Open and Compact Model for the German Energy Transition (OCGModel)

Julia Barbosa, Christopher Ripp and Florian Steinke

Jun 02, 2021

CONTENTS:

CHAPTER

ONE

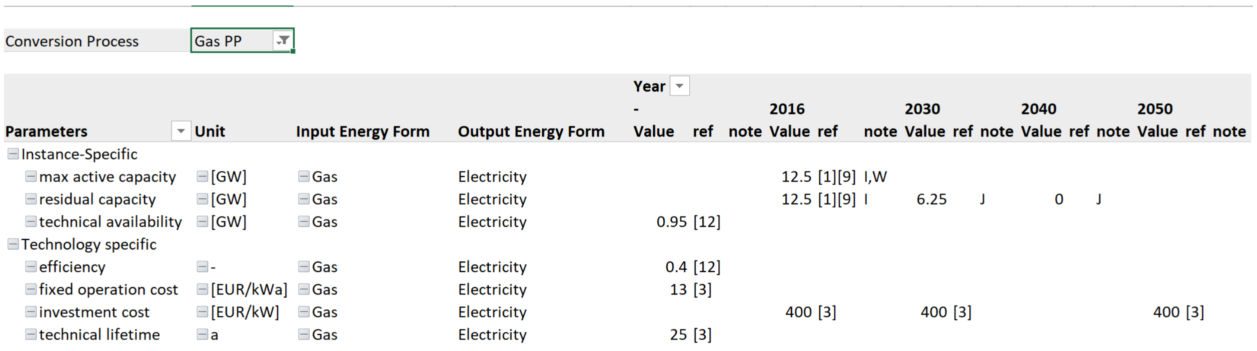
DATABASE

The OCGModel parametrization was perfomed using only public available data.

The complete database, including references and comments, can be found on the excel file under the folder **database**.

The file is structured as follows:

- At the sheet *Summary* we present a pivot table where the parameters of the selected conversion process can be easily visualized.



Conversion Process | Gas PP

				Year														
				-			2016			2030			2040			2050		
Parameters	Unit	Input Energy Form	Output Energy Form	Value	ref	note	Value	ref	note	Value	ref	note	Value	ref	note	Value	ref	note
Instance-Specific																		
max active capacity	[GW]	Gas	Electricity				12.5	[1][9]	I,W									
residual capacity	[GW]	Gas	Electricity				12.5	[1][9]	I	6.25		J	0		J			
technical availability	[GW]	Gas	Electricity	0.95	[12]													
Technology specific																		
efficiency	-	Gas	Electricity	0.4	[12]													
fixed operation cost	[EUR/kWa]	Gas	Electricity	13	[3]													
investment cost	[EUR/kW]	Gas	Electricity				400	[3]		400	[3]					400	[3]	
technical lifetime	a	Gas	Electricity	25	[3]													

Summary sheet of database file. Data is displayed according to the selected conversion process.

- At the sheet *Data* you can find the data in a computer-readable format.
- At the sheet *Timeseries* we name all timeseries used in the model, including type, short description, reference and name of the respective file. The time series files can be found under the folder **database/timeseries**
- At the sheet *References* we name all the references and provide the respective links to them.

Important: You can find more information about the model structre on the Original Paper.

CHAPTER

TWO

OSEMOSYS IMPLEMENTATION

The OCG Model is implemented using the Open Source Energy Modelling System (OSeMOSYS) through the int4osemosys interface. See *Osemosys Interface Documentation*.

Note: Learn more about OSeMOSYS at http://www.osemosys.org

2.1 Installation

1. **Download the OCG Model Repository**

   The OCG Model can be downloaded from our GitHub repository

2. **Install** GNU Linear Programming Kit.

   At Windows:

   - First download GLPK lattest version at https://sourceforge.net/projects/winglpk/
   - Unzip the download file *winglpk-4.XX* and locate the folder *glpk-4.XX* inside.
   - Copy the whole folder *glpk-4.XX* to your "C:" directory.
   - Navigate to *Control Panel* > *System* > *About* and check the system type (x32 or x64).
   - Type Win+r, type *systempropertiesadvanced* and press ok.
   - At the new window, select "Environmental variables" below right
   - Under the System variables, select "Path" and then click "Edit"
   - Select "New" and add the path to GLPK. Be aware to use the version corresponding to your system type, i.e. "C:glpk-4.**XX**w32" for 32-Bit system or "C:glpk-4.**XX**w64" for 64-Bit system.

     Test your installation by running the following line at the terminal:

     ```
     $ glpsol --version
     ```

   At Linux the following call should install GLPK:

   ```
   $ sudo apt-get install GLPK
   ```

3. **Install Python (with pip)**

   If you do not have Python installed in your computer, you can download it here. Make sure to check **"Add Python to PATH"** at the installation assistant. If you opt for the customizable installation make sure to include pip.

4. **Install required python packages**

   At Windows:

   Run as *administrator* the ***setup.bat*** file in the int4osemosys directory.

   At Linux:

   Be sure open the terminal in the int4osemosys directory and the following call should install the packages:

   ```
   $ pip install -r core/requirements.txt
   ```

5. (optional) Install CPLEX API

   If you have acess to CPLEX, the interface also offers support to the CPLEX solver. For that you need to install the CPLEX Python API and change the interface settings to CPLEX. See how to change the settings *here*.

2.2 Run the model

To start the interface:

- On Windows, you can simply run the **start_int4osemosys.bat** file.
- Or **Run the startOsemosysInt.py** script in this folder. This can be done using the prefered IDE or by the following command:

  ```
  $ python startOsemosysInt.py
  ```

The Following menu will appear:

```
...reading Settings from settings.ini
####Interface for Osemosys####
Select option:
r - Run model
p - Plot results
s - Settings
q - Quit
>> r
```

Type **"r"** to run the model. You will then be asked for the model name as scenario.

```
Model name: OCGModel
Scenario: base
```

If the model and scenario name were given correctly, the solver log will be displayed.

2.3 Plot Results

You can use the same interface to plot the model resuls. For that:

1. Select the option "p" - Plot results at the main menu.
2. A list with all simulations in the runs folder will be displayed and can be selected.
3. Give the additional information for the type of plot selected. e.g. Fuel, year,… **The paramerets must be given exactly how declared in the Techmap file!! CASE SENSITIVE!**

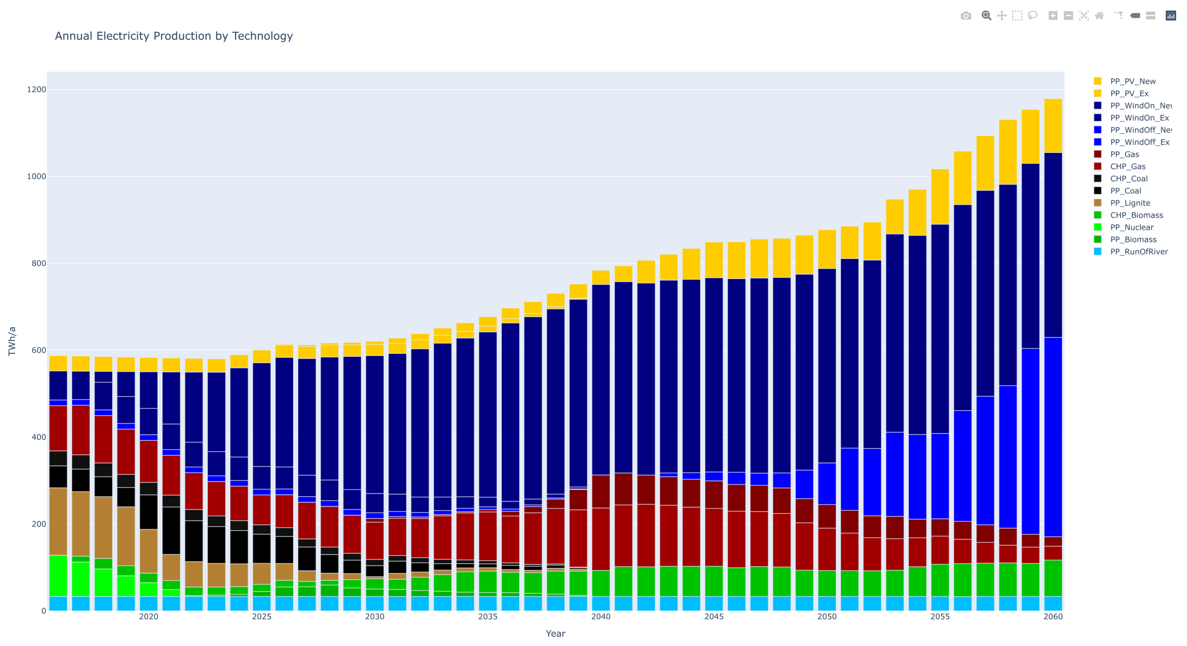
Example:

```
Select Plot:
0 - plot_annual_active_capacities
1 - plot_annual_emissions
2 - plot_annual_supply
3 - plot_annual_use
4 - plot_sankey
5 - plot_supply_timeseries
6 - plot_use_timeseries
>> 2
Please give the inputs for plot_annual_supply. [help: (fuel)]
fuel: Electricity
```

Note: See the *plotting engine documentation* for a short description of each plot.

---

4. The plot will be displayed at your internet browser. For the example above, the following image is expected:



Important: You can also find the CSV files with the simulation results at **". run OCGModel_base"**

2.4 Edit Model

The model instance parameters are located at the Technology Map **OCGModel.xlsx** at the folder **data**. To edit the model, simply open the file with the model name and edit it there. You can change existing parameters, create new scenarios, add fuels and technologies to the model. After editing save the file and run the model again.

To create a new model, you can duplicate the OCGModel technoly map file. Then, rename the file with the desired model name and edit it. To run the new model, use the same name given to the technology map.

Important: *time-slice dependent paramerets*: These parameters are given in the Technology map file by the name of the profile. Be sure to have at the **datatimeseries** folder, a .txt file with the profile name. The file should contain 8760 entries. See the examples provided in the folder.

Important: *Time Setting*: The time setting will affect how the time profiles files are interpreted. You can use two different types of time settings: **mean** and **selection**.

For a **selection** time setting, a .csv file is required at the timeseries folder only with the selected timesteps, the profiles are obtained from the .txt files.

For a **mean** time setting, a .csv file with the time setting name is also required. This file should contain not only the time-slices names, but also one column for each profile used in the model.

By using the *script* made availble for selecting the timeslices, the output is alredy in the required format.

2.5 Change Settings

You can change some settings by typing “s” on the main interface menu. These are:

- ***Osemosys code version*:** Version of the OSeMOSYS to be used to generate the model instance.

  Defaults to our adapted version (osemosys_short_OCG.txt) at the OSeMOSYS directory. All adpations to the original version are docummented in the file.

  You can download the other versions of OSeMOSYS, add to the the OSeMOSYS directory and change this setting to build the model instance with them.

- ***solver*:** Solver to be used. The interface offers support for the open-sorce GLPK and also CPLEX.

  If you have access to CPLEX and wish to use it, don’t forget to install the cplex python API before changing this configuration.

- ***compute missing variables*:** This parameter tell then if the not computed variables should be calcuated.

  The OSeMOSYS short code version does not compute intermediate variables, e.g ProductionByTechnology, UseBytechnology.

  If True (1-default), the necessary variables will be computed on demand by the plotting engine. (takes longer time to initilize the plotting engine).

  If False (0), trying to plot plots that required a non-computed variable will raise an error.

- ***sankey opacity*:** Link opacity on the Sankey plot.

(NOT RECOMMENDED) You can also change the settings directlly on the **settings.ini** file located at the **core** directory.

CHAPTER

THREE

INT4OSEMOSYS DOCUMENTATION

This is the code documentation. For information about how to use the interface see *Osemosys Implementation*.

3.1 Core Modules

3.1.1 Running Engine

Module to run the simulation.

(1) Calls Input Processing to build the input file model from the map.

(2) Creates simulation folder in the runs directory

(3) Calls solvers according to configuration file

(4) Save output in csv format. If Solver is CPLEX then the CPLEXSolutionProcessor is called to save selected variables to .csv

Example

```
>>> from core.ModelRunner import ModelRunner
>>> from core.utils import settings
```

Build engine and run:

```
>>> run_engine = ModelRunner(model="Germany",scenario="base", sett=settings)
>>> run_engine.run()
```

or

```
>>> ModelRunner.build_and_run(model="Germany",scenario="base", sett=settings)
```

class `core.ModelRunner.`**`ModelRunner`**(*model*, *scenario*, *sett:* core.Settings.Settings)

Bases: `object`

Model Runner

Parameters

- **`model`** (*`str`*) – Model name
- **`scenario`** (*`str`*) – Scenario name
- **`sett`** (`Settings`) – Settings object with the simulation settings

`classmethod build_and_run`(*model*, *scenario*, *sett: core.Settings.Settings*)

Initialize the runner engine and call the run method.

Parameters

- **model** (*str*) – Model name
- **scenario** (*str*) – Scenario name
- **sett** (Settings) – Settings object with the simulation settings.

Returns Model runner engine

Return type *ModelRunner*

`run()`

Runs the model.

Performs all steps necessary to run the model. (1) Calls Input Processing to build the input file model from the map. (2) Creates simulation folder in the runs directory (3) Calls solvers according to configuration file (4) Save output in csv format. If Solver is CPLEX then the CPLEXSolutionProcessor is called to save selected variables to .csv

Raises ***ModelRunnerException*** – Invalid input file format or generation of CPLEX input file failed.

`exception core.ModelRunner.ModelRunnerException`

Bases: `Exception`

Class for throwing Exceptions during the simulation running

3.1.2 Plotting Engine

Module for plotting model results.

Generates HTML figures produced using plotly library.

`class core.ResPlotting.OsemosysPlottingEngine`(*sim_dir*, *compute_missing_variables=False*)

Bases: `object`

Plotting Engine.

Parameters

- **sim_dir** (*str*) – Simulation directory generated by the Model Runner engine
- **compute_missing_variables** (*bool, optional*) – Compute missing intermediate variables. Default False. If True and OsemosysModel object will be created from the input file in the simulation folder. This will make the object initialization slower.

Raises ***ResPlottingException*** – Error during the execution of a plot method.

`plot_annual_active_capacities()`

Plot annual active capacities

Bar plot active capacities by year. Stacked by technology.

`plot_annual_emissions()`

Plot annual emissions.

Bar plot of the emissions by year, stacked by emission type.

plot_annual_supply(*fuel*)

Plot annual energy supply of a fuel.

Bar plot of the annual energy supply of a given fuel by year. Stacked by technology.

Parameters **fuel** (*str*) – Name of fuel as defined in the model. CASE SENSITIVE.

plot_annual_use(*fuel*)

Plot annual energy consumption of a fuel.

Bar plot of the annual energy consumption of a given fuel by year. Stacked by technology. Usable energy demand not included!!

Parameters **fuel** (*str*) – Name of fuel as defined in the model. CASE SENSITIVE.

plot_sankey(*year*)

Plot sankey diagram of a year.

Parameters **year** (*str*) – Year to be plotted

plot_supply_timeseries(*fuel*, *year*)

Plot the timeslice depended supply of a fuel in a year.

Area plot of the supply of a fuel by the timeslices, stacked by technology. The values are corrected with the timeslices weights to represent the actual power output of the process.

Parameters

- **fuel** (*str*) – Name of fuel as defined in the model. CASE SENSITIVE.
- **year** (*str*) – Year to be plotted

plot_use_timeseries(*fuel*, *year*)

Plot the timeslice depended consumption of a fuel in a year.

Area plot of the consumption of a fuel by the timeslices, stacked by technology. The demand is also included in the plot. The values are corrected with the timeslices weights to represent the actual power output of the process.

Parameters

- **fuel** (*str*) – Name of fuel as defined in the model. CASE SENSITIVE.
- **year** (*str*) – Year to be plotted

exception core.ResPlotting.**ResPlottingException**

Bases: `Exception`

Class for throwing Exceptions by the plotting engine

3.1.3 Input Processing Engine

Tools to construct an Osemosys Model from the excel model map or from an previous input file

exception core.InputProcessing.**MapProcessingException**

Bases: `Exception`

Class for throwing Exceptions during the map processing

class core.InputProcessing.**OsemosysInputGenerator**(*name: str*, *scenario: str*, *data_dir: str*, *sim_dir: str*)

Bases: `object`

OSeMOSYS Input Generator

Initialize Osemosys Model by creating empty sets and parameters.

Parameters

- **`name`** (*`str`*) – Model name
- **`scenario`** (*`str`*) – Scenario name
- **`data_dir`** (*`str`*) – Path to Map file
- **`sim_dir`** (*`str`*) – Path to simulation dir

`sets`

Model defined parameters indexed by parameter name.

Type dict[str, *OsemosysSet*]

`params`

Model defined sets indexed by set name.

Type dict[str,*OsemosysParam*]

`classmethod build_from_input_file`(*sim_dir*, *filename='input.txt'*)

Construct model from an input file. Useful for model post processing.

Parameters

- **`sim_dir`** – Path to simulation dir. Model name and scenario read from simulation dir name.
- **`filename`** (*`str, optional`*) – Input filename. Default "input.txt"

Returns Model with all parameters and set defined.

Return type *OsemosysInputGenerator*

`classmethod build_load_map_and_write`(*name: str*, *scenario: str*, *data_dir: str*, *sim_dir: str*)

Construct Model, load map and write input file.

Parameters

- **`name`** (*`str`*) – Model name
- **`scenario`** (*`str`*) – Scenario name
- **`data_dir`** (*`str`*) – Path to Map file
- **`sim_dir`** (*`str`*) – Path to simulation dir

Returns Model with all parameters and set defined.

Return type *OsemosysInputGenerator*

`property emissions_params: Dict[str, core.InputProcessing.OsemosysParam]`

All parameters of "Emissions" group

`property param_names: List[str]`

Name of all Parameters

`read_model_map`()

Populate the model with the excel model map parameters values.

Raises *`MapProcessingException`* – The excel file with the model parameters could not be found. The file must have the same name as the model and be located at the data dir.

`property set_names: List[str]`

Name of all Sets

`property temporal_params: Dict[str, core.InputProcessing.OsemosysParam]`
All parameters of that are "Temporal"

`write_model`(*filename='input.txt'*)
Write model to osemosys input file at the simulation directory.

Parameters **`filename`** (*`str, optional`*) – Filename. Default "input.txt"

`class core.InputProcessing.OsemosysParam`(*name: str*, *set_list: List[core.InputProcessing.OsemosysSet]*, *scaling_factor: float = 1*, *default: Optional[float] = None*, *group: Optional[str] = None*, *type=None*, *normalize=None*)

Bases: `object`

Class for constructing Osemosys Parameters

Parameters

- **`name`** (*`str`*) – Param name
- **`set_list`** (*`List[OsemosysSet]`*) – List with the sets that index this parameter.
- **`scaling_factor`** (*`float, optional`*) – Scaling factor. Default 1.
- **`default`** (*`float, optional`*) – Default value of parameter. Default None.
- **`group`** (*`str, optional`*) – Identifier of which groups this parameter belongs. Default None.
- **`type`** (*`str, optional`*) – Parameter type. Default None. Possible types: "Temporal"
- **`normalize`** (*`bool, optional`*) – Indicator weather the values should be normalized to sum 1. Default None. Necessary for adjusting the parameter values to the model constraints, e.g. demand time profiles.

`set_names`
List with the same of the sets that index this parameter.

Type List[str]

`add_value`(*key: List[str]*, *val: float*)
Add value to parameter list

Parameters

- **`key`** (*`Iterable[str]`*) – Parameter index.
- **`val`** (*`float`*) – Parameter Value

`write`(*f*)
Write parameter to OSEMOSYS input file.

Parameters **`f`** (*`file`*) – File stream.

`class core.InputProcessing.OsemosysSet`(*name: str*, *dim: int*, *elements: Optional[Iterable] = None*, *numeric=False*)

Bases: `object`

Class for constructing Osemosys sets

Parameters

- **`name`** (*`str`*) – Set Name
- **`dim`** (*`int`*) – Set dimension
- **`elements`** (*`Iterable, optional`*) – Elements to be added to set. Default None.
- **`numeric`** (*`bool, optional`*) – Indicates weather the set is numeric. Default False.

`append`(*elem*)
Add new element to the set.

Parameters **`elem`** – Element or list of elements to be added to set.

`write`(*f*)
Write set to OSEMOSYS input file.

Parameters **`f`** (*`file`*) – File stream.

3.1.4 Output Processing Engine

Module for post processing

`exception core.OutputProcessing.`**`CPLEXSolutionProcessingException`**
Bases: *`core.OutputProcessing.OutputProcessingException`*

Class for throwing Exceptions during post processing with CPLEX python API

`class core.OutputProcessing.`**`CPLEXSolutionProcessor`**(*c: False*, *model_generator=None*)
Bases: `object`

Post Processing Engine for models solved with CPLEX.

Parameters **`c`** (*`cplex.Cplex`*) – Cplex object with the solved model

Raises ***`CPLEXSolutionProcessingException`*** – Error while post processing with CPLEX API

`main_vars`
Dictionary with variable name as key and respective indexing sets as list, e.g. [“FUEL”, “YEAR”]. “Main” Variables are the ones present at all OSEMOSYS code versions and that cannot be computed from other variables.

Type dict[str,list[str]]

`save_main_vars_to_csv`(*save_folder*)
Save variables to csv.

Each variable is saved to a different csv file named after the variable name.

Parameters **`save_folder`** (*`str`*) – Folder where the csv files will be saved

Returns:

`exception core.OutputProcessing.`**`OutputProcessingException`**
Bases: `Exception`

Class for throwing Exceptions during post processing

`class core.OutputProcessing.`**`SolutionInterface`**(*sol_dir*)
Bases: `object`

Class for post processing of the model solution

Parameters **`sol_dir`** (*`str`*) – Directory where the solution is found.

Raises ***`OutputProcessingException`*** – Error during results post processing

`compute_all_variables`(*input_generator*)
Compute intermediate variables from main variables.

Compute intermediate variables from main by using the model parameters set in the the input_generator object. As not all Osemosys code version contains the intermediate variables, this methods computes them and also add the respective csv file to the solution directory. The computed variables are:

```
>>> vars_to_compute = ["AnnualEmissions", "TotalTechnologyAnnualActivity",
↪"UseByTechnology",
>>>                     "UseByTechnologyAnnual", "ProductionByTechnologyAnnual",
↪"ProductionByTechnology",
>>>                    "Demand",  "RateOfUseByTechnology",
↪"RateOfProductionByTechnology", "RateOfDemand",
>>>                    "TotalCapacityAnnual" ]
```

Parameters **input_generator** (`OsemosysInputGenerator`) – OSEMOSYS Model with all parameters values

compute_variable(*varname*, *input_generator:* core.InputProcessing.OsemosysInputGenerator)

Compute intermediate variable from main variables

As not all Osemosys code version contains the intermediate variables, this methods computes the varibale with name "varname" and also add the respective csv file to the solution directory.

Parameters

- **varname** (*str*) – Variable name to be computed
- **input_generator** (`OsemosysInputGenerator`) – OSEMOSYS Model with all parameters values

get_var_values(*var*, *select=None*, *return_zeros=False*, *compute_missing_vars=False*, *input_generator: Optional[*core.InputProcessing.OsemosysInputGenerator*] = None*) → pandas.core.frame.DataFrame

Get the value of a variable.

Parameters

- **var** (*str*) – Variable name
- **select** (*dict[str,str], optional*) – Select index of variables to be returned. Dictionary where the keys are the set names and the entries the index to be selected. Defaults to None.
- **return_zeros** (*bool, optional*) – Indicator weather variables with zero as values should be selected. Defaults to False.
- **compute_missing_vars** (*bool, optional*) – If true and the variable ".csv" is not found the method compute_variable is called. Requires input_generator to be defined. Defaults to False
- **input_generator** (`OsemosysInputGenerator`*, optional*) – OSEMOSYS Model with all parameters values. Defaults to None. If compute_missing_vars is True, this must be given otherwise an error is raised.

Returns Dataframe with the indexes and variable values.

Return type pandas.DataFrame

3.1.5 Settings Module

Concatenates definitions that are shared by the different modules and external configurations

`core.Settings.`**`cplex`**

If the cplex API is found by the python, than cplex is imported else this a attribute is set as False.

Type cplex or False

`core.Settings.`**`sankey_opacity`**

Opacity of links in the sankey diagram plot.

Type float

`core.Settings.`**`Y`**

“YEAR”

`core.Settings.`**`T`**

“TECHNOLOGY”

`core.Settings.`**`L`**

“TIMESLICE”

`core.Settings.`**`F`**

“FUEL”

`core.Settings.`**`E`**

“EMISSION”

`core.Settings.`**`M`**

“MODE_OF_OPERATION”

`core.Settings.`**`R`**

“REGION”

`core.Settings.`**`LS`**

“SEASON”

`core.Settings.`**`LD`**

“DAYTYPE”

`core.Settings.`**`LH`**

“DAILYTIMEBRACKET”

`core.Settings.`**`S`**

“STORAGE”

`class` `core.Settings.`**`Settings`**(*settings_file*)

Bases: `object`

Settings object

Read model settings.ini file and summarize important information to other modules.

Parameters **`settings_file`** (*`str`*) – .ini file path

`update`()

Update settings file

`exception` `core.Settings.`**`SettingsException`**

Bases: `Exception`

3.2 Scripts

3.2.1 Time slices selection

scripts.timeslicesSelection.**time_slices_selection**(*tssname*, *timeseries_dir*, *ts_names*, *n_clusters=12*, *n_far_points=1*, *n_global_far_points=1*)

Select time slices based on k-means and prepare time configuration file.

Apply k-means algorithm for clustering the 8760 hours of a year into **n_cluesters** different groups. Each hour is distinguished the vales of the different timeseries (in **ts_names**) files which can be found at the **timeseries_dir**. To the cluster centers selection is added:

(1) **n_far_points** with the highest overall distance to their own center are added to the time slice selection.

(2) **n_global_far_points** with the highest overall all centers are also selected as a time slices.

Making a total of **n_clusters + n_clusters*n_far_points + n_global_far_points** time slices selected.

The time slice weight parameter is set according to the number hours represented by the time steps (i.e. 1 for the “far points” and the number of elements in the cluster for the centers). The obtained time slices and time distributions are wirtten to an **tssname**.csv file at the **timeseries_dir**.

Example

```
>>> from scripts import timeslicesSelection as tss
>>> import os # relative path
```

Set path and give timeseries names to be used

```
>>> # Set use the timeseries dir inside the data dir
>>> timeseries_dir = os.sep.join(["..", "data", "timeseries"])
>>> # Select timeseries to be considered for clustering (must be on the␣
↪aforementioned folder)
>>> ts_names = ["corrected_eletricity_demand_2016.txt",
>>>             "heat_household_demand_2016.txt",
>>>             "Solar_availability_1004FLH.txt",
>>>             "Solar_availability_2016.txt",
>>>             "WindOffshore_availability_2016.txt",
>>>             "WindOffshore_availability_4300FLH.txt",
>>>             "WindOnshore_availability_2016.txt",
>>>             "WindOnshore_availability_2700FLH.txt",
>>>             "Uniform.txt"]
```

Run k-means

```
>>> tss.time_slices_selection("tssname", timeseries_dir, ts_names)
```

Parameters

- **tssname** (*str*) – Name for the timeslice selection
- **timeseries_dir** (*str*) – Directory where the time series files are. Files must be “.txt” with 8760 entries separated by space. See examples in dir.
- **ts_names** (*list[str]*) – List with the name of the timeseries to be considered for the k-means clustering.

- **n_clusters** (*int*) – n of clusters for the k-means algorithm.
- **n_far_points** (*int*) – n furthest points from its own cluster center to be considered as individual timeslices.
- **n_global_far_points** (*int*) – n furthest points from all clusters to be considered as individual timeslices.

Raises `ValueError` – File from timeseries name given in **ts_names** not found in **timeseries_dir**

3.2.2 Rescale Availability from VREs

`scripts.RescaleRE.`**`rescale_re_flh`**(*ts_filename*, *flh*, *save=False*)

Rescale VREs availability to meet a given full load hours.

Rescale the availability values to meet the given full load hours **flh**, by modifying intermediate values while keeping minimum and maximum values fixed.

Parameters

- **ts_filename** (*str*) – Path to timeseries text file. The availbility timeseries must have 8760 entries, i.e. one for each your of the year.
- **flh** (*float*) – The full load hours the availability time series must be rescaled to.
- **save** (*bool, optional*) – Indicator if the new availbility timeseries should be save. Defaults to False. If True, the new time series is saved at the save path of the **ts_filename** with "_**flh** FLH" added to the name.

Returns List with len 8760 with the rescaled availability factors.

Return type list

PYTHON MODULE INDEX