19193 | 18.19193 | 18.19193 | 23.78045 | 23.78045 | 31.87778 | 31.87778 | 31.87778 | 23.78045 | 23.78045 | 23.78045 | 23.78045 | 23.78045 | 23.78045 | 23.78045 | 23.78045 |
| heat_pump/electricity/FR | 13.92653 | 13.92653 | 13.92653 | 17.69159 | 17.69159 | 24.85001 | 24.85001 | 24.85001 | 17.69159 | 17.69159 | 17.69159 | 17.69159 | 17.69159 | 17.69159 | 17.69159 | 17.69159 |
| lignite_coal_plant/lignite/CH | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| lignite_coal_plant/lignite/DE | 0.00000 | 0.00000 | 0.00000 | 30.95859 | 30.95859 | 30.95859 | 30.95859 | 30.95859 | 30.95859 | 30.95859 | 30.95859 | 30.95859 | 30.95859 | 30.95859 | 30.95859 | 30.95859 |
| lignite_coal_plant/lignite/FR | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| natural_gas_boiler/natural_gas/CH | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 2.29462 | 2.29462 | 2.29462 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| natural_gas_boiler/natural_gas/DE | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 1.83633 | 1.83633 | 1.83633 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| natural_gas_boiler/natural_gas/FR | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 1.91709 | 1.91709 | 1.91709 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| natural_gas_turbine/natural_gas/CH | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| natural_gas_turbine/natural_gas/DE | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 18.10258 | 18.10258 | 18.10258 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00000 |
| natural_gas_turbine/natural_gas/FR | 0.00000 | 0.00000 | 0.00000 | 0.94681 | 0.94681 | 7.20783 | 7.20783 | 7.20783 | 0.94681 | 0.94681 | 0.94681 | 0.94681 | 0.94681 | 0.94681 | 0.94681 | 0.94681 |
| nuclear/uranium/CH | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 | 12.54420 |
| nuclear/uranium/DE | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 | 76.19843 |
| nuclear/uranium/FR | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 | 171.81000 |

### 2.2.2.3.  self.get_total()

If you're not interested in the hourly resolution of the variable values,
*self.get_total("flow_conversion_input")* can be used to obtain the yearly sums of the hourly data.

| | 0 | 1 | 2 |
|---|---|---|---|
| biomass_plant/biomass/DE | | | |
| biomass_plant/biomass/FR | 6928.874… | 6385.374… | 6278.759… |
| electrode_boiler/electricity/CH | 500.99039 | 1438.128… | 1497.060… |
| electrode_boiler/electricity/DE | 6826.321… | 5974.843… | 5830.382… |
| electrode_boiler/electricity/FR | 19313.41… | 31253.19… | 34107.87… |
| hard_coal_boiler/hard_coal/CH | 703.04750 | 696.99206 | 696.99206 |
| hard_coal_boiler/hard_coal/DE | 19835.34… | 19741.43… | 19520.49… |
| hard_coal_boiler/hard_coal/FR | 783.73352 | 783.73352 | 1002.791… |
| hard_coal_plant/hard_coal/CH | 0.00000 | 0.00000 | 0.00000 |
| hard_coal_plant/hard_coal/DE | 40596.64… | 40199.47… | 40257.89… |
| hard_coal_plant/hard_coal/FR | 2585.724… | 2726.462… | 2478.375… |
| heat_pump/electricity/CH | 18526.75… | 18755.79… | 18755.79… |
| heat_pump/electricity/DE | 128462.6… | 129016.8… | 129016.8… |
| heat_pump/electricity/FR | 98406.84… | 98406.84… | 98522.42… |
| lignite_coal_plant/lignite/CH | 0.00000 | 0.00000 | 0.00000 |
| lignite_coal_plant/lignite/DE | 117739.8… | 115236.4… | 113795.9… |
| lignite_coal_plant/lignite/FR | 0.00000 | 0.00000 | 0.00000 |
| natural_gas_boiler/natural_gas/CH | 2353.467… | 2166.532… | 2166.532… |
| natural_gas_boiler/natural_gas/DE | 3429.868… | 3003.047… | 3324.969… |
| natural_gas_boiler/natural_gas/FR | 1574.848… | 1728.270… | 1624.832… |
| natural_gas_turbine/natural_gas/CH | 0.00000 | 0.00000 | 0.00000 |
| natural_gas_turbine/natural_gas/DE | 34633.01… | 34296.42… | 33245.51… |
| natural_gas_turbine/natural_gas/FR | 4576.558… | 5035.394… | 5534.793… |
| nuclear/uranium/CH | 92638.35… | 93954.40… | 87898.20… |
| nuclear/uranium/DE | 645155.8… | 696369.0… | 723363.1… |
| nuclear/uranium/FR | 1273541.… | 1321776.… | 1327109.… |

### 2.2.3.  Compare two datasets

You can compare two "Results" objects by using the following class methods. They can help you to get a fast overview of two datasets' differences which facilitates spotting the reasons for errors. Again, the Jupyter Notebook shows some practical examples, but the functionalities can be used in the debug console as well by creating the desired "Results" objects the way it is done in the beginning of the notebook. All the functions take a list of two "Results" instances as their input argument.

- relate the configs of two datasets:
  *Results.compare_configs([result_instance_1, result_instance_2])*
- relate model parameters:
  *Results.compare_model_parameters([result_instance_1, result_instance_2])*
- relate model variables:
  *Results.compare_model_variables([result_instance_1, result_instance_2])*

### 2.3. Run Tests

The main purpose of the test files is their usage for the automated testing functionality of ZEN-garden. By comparing the variables' values gathered by simulating the testcases with some reference values, the correctness of the current framework code can be proved. Whenever you adapted some framework code, you can use the run test configuration to ensure that ZEN-garden does still function properly.

## 3. Parameters, variables, and constraints

An important concept in ZEN-garden, or for optimization problems in general, is the definition of parameters, variables, and constraints. Parameters are used to store data that is immutable, meaning once a parameter's values are specified, they stay the same for the whole optimization (e.g., the hourly electricity demand per country). On the other hand, variables represent quantities whose values are computed by solving the optimization problem (e.g., the hourly electricity output flow of a gas turbine). By defining constraints, the parameters and variables can be related to each other such that they follow the rules of physical properties etc. (e.g., energy conservation at nodes). In the example optimization problem below, $c^T x$ is the so-called objective function whose value is optimized (mostly minimizing the net present cost of the entire system), $x$ and $b$ are vectors containing all the variables and parameters, respectively, which are related by constraints of the form $Ax \leq b$. Additionally, some variables are defined as non-negative numbers, i.e., $x \geq 0$, as physical metrics like costs, power flows and energy etc. can only be positive.

$$\min_{x} c^T x$$
$$s.t. \, Ax \leq b$$
$$x \geq 0$$

To get an overview of all the existing parameters and variables you can access your "Results" object in your debugger or in the Jupyter Notebook, which contains the dictionary *component_names*, which itself contains the dictionaries *pars* and *vars*. Inside these dictionaries, you can find all the parameters'/variables' names.

To find the definitions of all the parameters, variables and constraints you can look up every appearance of *add_parameter/add_variable/add_constraint* in all of ZEN-garden's files by using CTRL+Shift+F. Assessing the definitions can be quite helpful to get a better understanding as they include the *doc* strings, a brief explanation of the underlying parameter, variable or constraint. In addition, it can be seen in which file *(technology.py, carrier.py*, etc.) the definition is located, revealing some extra information. Since this method takes some time to find the desired doc string, the *Results* class contains the function *r.get_doc("component")* which returns the doc string of the corresponding component.

# 4. Input data structure

The input data of a dataset must be composed of the *system.py* file and the five folders *set_carriers*, *set_conversion_technologies*, *set_transport_technologies*, *set_storage_technologies* and *system_specification*.

## 4.1. system.py

The *system.py* file must contain the sets of technologies that constitute the energy system, i.e. , that take part in supplying the final energy demands. You can have technologies in your input data folder but not list them in the system.py. In this case, they are excluded from the optimization. Additionally, a subset of nodes (from *system_specification/set_nodes.csv*), the starting year of the optimization (*reference_year*) and a lot of other time related specifications can be defined. The time step parameters are discussed in this Git Discussion. To get an overview of how to define the different properties, have a look at the system settings.

## 4.2. set_carriers

The *set_carriers* folder contains the energy carrier types such as electricity, heat, biomass, natural gas, etc. All the carriers that are needed by the technologies specified in the *system.py* file must be contained in this directory; additional carriers are allowed as well. You do not need to specifically list the carriers in system.py as they are implied by the included technologies.

To define a specific carrier, a folder named after the carrier containing the attributes file must be created.
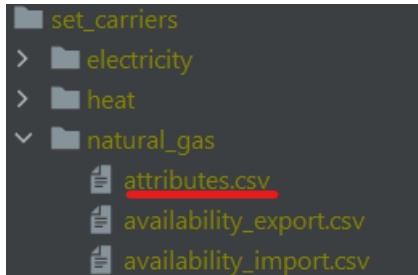


The attributes file contains all the default values of the parameters' needed to describe the carrier. As the parameters' values can differ along the energy systems' nodes and the simulated time steps, the variations can be described by creating additional input files having the following name structure (parameter name without the "default" ending):
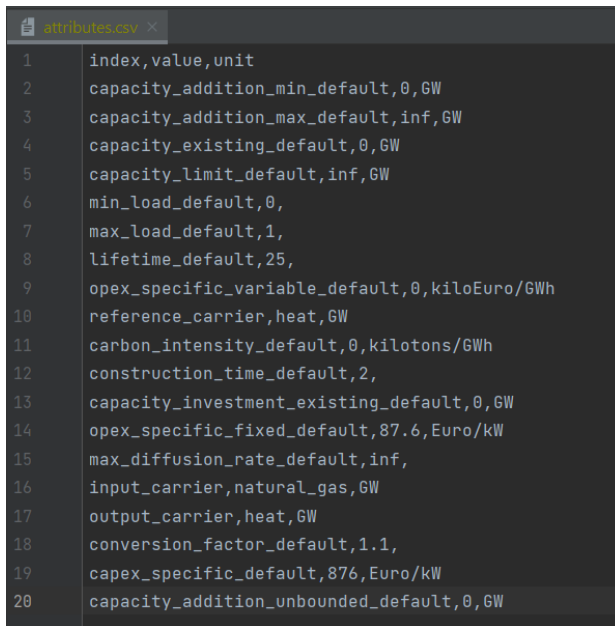
- **demand.csv:** If there exists a demand for a carrier, it can be described in the demand file.
- **availability_import.csv:** This file can be used to specify different values of a carrier's import availability as it may differ for the nodes, time steps etc.
- **availability_export.csv:** As for the import availability the export availability can be customised.
- **...**

Examples of existing parameters can be assessed in the attribute files of the test datasets (for completion, the whole set of parameters can be found in the appendices). To get a better understanding of how to structure these additional input files, have a look at the "Spreadsheet structure" section.



```
attributes.csv
1    index,value,unit
2    carbon_intensity_default,0.0,kilotons/GWh
3    demand_default,0.0,GW
4    price_shed_demand_default,inf,kiloEuro/GWh
5    availability_import_default,0.0,GW
6    availability_export_default,0.0,GW
7    availability_import_yearly_default,inf,GWh
8    availability_export_yearly_default,inf,GWh
9    price_export_default,0.0,kiloEuro/GWh
10   price_import_default,0.0,kiloEuro/GWh
```

### 4.3. set_conversion_technologies

The *set_conversion_technologies* folder contains the energy conversion technologies such as boilers, power plants (e.g., lignite coal plants), or renewables. All the conversion technologies that are specified in the system file's technology sets must be contained in this directory; additional conversion technologies are allowed. The procedure of defining a specific conversion technology is the very same as for energy carriers, described in the previous section. Again, a folder with the conversion technology's name must be created, including the attributes file for conversion technologies and variations in space and time can be specified with additional input data files.

```
attributes.csv ×
1    index,value,unit
2    capacity_addition_min_default,0,GW
3    capacity_addition_max_default,inf,GW
4    capacity_existing_default,0,GW
5    capacity_limit_default,inf,GW
6    min_load_default,0,
7    max_load_default,1,
8    lifetime_default,25,
9    opex_specific_variable_default,0,kiloEuro/GWh
10   reference_carrier,heat,GW
11   carbon_intensity_default,0,kilotons/GWh
12   construction_time_default,2,
13   capacity_investment_existing_default,0,GW
14   opex_specific_fixed_default,87.6,Euro/kW
15   max_diffusion_rate_default,inf,
16   input_carrier,natural_gas,GW
17   output_carrier,heat,GW
18   conversion_factor_default,1.1,
19   capex_specific_default,876,Euro/kW
20   capacity_addition_unbounded_default,0,GW
```

### 4.4. set_transport_technologies

The *set_transport_technologies* folder contains the energy transport technologies such as natural gas pipelines or power lines. All the transport technologies that are specified in the system file's technology sets must be contained in this directory; additional transport technologies are allowed. Once more, the individual transport technologies must be defined the same way as carriers and other technologies.

### 4.5. set_storage_technologies

The *set_storage_technologies* folder contains the energy storage technologies such as pumped hydro, natural gas storages, batteries, etc. All the storage technologies that are specified in the system file's technology sets must be contained in this directory; additional storage technologies are allowed. Again, the procedure of defining them is equivalent as before.

### 4.6. system_specification

The *system_specification* folder contains additional input data that is needed to define the energy system as a whole. Other than the carrier and technology folders, this folder must contain more files than just the attributes file:

- attributes.csv: carbon emissions related information etc.
- base_units.csv: definition of base units to which input data units are converted
- set_edges.csv: definition of existing connections (edges) between node pairs
- set_nodes.csv: set of all nodes along with its longitude and latitude, which are used to compute the distances between the nodes (=length of edges). These distances are needed to compute distance-dependent variables such as the cost of building transport technologies.

- unit_definitions.txt: definition of additional units

## 4.7. Spreadsheet structure

The individual values at different nodes and time steps can be entered into the input files by using the column headers "node" and "time"/"year" as it is done in the pictures.



The header "time" is used to represent hourly time steps, whereas the "year" header serves for yearly time steps (time step discussion). An overview of the parameters' time step types is given in the appendices. In addition to the one-dimensional input structure above, data varying in space (nodes) and time can be structured by stating the nodes and time steps explicitly:



Thanks to ZEN garden's capability of completing required parameter values which are not stated in the input files explicitly, the user doesn't have to specify the values for all indices of the parameter. For example, the parameter *availability_import* is defined by the *index_sets* "set_nodes" and "set_time_steps", however it is possible to only provide data for the individual nodes and none for the different time steps (see screen shot above). The input data handling will then complete the "missing" values by using the nodes' individual import availabilities for all the time steps (time independent parameter). Therefore, the framework user can choose for each parameter, if the default value (specified in the attributes file) or individual values for the parameter's index sets should be used.

```python
41
42
43         "] = self.data_input.extract_input_data("demand", index_sets=["set_nodes", "set_time_steps"], time_steps=set_base_time_
44         bility_import"] = self.data_input.extract_input_data("availability_import", index_sets=["set_nodes", "set_time_steps"],
45         bility_export"] = self.data_input.extract_input_data("availability_export", index_sets=["set_nodes", "set_time_steps"],
46         export"] = self.data_input.extract_input_data("price_export", index_sets=["set_nodes", "set_time_steps"], time_steps=se
47         import"] = self.data_input.extract_input_data("price_import", index_sets=["set_nodes", "set_time_steps"], time_steps=se
```
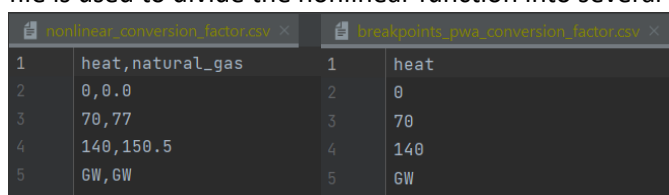
## 4.8. Additional methods to enter input data

### 4.8.1. PWA

To approximate nonlinear functions such as the conversion efficiency of e.g., heat pumps the so-called piecewise affine (PWA) approximation can be used. In this method, the nonlinear function is divided into several regions where it can be approximated as a linear function. To specify such PWA input data a similar combination of the following two files is needed (in the folder of the corresponding technology):

- *breakpoints_pwa_conversion_factor.csv* and *nonlinear_conversion_factor.csv* to approximate the nonlinear conversion efficiency of e.g., a natural gas boiler
- *breakpoints_pwa_coapex.csv* and *nonlinear_capex.csv* to approximate the nonlinear relation of capex and installed capacity of a technology
- …

The "nonlinear" file must thereby contain the values for the nonlinear relation between the two metrics (e.g., heat output vs. natural gas input) followed by a pair of units whereas the "breakpoints" file is used to divide the nonlinear function into several intervals (#intervals = #breakpoints - 1).



### 4.8.2. Define technology with multiple input/output carriers

To define a conversion technology with multiple input and output carriers, several carrier types can be specified as the technology's input/output carriers in its corresponding attributes file (e.g., heat and carbon as output carrier). Having several input/output carriers requires additional conversion factors, i.e., the factor relating the amount generated/consumed of a carrier with respect to the one's of the reference carrier. Therefore, the *conversion_factor_default* needs to be overwritten by providing the additional *conversion_factor.csv* file, where all conversion factors need to be specified with respect to the *reference_carrier* (e.g., since the reference carrier in the screenshot is heat and the two other carriers are natural gas and carbon, the conversion factors relating heat with natural gas (1.1 GWh NG/GWh heat) and heat with carbon (0.01 kt carbon/GWh heat) need to be declared).

```
attributes.csv ×
1    index,value,unit
2    capacity_addition_min_default,0,GW
3    capacity_addition_max_default,inf,GW
4    capacity_existing_default,0,GW
5    capacity_limit_default,inf,GW
6    min_load_default,0,
7    max_load_default,1,
8    lifetime_default,25,
9    opex_specific_variable_default,0,kiloEuro/GWh
10   reference_carrier,heat,GW
11   carbon_intensity_default,0,kilotons/GWh
12   construction_time_default,0,
13   capacity_investment_existing_default,0,GW
14   opex_specific_fixed_default,87.6,Euro/kW
15   max_diffusion_rate_default,inf,
16   input_carrier,natural_gas,GW
17   output_carrier,heat_carbon,GW
18   conversion_factor_default,1.1,
19   capex_specific_default,876,Euro/kW
20   capacity_addition_unbounded_default,0,GW
```

```
conversion_factor.csv ×
1    year,natural_gas,carbon
2    2022,1.1,0.01
3    unit,GWh/GWh,kilotons/GWh
4
```

### 4.8.3.  Define input data with yearly variations

To simplify the hourly dependent input data which varies each year by a specific factor the "yearly_variation" file can be used. For example, if the heat demand is expected to increase or decrease over the years by a known percentage, the change can be specified as it is done in the following figure instead of defining all the values explicitly in the heat demand file. By doing so, the demand values will be scaled accordingly to the yearly variation factors (e.g., demand CH in year 2022 and time step 0 will be 9).

```
demand_yearly_variation_4.csv ×
1    year,CH,DE
2    2022,0.9,1.1
3    2023,1.0,1.2
4
```

```
demand.csv ×
1    time,CH,DE
2    0,10,100
3    1,20,90
```
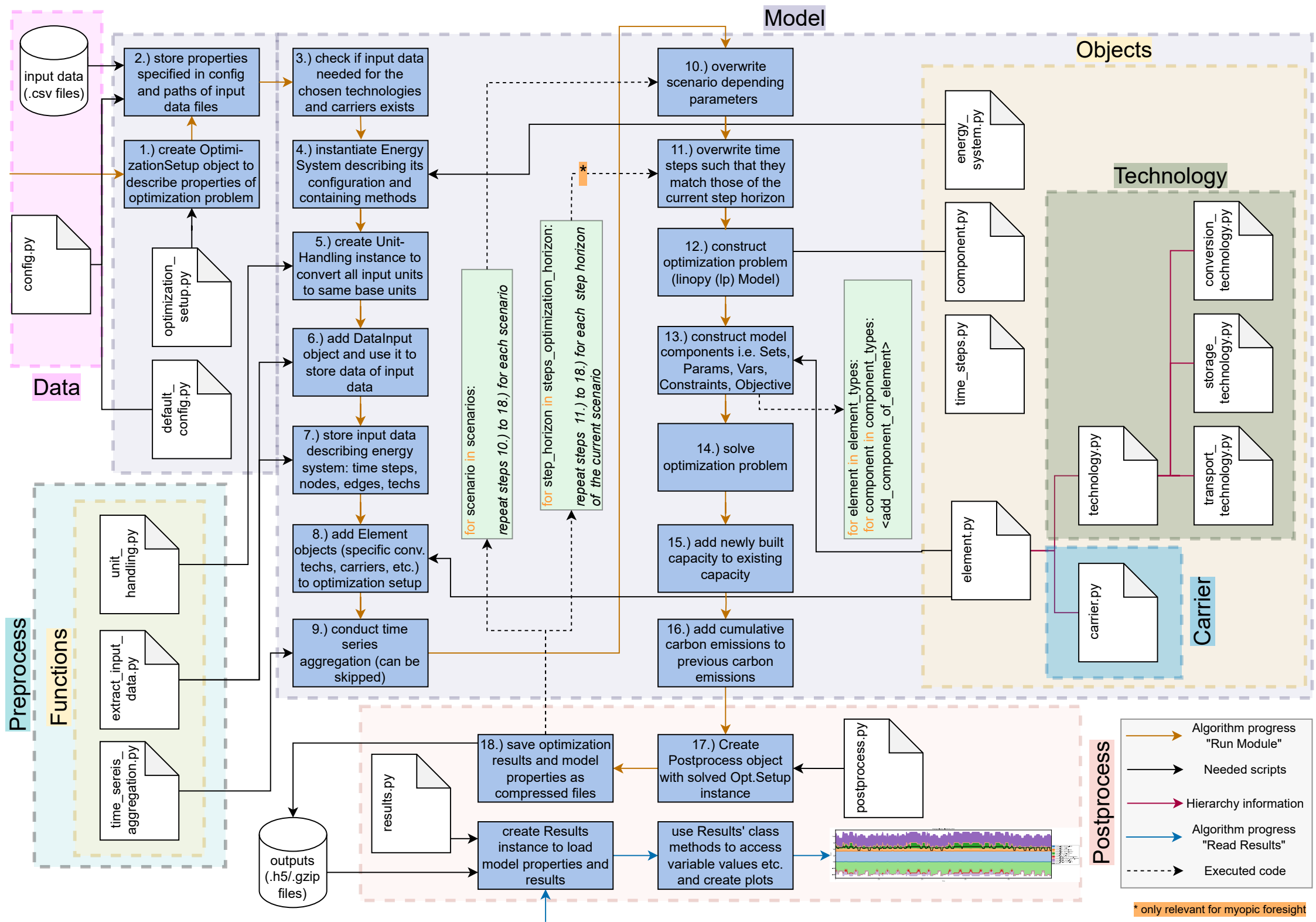
# 5. Framework structure

ZEN-garden is structured into the three building blocks *preprocess*, *model* and *postprocess*. *model* is used to describe the optimization problem of the energy system containing the different technologies and carriers, *preprocess* extracts the provided input data and *postprocess* saves and visualizes the simulation results. To get a better understanding of the general order of ZEN-gardens execution steps and the package levels, have a look at the flowchart.

## 5.1. Preprocess

### 5.1.1. Functions

#### extract_input_data.py

The functions to extract the data from the differently structured spreadsheets and store the information in dataframes are in this script. As there are a lot of different ways in which the input data itself or the description of specific parameters can be specified (linear/PWA, from attributes/extra file, etc.), the data extraction process is quite complicated, leading to the large number of functions.

#### time_series_aggregation.py

To reduce the complexity and thus the computational cost of optimizing energy systems with hourly data resolution, the time series of the underlying data can be aggregated to decrease the number of time steps with individual data values. For example, *test_4b* aggregates the 8760 hourly time steps of a year to ten representative time steps, thus reducing the computational effort heavily while approximating the original data still well. The time series aggregation parameters are defined in the system file and further details can be found at: TSA discussion

#### unit_handling.py

To ensure that all the units of the input data are consistent, a unit-handling is implemented. By specifying a set of base units (*system_specification/base_units.csv*), the units used in the input data files do not have to be consistent as the unit handling will transform the units according to the chosen base units (unit handling discussion).

## 5.2. Model

### default_config.py

The default configuration is defined in the *Config* class which describes all the relevant specifications for ZEN-garden with the four dictionaries:

- **analysis**: describes the desired analysis such as the objective (e.g., total cost), if the problem should be minimized or maximized, the discount rate, etc.
- **solver**: contains the information regarding the solver used to solve the optimization problem such as its name (e.g. glpk (free) or gurobi_persistent (commercial; free with academic license)) and different corresponding solver specifications (Gurobi documentation)
- **system**: describes the energy system; used to select technologies which are included in the optimization problem and to define a subset of nodes which should exist (more technologies and nodes allowed in input data; see above), contains time related parameters to specify time series aggregation (TSA discussion)
- **scenarios**: used to define the individual settings of multi-scenario simulations (see *test_6a*)

To specify changes from the default configuration the files *config.py*, *system.py* and *scenarios.py* are used.

- **config.py:** The config.py file is in the *data* and the *testcases* folder. The changes made in these files apply to all the datasets contained in the corresponding folder. For example, if you specify a solver name in the *data* config, it affects all the datasets located in the *data* directory. As described in the "Run ZEN-garden module" section, the config can be used to select which dataset should be executed.
- **system.py:** Parameters that describe the energy system are changed in the system.py file and only apply to the specific dataset in which the system.py file is located. Each dataset must contain its own system file which should look similar as:

```
9    system = dict()
10
11   ## System - settings update compared to default values
12   system['set_conversion_technologies']    = ["natural_gas_turbine","hard_coal_plant",
13                                                "nuclear","lignite_coal_plant",
14                                                "wind_onshore","photovoltaics",
15                                                "biomass_plant","reservoir_hydro",
16                                                "heat_pump","natural_gas_boiler","oil_boiler","electrode_boiler","biomass_boiler","hard_coal_boiler"]
17   system['set_storage_technologies']       = ["natural_gas_storage","pumped_hydro"]
18   system['set_transport_technologies']     = ['power_line',"natural_gas_pipeline"]
19
20   system['set_nodes']                      = ["DE","FR","CH"]
21   # time steps
22   system["reference_year"] = 2022
23   system["unaggregated_time_steps_per_year"] = 8760
24   system["aggregated_time_steps_per_year"] = 50
25   system["conduct_time_series_aggregation"] = True
26
27   system["optimized_years"] = 3
28   system["interval_between_years"] = 2
29   system["use_rolling_horizon"] = False
30   system["years_in_rolling_horizon"] = 1
```

- **scenarios.py:** To specify the individual scenarios of a multi-scenario simulation, this file is used similarly as in *test_6a*. By defining additional input data files, individual parameter values can be modified with respect to the default dataset, thus allowing a more efficient way than running a completely new dataset (Scenario Analysis).

### optimization_setup.py

The *OptimizationSetup* class defines the optimization model by saving the properties of the *analysis* and *system* dictionaries. Using this information, the class adds the specified carriers and technologies to the optimization model such that it can be solved with its built-in solving method afterwards.

### 5.2.1.  Objects

#### component.py

The *Component* class is used to add the parameters, variables and constraints to the optimization model represented by a linopy model. Since the linopy modelling language requires specific ways of how these parameters, variables and constraints must be constructed to suit its properties, *component.py* is needed. Component.py is therefore needed to adapt the parameters constructed via element.py such that they can be added to the linopy optimization model.

#### element.py

The *Element* class serves as the incremental building block of all technologies and carriers by defining all the necessary methods to define specific carriers and technologies. Therefore, it is the parent class of the *Carrier* and the *Technology* class.

#### energy_system.py

The *EnergySystem* class contains methods to add parameters, variables and constraints, to the optimization problem. In general, these components concern system-wide properties such as carbon or cost metrics. Additionally, the connections between the individual nodes are calculated and the objective function is created and passed to the concrete model.

#### time_steps.py

Contains a helper class containing methods to deal with timesteps.

#### 5.2.1.1.  Carrier

#### carrier.py

This script defines the class *Carrier*, which describes the individual energy carriers such as electricity, heat, biomass, etc. By extracting the corresponding input files, the carrier-related parameters, variables, and constraints are created.

#### 5.2.1.2.  Technology

#### technology.py

Contains the *Technology* class which defines all the technology-related parameters, variables and constraints that hold for all the existing technologies.

#### conversion_technology.py

Creates parameters, variables and constraints specifically needed for conversion technologies (e.g., natural gas boiler).

#### storage_technology.py

Creates parameters, variables and constraints specifically needed for storage technologies (e.g., pumped hydro storage).

#### transport_technology.py

Creates parameters, variables and constraints specifically needed for transport technologies (e.g., power lines).

## 5.3. Postprocess

### 5.3.1.  postprocess.py

The *Postprocess* class saves all the information contained in the optimization problem and all the system configurations such that the gathered solution and the whole setup can be accessed without running the optimization again.

### 5.3.2. results.py

The "Results" class can read the files created with the Postprocess class and contains methods to visualize these results.

# 6. Appendices

## 6.1. Config Settings

### System

system[Dictionary Key] = Value Type(Value Options)

e.g.:

```
system['set_transport_technologies'] = ['power_line','natural_gas_pipeline','carbon_pipeline']
```

| Dictionary Key: | Value Options/Meaning: | Value Type: |
|---|---|---|
| "set_conversion_technologies" | Subset of the names of the folders located in the *set_conversion_technologies* directory of the dataset | List of strings |
| "set_storage_technologies" | Subset of the names of the folders located in the *set_storage_technologies* directory of the dataset | List of strings |
| "set_transport_technologies" | Subset of the names of the folders located in the *set_transport_technologies* directory of the dataset | List of strings |
| "set_nodes" | Nodes defined in *set_nodes.csv* | List of strings |
| "reference_year" | Starting year of optimization | Positive integer >=1900 |
| "unaggregated_time_steps_per_year" | First x hours of year (normally 8760 for full year) | Positive integer |
| "aggregated_time_steps_per_year" | Number of representative time steps for time series aggregation | Positive integer |
| "conduct_time_series_aggregation" | True/False | Bool (default: False) |
| "optimized_years" | Number of optimized years | Positive integer |
| "interval_between_years" | Every "i-th" year is optimized (e.g. every second year for interval of 2) | Positive integer |
| "use_rolling_horizon" | Switch between perfect- and myopic foresight | Bool (default: False) |
| "years_in_rolling_horizon" | Number of years in foresight horizon | Positive integer |
| "double_capex_transport" | Account transport technology capex twice: per installed capacity (€/W) and per installed capacity per length (€/(W*m)) | Bool (default: False) |

| "exclude_parameter_from_TSA" | Choose if parameters defined in the optional file *exclude_parameter_from_TSA.csv* should be excluded from TSA | Bool (default: True) |
|---|---|---|
| "conduct_scenario_analysis" | Choose if scenario analysis should be conducted | bool (default: False) |
| "run_default_scenario" | Choose if default scenario should be executed or not | Bool (default: True) |
| "clean_sub_scenarios" | Choose if result files of scenarios that are not in the current scenario dict should be deleted | Bool (default: False) |
| "knowledge_depreciation_rate" | Rate at which knowledge stock of existing capacities is depreciated annually | Positive float $\in [0,1]$ |

## Analysis

| Dictionary Key: | Value Options/Meaning: | Value Type: |
|---|---|---|
| "objective" | "total_cost" or "total_carbon_emissions" | String |
| "sense" | "minimize" or "maximize" | String |
| "time_series_aggregation" | See tsam package | Dictionary |
| "folder_output" | Specify path of output folder to store optimization results (e.g. "./outputs/") | String |
| "output_format" | "h5", "json" or "gzip" | String |
| "max_output_size_mb" | Limit the maximum file size of json result files | Positive Integer (default: 500) |
| "use_capacities_existing" | Choose if existing capacity should be considered (brownfield vs. greenfield) | Bool (default: False) |

## Solver

| Dictionary Key: | Value Options/Meaning: | Value Type: |
|---|---|---|
| "name" | Choose solver: "glpk" or "gurobi" | String (default: "glpk") |
| "solver_options" | Additional solver options (see *default_config.py*) | Dictionary |
| "keep_files" | Whether to keep temporary solver files | Bool (default: False) |
| "io_api" | Solver option: See linopy package | String |
| "analyze_numerics" | Whether details about the optimization problem's numerics should be printed in the console | Bool (default: False) |
| "recommend_base_units" | Check if there is a better set of base units in terms of the optimizations numerics | Bool (default: False) |

| "immutable_unit" | Define base units which must not be changed to find a better set of base units | List of strings |
|---|---|---|
| "range_of_exponents" | Range of exponents the base units are allowed to be shifted to become "better" e.g. {"min": -2, "max": 3} →base units can be decreased by 10^-2 or increased by 10^3 | Dictionary (default: {"min": -3, "max": 3} |
| "define_ton_as_metric_ton" | Whether the unit "ton" should be a metric ton (True) or an imperial ton (False) | Bool (default: True) |
| "rounding_decimal_points" | Rounding tolerance for new capacity and unit multipliers | Positive integer (default: 5) |
| "rounding_decimal_points_ts" | Rounding tolerance for time series after TSA | Positive integer (default: 5) |
| "linear_regression_check" | Tolerances to determine if PWA-modelled x-y relationships can be represented by a linear slope while meeting these tolerances | Dictionary (default: {"eps_intercept": 0.1, "epsRvalue": 1-1E-5} |

## 6.2. Parameter, Variable and Constraint Overview

| Parameters | | | |
|---|---|---|---|
| **Name:** | **Time Step Type:** | **Doc String:** | **Scope:** |
| carbon_emissions_annual_limit | set_time_steps_yearly | Parameter which specifies the total limit on carbon emissions | system |
| carbon_emissions_budget | temporal immutable | Parameter which specifies the total budget of carbon emissions until the end of the entire time horizon | system |
| carbon_emissions_cumulative_existing | temporal immutable | Parameter which specifies the total previous carbon emissions | system |
| price_carbon_emissions | set_time_steps_yearly | Parameter which specifies the yearly carbon price | system |
| price_carbon_emissions_budget_overshoot | temporal immutable | Parameter which specifies the carbon price for budget overshoot | system |
| price_carbon_emissions_annual_overshoot | temporal immutable | Parameter which specifies the carbon price for annual overshoot | system |
| market_share_unbounded | temporal immutable | Parameter which specifies the unbounded market share | system |
| knowledge_spillover_rate | temporal immutable | Parameter which specifies the knowledge spillover rate | system |
| time_steps_operation_duration | set_time_steps_operation | Parameter which specifies the time step duration in operation for all technologies | system |
| demand | set_time_steps_operation | Parameter which specifies the carrier demand | carrier |
| availability_import | set_time_steps_operation | Parameter which specifies the maximum energy that can be imported from outside the system boundaries | carrier |
| availability_export | set_time_steps_operation | Parameter which specifies the maximum energy that can be exported to outside the system boundaries | carrier |
| availability_import_yearly | set_time_steps_yearly | Parameter which specifies the maximum energy that can be imported from outside the system boundaries for the entire year | carrier |

| | | | |
|---|---|---|---|
| availability_export_yearly | set_time_steps_yearly | Parameter which specifies the maximum energy that can be exported to outside the system boundaries for the entire year | carrier |
| price_import | set_time_steps_operation | Parameter which specifies the import carrier price | carrier |
| price_export | set_time_steps_operation | Parameter which specifies the export carrier price | carrier |
| price_shed_demand | temporal immutable | Parameter which specifies the price to shed demand | carrier |
| carbon_intensity_carrier | set_time_steps_yearly | Parameter which specifies the carbon intensity of carrier | carrier |
| capacity_existing | temporal immutable | Parameter which specifies the existing technology size | technology |
| capacity_investment_existing | set_time_steps_yearly_entire_horizon | Parameter which specifies the size of the previously invested capacities | technology |
| capacity_addition_min | temporal immutable | Parameter which specifies the minimum capacity addition that can be installed | technology |
| capacity_addition_max | temporal immutable | Parameter which specifies the maximum capacity addition that can be installed | technology |
| capacity_addition_unbounded | temporal immutable | Parameter which specifies the unbounded capacity addition that can be added each year (only for delayed technology deployment) | technology |
| lifetime_existing | temporal immutable | Parameter which specifies the remaining lifetime of an existing technology | technology |
| capex_capacity_existing | temporal immutable | Parameter which specifies the total capex of an existing technology which still has to be paid | technology |
| opex_specific_variable | set_time_steps_operation | Parameter which specifies the variable specific opex | technology |
| opex_specific_fixed | set_time_steps_yearly | Parameter which specifies the fixed annual specific opex | technology |
| lifetime | temporal immutable | Parameter which specifies the lifetime of a newly built technology | technology |
| construction_time | temporal immutable | Parameter which specifies the construction time of a newly built technology | technology |

| | | | |
|---|---|---|---|
| max_diffusion_rate | set_time_steps_yearly | Parameter which specifies the maximum diffusion rate which is the maximum increase in capacity between investment steps | technology |
| capacity_limit | temporal immutable | Parameter which specifies the capacity limit of technologies | technology |
| min_load | set_time_steps_operation | Parameter which specifies the minimum load of technology relative to installed capacity | technology |
| max_load | set_time_steps_operation | Parameter which specifies the maximum load of technology relative to installed capacity | technology |
| carbon_intensity_technology | temporal immutable | Parameter which specifies the carbon intensity of each technology | technology |
| capex_specific_conversion | set_time_steps_yearly | Parameter which specifies the slope of the capex if approximated linearly | conversion technology |
| conversion_factor | set_time_steps_yearly | Parameter which specifies the slope of the conversion efficiency if approximated linearly | conversion technology |
| time_steps_storage_level_duration | set_time_steps_storage_level | Parameter which specifies the time step duration in StorageLevel for all technologies | storage technology |
| efficiency_charge | set_time_steps_yearly | efficiency during charging for storage technologies | storage technology |
| efficiency_discharge | set_time_steps_yearly | efficiency during discharging for storage technologies | storage technology |
| self_discharge | temporal immutable | self-discharge of storage technologies | storage technology |
| capex_specific_storage | set_time_steps_yearly | specific capex of storage technologies | storage technology |
| distance | temporal immutable | distance between two nodes for transport technologies | transport technology |
| capex_specific_transport | set_time_steps_yearly | capex per unit for transport technologies | transport technology |
| capex_per_distance_transport | set_time_steps_yearly | capex per distance for transport technologies | transport technology |
| transport_loss_factor | temporal immutable | carrier losses due to transport with transport technologies | transport technology |

| Variables | | | |
|---|---|---|---|
| **Name:** | **Time Step Type:** | **Doc String:** | **Scope:** |
| carbon_emissions_annual | set_time_steps_yearly | annual carbon emissions of energy system | system |
| carbon_emissions_cumulative | set_time_steps_yearly | cumulative carbon emissions of energy system over time for each year | system |
| carbon_emissions_budget_overshoot | set_time_steps_yearly | overshoot carbon emissions of energy system at the end of the time horizon | system |
| carbon_emissions_annual_overshoot | set_time_steps_yearly | overshoot of the annual carbon emissions limit of energy system | system |
| cost_carbon_emissions_total | set_time_steps_yearly | total cost of carbon emissions of energy system | system |
| cost_total | set_time_steps_yearly | total cost of energy system | system |
| net_present_cost | set_time_steps_yearly | net_present_cost of energy system | system |
| flow_import | set_time_steps_operation | node- and time-dependent carrier import from the grid | carrier |
| flow_export | set_time_steps_operation | node- and time-dependent carrier export from the grid | carrier |
| cost_carrier | set_time_steps_operation | node- and time-dependent carrier cost due to import and export | carrier |
| cost_carrier_total | set_time_steps_yearly | total carrier cost due to import and export | carrier |
| carbon_emissions_carrier | set_time_steps_operation | carbon emissions of importing and exporting carrier | carrier |
| carbon_emissions_carrier_total | set_time_steps_yearly | total carbon emissions of importing and exporting carrier | carrier |
| shed_demand | set_time_steps_operation | shed demand of carrier | carrier |
| cost_shed_demand | set_time_steps_operation | shed demand of carrier | carrier |
| technology_installation | set_time_steps_yearly | installment of a technology at location l and time t | technology |
| capacity | set_time_steps_yearly | size of installed technology at location l and time t | technology |
| capacity_addition | set_time_steps_yearly | size of built technology (invested capacity after construction) at location l and time t | technology |
| capacity_investment | set_time_steps_yearly | size of invested technology at location l and time t | technology |
| cost_capex | set_time_steps_yearly | capex for building technology at location l and time t | technology |
| capex_yearly | set_time_steps_yearly | annual capex for having technology at location l | technology |

| | | total capex for installing all technologies in all locations at all | |
|---|---|---|---|
| cost_capex_total | set_time_steps_yearly | times | technology |
| cost_opex | set_time_steps_operation | opex for operating technology at location l and time t | technology |
| opex_yearly | set_time_steps_yearly | yearly opex for operating technology at location l and year y | technology |
| cost_opex_total | set_time_steps_yearly | total opex all technologies and locations in year y | technology |
| | | carbon emissions for operating technology at location l and | |
| carbon_emissions_technology | set_time_steps_operation | time t | technology |
| | | total carbon emissions for operating technology at location l | |
| carbon_emissions_technology_total | set_time_steps_yearly | and time t | technology |
| | | | conversion |
| flow_conversion_input | set_time_steps_operation | Carrier input of conversion technologies | technology |
| | | | conversion |
| flow_conversion_output | set_time_steps_operation | Carrier output of conversion technologies | technology |
| | | pwa variable for size of installed technology on edge i and time | |
| capacity_approximation | set_time_steps_yearly | t | technology |
| | | pwa variable for capex for installing technology on edge i and | |
| capex_approximation | set_time_steps_yearly | time t | technology |
| | | | conversion |
| flow_approximation_reference | set_time_steps_operation | pwa of flow of reference carrier of conversion technologies | technology |
| | | | conversion |
| flow_approximation_dependent | set_time_steps_operation | pwa of flow of dependent carriers of conversion technologies | technology |
| | | | storage |
| flow_storage_charge | set_time_steps_operation | carrier flow into storage technology on node i and time t | technology |
| | | | storage |
| flow_storage_discharge | set_time_steps_operation | carrier flow out of storage technology on node i and time t | technology |
| | set_time_steps_storage_lev | storage level of storage technology on node in each storage | storage |
| storage_level | el | time step | technology |
| | | | transport |
| flow_transport | set_time_steps_operation | carrier flow through transport technology on edge i and time t | technology |
| | | | transport |
| flow_transport_loss | set_time_steps_operation | carrier flow through transport technology on edge i and time t | technology |

| Constraints | | | | |
|---|---|---|---|---|
| **Name:** | **Time Step Type:** | **Doc String:** | | **Scope:** |
| constraint_carbon_emissions_annual | set_time_steps_yearly | total annual carbon emissions of energy system | | system |
| constraint_carbon_emissions_cumulative | set_time_steps_yearly | cumulative carbon emissions of energy system over time | | system |
| constraint_cost_carbon_emissions_total | set_time_steps_yearly | total carbon emissions cost of energy system | | system |
| constraint_carbon_emissions_annual_limit | set_time_steps_yearly | limit of total annual carbon emissions of energy system | | system |
| constraint_carbon_emissions_budget | set_time_steps_yearly | Budget of total carbon emissions of energy system | | system |
| constraint_carbon_emissions_budget_overshoot | set_time_steps_yearly | Disable carbon emissions budget overshoot if carbon emissions budget overshoot price is inf | | system |
| constraint_carbon_emissions_annual_overshoot | set_time_steps_yearly | Disable annual carbon emissions overshoot if annual carbon emissions overshoot price is inf | | system |
| constraint_carbon_emissions_overshoot_limit | set_time_steps_yearly | Limit of overshot carbon emissions of energy system | | system |
| constraint_cost_total | set_time_steps_yearly | total cost of energy system | | system |
| constraint_net_present_cost | set_time_steps_yearly | net_present_cost of energy system | | system |
| constraint_availability_import | set_time_steps_operation | node- and time-dependent carrier availability to import from outside the system boundaries | | carrier |
| constraint_availability_export | set_time_steps_operation | node- and time-dependent carrier availability to export to outside the system boundaries | | carrier |
| constraint_availability_import_yearly | set_time_steps_yearly | node- and time-dependent carrier availability to import from outside the system boundaries summed over entire year | | carrier |
| constraint_availability_export_yearly | set_time_steps_yearly | node- and time-dependent carrier availability to export to outside the system boundaries summed over entire year | | carrier |
| constraint_cost_carrier | set_time_steps_operation | cost of importing and exporting carrier | | carrier |
| constraint_cost_shed_demand | set_time_steps_operation | cost of shedding carrier demand | | carrier |
| constraint_limit_shed_demand | set_time_steps_operation | limit of shedding carrier demand | | carrier |
| constraint_cost_carrier_total | set_time_steps_yearly | total cost of importing and exporting carriers | | carrier |
| constraint_carbon_emissions_carrier | set_time_steps_operation | carbon emissions of importing and exporting carrier | | carrier |

| | | total carbon emissions of importing and exporting | |
|---|---|---|---|
| constraint_carbon_emissions_carrier_total | set_time_steps_yearly | carriers | carrier |
| constraint_nodal_energy_balance | set_time_steps_operation | node- and time-dependent energy balance for each carrier | carrier |
| constraint_technology_capacity_limit | set_time_steps_yearly | limited capacity of technology depending on loc and time | technology |
| constraint_technology_min_capacity | set_time_steps_yearly | min capacity of technology that can be installed | technology |
| constraint_technology_max_capacity | set_time_steps_yearly | max capacity of technology that can be installed | technology |
| constraint_technology_construction_time | set_time_steps_yearly | lead time in which invested technology is constructed | technology |
| constraint_technology_lifetime | set_time_steps_yearly | max capacity of technology that can be installed | technology |
| constraint_technology_diffusion_limit | set_time_steps_yearly | Limits the newly built capacity by the existing knowledge stock | technology |
| constraint_capacity_factor | set_time_steps_operation | limit max load by installed capacity | technology |
| constraint_capex_yearly | set_time_steps_yearly | annual capex of having capacity of technology. | technology |
| constraint_cost_capex_total | set_time_steps_yearly | total capex of all technology that can be installed. | technology |
| constraint_opex_technology | set_time_steps_operation | opex for each technology at each location and time step | technology |
| constraint_opex_yearly | set_time_steps_yearly | total opex of all technology that are operated. | technology |
| constraint_cost_opex_total | set_time_steps_yearly | total opex of all technology that are operated. | technology |
| constraint_carbon_emissions_technology | set_time_steps_operation | carbon emissions for each technology at each location and time step | technology |
| constraint_carbon_emissions_technology_total | set_time_steps_yearly | total carbon emissions for each technology at each location and time step | technology |
| disjunct_on_technology | set_time_steps_operation | disjunct to indicate that technology is on | technology |
| disjunct_off_technology | set_time_steps_operation | disjunct to indicate that technology is off | technology |
| disjunction_decision_on_off_technology | set_time_steps_operation | disjunction to link the on off disjuncts | technology |
| constraint_linear_capex | set_time_steps_yearly | Linear relationship in capex | technology |
| constraint_linear_conversion_factor | set_time_steps_operation | Linear relationship in conversion_factor | conversion technology |
| constraint_capex_coupling | set_time_steps_yearly | couples the real capex variables with the approximated variables | conversion technology |

| | | couples the real capacity variables with the approximated variables | conversion technology |
|---|---|---|---|
| constraint_capacity_coupling | set_time_steps_yearly | | |
| constraint_reference_flow_coupling | set_time_steps_operation | couples the real reference flow variables with the approximated variables | conversion technology |
| constraint_dependent_flow_coupling | set_time_steps_operation | couples the real dependent flow variables with the approximated variables | conversion technology |
| constraint_storage_level_max | set_time_steps_storage_level | limit maximum storage level to capacity | storage technology |
| constraint_couple_storage_level | set_time_steps_storage_level | couple subsequent storage levels (time coupling constraints) | storage technology |
| constraint_storage_technology_capex | set_time_steps_yearly | Capital expenditures for installing storage technology | storage technology |
| constraint_transport_technology_losses_flow | set_time_steps_operation | Carrier loss due to transport with through transport technology | transport technology |
| constraint_transport_technology_capex | set_time_steps_yearly | Capital expenditures for installing transport technology | transport technology |
| constraint_transport_technology_bidirectional | set_time_steps_yearly | Forces that transport technology capacity must be equal in both directions | transport technology |