
SecMOD MILP Documentation

Release 1.0

**Niklas Nolzen, Christiane Reinert,
Julia Frohmann, Dominik Tillmanns,
André Bardow**

Feb 24, 2023

CONTENTS:

1 Quickstart	3
1.1 Set up a working directory	3
2 Installation	5
2.1 Editable Installation	5
2.2 Package Installation	6
3 Data in SecMOD	7
3.1 Input data	7
3.2 Units in SecMOD	7
3.3 Output data	7
4 Developer reference	9
4.1 secmod.classes module	9
4.2 secmod.data_preprocessing module	24
4.3 secmod.data_processing module	25
4.4 secmod.evaluation module	25
4.5 secmod.helpers module	27
4.6 secmod.optimization_MILP module	28
4.7 secmod.setup module	37
5 Troubleshooting	39
6 Indices and tables	41
Python Module Index	43
Index	45

SecMOD MILP provides an open-source framework for optimizing the design and operation of industrial energy systems. The framework, including the entire equations and all features, is published in an open-source [Git Repository](#).

This tutorial explains how to install and use the SecMOD MILP extension of the SecMOD framework. More information and explanations can be found in the following peer-reviewed [publication](#):

Note: Nolzen, N., Reinert, C., Frohmann, J., Tillmanns, D., Bardow, A. (2023): “Design of low-carbon multi-energy systems in the SecMOD framework by combining MILP optimization and life-cycle assessment”. *Computers & Chemical Engineering*. 10.1016/j.compchemeng.2023.108176.

For further descriptions please refer to the [SecMOD LP framework](#) described in the following [publication](#):

Note: Reinert, C.; Schellhas, L.; Mannhardt, J.; Shu, D.; Kämper, A.; Baumgärtner, N.; Deutz, S., and Bardow, A. (2022): “SecMOD: An open-source modular framework combining multi-sector system optimization and life-cycle assessment”. *Frontiers in Energy Research*. DOI: 10.3389/fenrg.2022.884525.

QUICKSTART

In order to improve the distributability of SecMOD the package code is separated from the handled input data. This allows to use a single installation of SecMOD for several projects or scenarios in multiple working directories.

To get started with SecMOD, the following two steps must be completed:

1. *Installation of SecMOD*
2. *Set up a working directory*

For the first step follow the instructions for the *installation of SecMOD*. Afterwards return to *Step 2* to set up your first working directory.

1.1 Set up a working directory

Create a new directory where you want. Note that this directory will contain all input and results data and can become large in size, depending on the size of your examined model.

1. Open a terminal with an activated Python environment (e.g. Anaconda prompt)!
2. Navigate to your newly created directory in the terminal (e.g. “cd D:/WorkingDirectory”)!
3. Run the command “python -m secmod.setup” in the terminal!

SecMOD will now create the necessary folders for the input data and furthermore copy the sample data provided in the SecMOD repository. Afterwards all necessary data to run SecMOD is available in the working directory. Furthermore, a file called “start.bat” is created at the top level of your working directory. This file allows you to start SecMOD with a simple double click. Depending on your Python installation, you might need to edit the file (right click > edit) according to the instructions.

If you encounter problems during the automatic setup, please refer to *Troubleshooting*.

INSTALLATION

The installation of SecMOD is easily done. If you want to edit its source code, you need to follow *Editable Installation*. If you just want to use the published package, follow *Package Installation*.

2.1 Editable Installation

To edit the source code and therefore contribute to the development of SecMOD, you need to clone the repository to your local machine **recursively**. Using SSH as authentication to the git server the following command needs to be used:

```
git clone --recurse-submodules -j8 git@git-ce.rwth-aachen.de:ltt/secmod-milp.git 01-  
↔SecMOD
```

Note: If you don't know yet how to clone a repository have a look [here](#). Make sure that you have Python already installed on your PC.

After the clone process is completed, open a python terminal e.g. in your IDE (Visual Studio Code, Spyder, ...) or the Anaconda prompt. Afterwards you can install SecMOD in form of your local repository by using the following install command:

```
pip install --user -e PATH
```

Note: The PATH at the end of the command stands for the directory of your local repository. You can either use an absolute path or a path relative to the currently active directory of the terminal.

Your local repository should now be installed, if no errors occurred during the process. You can now proceed to the *quickstart* guide to complete your setup.

2.2 Package Installation

Error: SecMOD is not yet packaged and therefore not available through PyPI. Use the *Editable Installation* instead.

If you don't need to edit the source code of the SecMOD package, but just want to use the modules, you can install the package directly from PyPi using the following command:

```
pip install secmod
```

SecMOD should now be installed, if no errors occurred during the process. You can now proceed to the *quickstart* guide to complete your setup.

DATA IN SECMOD

Please refer to our publication for a detailed description of required data:

Note: Reinert, C., Nolzen, N., Frohmann, J., Tillmanns, D., Bardow, A. (2023): “Design of low-carbon multi-energy systems in the SecMOD framework by combining MILP optimization and life-cycle assessment”. *Computers & Chemical Engineering*. DOI: 10.1016/j.compchemeng.2023.108176.

3.1 Input data

Input data is used to model a given multi-sector system with the desired spatial, technical and temporal resolution. Therein, the model entails continuous sizing of discrete components and detailed component behavior, such as minimal part-load, load-dependent efficiencies. Input data can be added manually in the `sampledata` folder or in some cases be obtained automatically from open webpages, such as the open power system database by the scripts in `sampledata/00-EXTERNAL`.

3.2 Units in SecMOD

SecMOD employs unitizing to automatically convert units. All units used in the model must be defined in the sample data file.

3.3 Output data

The objective of the optimization is to determine an optimal design and operation of the multi-energy system according to the given objective and constraints. The output data of the optimization is saved in the working directory in `01-MODEL-RESULTS/InvestmentModel_year.pickle`. Additionally, the optimization variables stated in `config.py` can be written out. These variables are stored as `.csv`-files in the working directory in `01-MODEL-RESULTS/Extracted Results`. Usually, the evaluation will start automatically. To start the evaluation without the optimization, type:

```
python -m secmod.evaluation
```

This will open a graphical user interface (GUI). In the user interface, the capacity, product flows, and impacts can be shown for all products in different plot types. Clicking on a capacity will open a second layer to investigate the construction years. Further, all raw results are shown as a table next to the plot. The data can be exported in several formats including `tikz`, `png`, `xlsx` and `pdf`.

DEVELOPER REFERENCE

4.1 secmod.classes module

```
class secmod.classes.Grid(grid_path: Optional[pathlib.Path] = None, name: Optional[str] = None, nodes:
    Optional[pandas.core.frame.DataFrame] = None, connections:
    Optional[pandas.core.frame.DataFrame] = None)
```

Bases: object

This class is used to manage grids of nodes and connections.

This class represents a grid, which consists of a name, nodes and connections between these nodes. It can be created from scratch by providing at least a unique name and optionally pandas dataframes for data about the nodes and connections. It can alternatively be created from a path to a directory of an existing grid. The name will then be taken from the name of the grid directory and the node data and connection data will be loaded from .csv-files.

Furthermore the class has a static list of all created instances and a single static variable which represents the selected grid for the optimization.

Parameters

- **grid_path** (*Path*) – A path of the directory of an existing grid definition, which includes a CSV-file with data about nodes and connections in the grid.
- **name** (*str*) – Name for a new grid, if no `grid_path` is provided. The name has to be unique, since no two grids can have the same name.
- **nodes** (*pd.DataFrame*) – A pandas dataframe which has all the information about the nodes of a new grid, if no `grid_path` is provided. This dataframe must have the columns ["node", "latitude", "longitude"].
- **connections** (*pd.DataFrame*) – A pandas dataframe which has all the information about the connections of a new grid, if no `grid_path` is provided. This dataframe must have the columns ["connection", "node1", "node2"].

grids

Static list of all instanciated grids.

Type list

_selected

Static reference to a single Grid instance which is selected to be used in the optimization. Can be reached via the `selected()`-method.

Type *Grid*

Raises

- **ValueError** – If no name is provided, if no path is provided as well.
- **FileNotFoundError** – If “nodes.csv” or “connections.csv” does not exist in the provided directory.
- **NotADirectoryError** – If the provided path is no directory.

`CONNECTION_NODES = {None: ['node1', 'node2']}`

`grids = []`

`_selected = None`

classmethod `load_grids_from_directory`(*grids_category_path: pathlib.Path*)

Loads all grids of a specific class into their class.

property name

This property contains a unique name for the grid.

Getter: Gets the value of this property.

Returns: Name of the grid as string.

Setter: Sets the property to the provided name, if it isn’t already used by another grid.

Raises: ValueError: If the name is already used by another grid.

Type: str

property nodes

This property contains all information about the nodes of a grid.

Getter: Gets the value of this property.

Returns: Dataframe with the information about the nodes of the grid.

Setter: Sets the property to the provided dataframe, if the dataframe has the columns [“node”, “latitude”, “longitude”].

Type: pd.dataframe

static `get_list_of_node_ids()`

Return a list of all node ID numbers of the selected grid.

property connections

This property contains all information about the connections between nodes of a grid.

Getter: Gets the value of this property.

Returns: Dataframe with the information about the connections of the grid.

Setter: Sets the property to the provided dataframe, if the dataframe has the columns [“connection”, “node1”, “node2”].

Type: pd.dataframe

static `get_list_of_connection_ids()`

Return a list of all node ID numbers of the selected grid.

setup_distances()

Sets up the distances of all connections, using pint units.

Uses geodesic distance from the package GeoPy (<https://geopy.readthedocs.io/en/stable/#module-geopy.distance>) to calculate the distance between the two nodes of a connection.

static `calculate_distances(connection)`

Calculates the distance or takes it from existing data.

Uses geodesic distance from the package GeoPy (<https://geopy.readthedocs.io/en/stable/#module-geopy.distance>) to calculate the distance between the two nodes of a connection, if no distance is provided in the data.

Parameters `connection` – An itertuple from the connections DataFrame.

static `selected()`

This methods gets the value of the static variable “selected”.

This method returns the Grid object which is selected to be used as grid for the optimization. The static variable “selected” can only contain one Grid object and is set using the method “select()” of an instance of the Grid class.

Returns Value of `_selected`, which is either None or a Grid instance.

select()

This methods sets the value of the static member “selected”.

The static member “selected” is set to reference the Grid instance which called this method.

class `secmod.classes.Product(product_path: pathlib.Path)`

Bases: `object`

This class is used for all product related data, e.g. demand, impacts, etc.

DEFAULT_COST = 3000

products = []

classmethod `load_products_from_directory(products_path: pathlib.Path)`

Loads all products of a specific class into their class.

property name

This property contains a unique name for the grid.

Getter: Gets the value of this property.

Returns: Name of the grid as string.

Setter: Sets the property to the provided name, if it isn’t already used by another grid.

Raises: `ValueError`: If the name is already used by another grid.

Type: `str`

_get_price_time_series(year: int)

Returns the absolute price time series

_get_selling_price_time_series(year: int)

Returns the absolute selling price time series

_get_cost_non_served_demand(reference_year: int, invest_year: int)

This method gets the costs of non-served demand of a product, i.e. for purchasing a product.

Calculates the present value of the selected impact year and its investment period in the reference year for a product.

Parameters

- **reference_year** (`int`) – Reference year of the optimized time horizon, e.g. 2020
- **invest_year** (`int`) – An investment year of the optimized time horizon, e.g. 2025

`_get_revenues_add_served_demand`(*reference_year: int, invest_year: int*)

This method gets the costs of add-served demand of a product, i.e. for selling a product.

Calculates the present value of the selected impact year and its investment period in the reference year for a product.

Parameters

- **reference_year** (*int*) – Reference year of the optimized time horizon, e.g. 2020
- **invest_year** (*int*) – An investment year of the optimized time horizon, e.g. 2025

`get_impact_non_served_demand`(*impact_categories: list, reference_year: int, invest_years: list*)

This methods gets the impact of non-served demand of a product in one or multiple impact categories for a specific year.

If the impact category is “cost”, the corresponding method `Product._get_cost_non_served_demand` is used. In all other cases the impacts of all ecoinvent processes defined by the process itself as the impacts of non-served demand, are multiplied with their corresponding weight factors and summed up together.

`static get_combined_impact_non_served_demand()`

Returns the combined impact time series for all products and years.

`get_impact_add_served_demand`(*impact_categories: list, reference_year: int, invest_years: list*)

This methods gets the impact of add-served demand of a product in one or multiple impact categories for a specific year.

If the impact category is “cost”, the corresponding method `Product._get_cost_non_served_demand` is used. In all other cases the impacts of all ecoinvent processes defined by the process itself as the impacts of non-served demand, are multiplied with their corresponding weight factors and summed up together.

`static get_combined_impact_add_served_demand()`

Returns the combined impact time series for all products and years.

`get_nodal_demand_time_series`(*year: int*)

Returns the absolute nodal demand time series.

`static get_combined_demand_time_series()`

Returns the combined demand time series for all products and years.

`get_required_total_secured_capacity`(*invest_years: list*)

Return the required secured capacity in the whole grid for product in a specific year.

`static get_combined_required_total_secured_capacity()`

Returns the combined required total secured capacity for all products and years.

`get_required_nodal_secured_capacity`(*nodes: list, invest_years: list*)

Return the required secured capacity in the whole grid for product in a specific year.

`static get_combined_required_nodal_secured_capacity()`

Returns the combined required nodal secured capacity for all products and years.

`class secmod.classes.ImpactCategory`(*impact_category_path: pathlib.Path*)

Bases: object

This class is used to manage impact categories, their corresponding limits and costs of overshoot.

`impact_categories` = []

`FRAMEWORK` = 'ReCiPe Midpoint (H)'

`MANUAL_IMPACT_CATEGORY_SELECTION` = []

property name

This property contains a unique name for the impact category.

Getter: Gets the value of this property.

Returns: Name of the impact category as string.

Setter: Sets the property to the provided name, if it isn't already used by another impact category.

Raises: ValueError: If the name is already used by another impact category.

Type: str

property ecoinvent_name

This property contains a unique name for the impact category, which is used by ecoinvent.

Getter: Gets the value of this property.

Returns: ecoinvent name of the impact category as string.

Setter: Sets the property to the provided ecoinvent name, if it isn't already used by another impact category.

Raises: ValueError: If the ecoinvent name is already used by another impact category.

Type: str

classmethod load_impact_categories_from_directory(*impact_categories_path: pathlib.Path*)

Loads all impact categories of a specific class into their class.

static get_list_of_active_impact_categories()

Returns a list of the all active impact categories in the currently selected framework.

static get_list_of_names_of_active_impact_categories()

Returns a list of the ecoinvent names of all impact categories active in the currently selected framework.

_get_operational_nodal_impact_limits(*nodes: list, invest_years: list*)

Return the operational nodal impact limit for an impact category in a specific year.

static get_combined_operational_nodal_impact_limits()

Returns the combined operational nodal impact limit for all impact categories and years.

_get_invest_nodal_impact_limits(*nodes: list, invest_years: list*)

Return the invest nodal impact limit for an impact category in a specific year.

static get_combined_invest_nodal_impact_limits()

Returns the combined invest nodal impact limit for all impact categories and years.

_get_total_nodal_impact_limits(*nodes: list, invest_years: list*)

Return the total nodal impact limit for an impact category in a specific year.

static get_combined_total_nodal_impact_limits()

Returns the combined total nodal impact limit for all impact categories and years.

_get_operational_impact_limits(*invest_years: list*)

Return the operational impact limit for an impact category in a specific year.

static get_combined_operational_impact_limits()

Returns the combined operational impact limit for all impact categories and years.

_get_invest_impact_limits(*invest_years: list*)

Return the invest impact limit for an impact category in a specific year.

static get_combined_invest_impact_limits()

Returns the combined invest impact limit for all impact categories and years.

_get_total_impact_limits(*invest_years: list*)

Return the total impact limit for an impact category in a specific year.

static get_combined_total_impact_limits()

Returns the combined total impact limit for all impact categories and years.

_get_objective_factor_impact(*invest_years: list*)

Return the total impact limit for an impact category in a specific year.

static get_combined_objective_factor_impact()

Returns the combined total impact limit for all impact categories and years.

_get_objective_factor_impact_overshoot(*invest_years: list*)

Return the objective factors for impact overshoots for an impact category in a specific year.

static get_combined_objective_factor_impact_overshoot()

Returns the combined objective factors for impact overshoots for all impact categories and years.

class `secmod.classes.ProcessImpacts(process_path: Optional[pathlib.Path] = None)`

Bases: `abc.ABC`

This class is used as an interface for process impacts.

In this class two abstract methods are defined which need to be implemented by subclasses. Therefore this class acts as an interface which is implemented by the inheriting classes. This way it is assured that the methods used to get the impacts of investment and operation can be called in all inheriting classes.

Furthermore it includes static variables for the economic time period and assumed interest rate. And since costs are special impacts, which all processes must have, the instance variables for costs are defined in this class. Furthermore the initialization of this class imports the costs from a processes directory, if provided. Methods to get costs of investment and operation are implemented as well.

Parameters `process_path` (*Path*) – A path of the directory of an existing process definition, which includes a CSV-file with data about the costs of the process.

interest_rate

The interest rate used for economic impacts as share of 1.

Type float

economic_period

The time period used for the calculation of all annualized impacts.

Type pint quantity

Raises `FileNotFoundError` – If no file “costs.csv” can be found in the process directory.

`IMPACT_SOURCES = {None: ['operation', 'invest']}`

`interest_rate = 0.05`

`economic_period = <Quantity(30, 'year')>`

`invest_years = [2016, 2020, 2025, 2030, 2035, 2040, 2045, 2050]`

`construction_years = []`

`processes = []`

property name

This property contains a unique name for the grid.

Getter: Gets the value of this property.

Returns: Name of the grid as string.

Setter: Sets the property to the provided name, if it isn't already used by another grid.

Raises: ValueError: If the name is already used by another grid.

Type: str

abstract get_impact_invest(*impact_categories: list, construction_year: int, invest_year: int*)

This method gets the annual investment impacts of a process

Parameters

- **impacts_categories** (*list*) – List of strings, which define for which impact categories the yearly investment impacts shall be returned
- **construction_year** (*int*) – Year of construction of an instance of this process
- **invest_year** (*int*) – Year for which the impact is calculated

abstract get_impact_operation(*impact_categories: list, construction_year: int, reference_year: int, invest_year: int*)

This method gets the operational impacts of a process

Parameters

- **impacts_categories** (*list*) – List of strings, which define for which impact categories the yearly investment impacts shall be returned
- **construction_year** (*int*) – Year of construction of an instance of this process
- **invest_year** (*int*) – Year for which the impact is calculated

static _get_investment_period_duration(*invest_year: int*)

Gets the duration of the period between this and the next invest year.

If it is the last year in the list of invest years, a period duration of one year is returned.

Parameters invest_year (*int*) – A year, e.g. 2020.

_get_cost_invest(*construction_year: int, reference_year: int, invest_year: int*)

This method gets the annual investment costs of a process

Calculates the present value of the selected impact year and its investment period in the reference year for a process instance build in the construction year.

Parameters

- **construction_year** (*int*) – Year of construction, e.g. 2016. Earliest year is used, if construction year is earlier than the earliest year in the data.
- **reference_year** (*int*) – Reference year of the optimized time horizon, e.g. 2020
- **invest_year** (*int*) – An investment year of the optimized time horizon, e.g. 2025

_get_cost_operation(*construction_year: int, reference_year: int, invest_year: int*)

This method gets the annual investment costs of a process

Calculates the present value of the selected impact year and its investment period in the reference year for a process instance build in the construction year.

Parameters

- **construction_year** (*int*) – Year of construction, e.g. 2016. Earliest year is used, if construction year is earlier than the earliest year in the data.
- **reference_year** (*int*) – Reference year of the optimized time horizon, e.g. 2020
- **invest_year** (*int*) – An investment year of the optimized time horizon, e.g. 2025

classmethod `get_combined_impact_matrix()`

Method that is used to return the combined impact matrix of all process of a process category.

static `_get_single_combined_impact_matrix(combined_impact_matrix_list, process, impact_categories, construction_years, reference_year, invest_years)`

Returns impacts for component investment and operation

get_lifetime_duration(construction_years: list)

Return the lifetime duration of the process for a list of construction years

classmethod `get_combined_lifetime_duration()`

Returns the combined lifetime durations of all processes in the class using the construction years of the class.

abstract classmethod `setup_construction_years()`

Abstract method to setup the construction years from existing capacity data.

setup_years()

`_abc_impl = <_abc_data object>`

class `secmod.classes.EcoinventImpacts(process_path: Optional[pathlib.Path] = None)`

Bases: `secmod.classes.ProcessImpacts`

This class provides impacts based on a list of ecoinvent processes.

An instance based on this class gets its impacts from the ecoinvent database. The impacts are calculated by adding a list of ecoinvent processes multiplied with a scaling factor, which are used to define the environmental impact of a process in SecMOD.

Parameters `process_path (Path)` – A path to the directory of an ecoinvent process. If no path is provided, an empty instance will be created.

Raises `NotADirectoryError` – If provided path is not a directory.

`database = None`

`subassemblies = {}`

static `load_ecoinvent_database(database_path: pathlib.Path, units_to_change_path: Optional[pathlib.Path] = None)`

Loads the impacts of all ecoinvent processes from a multiindexed CSV-file.

Additionally this methods loads new unit defintions to be used in pint and translates ecoinvent units to the necessary format to be used with pint.

Furthermore the process names are edited to match previous naming logic from SecMOD 1.0.

Parameters

- **database_path (Path)** – A path which includes the filename of the CSV-file.
- **units_to_change_path (Path)** – A path to a JSON-file which defines a dictionary of existing units and the units they shall be replaced with.
- **new_pint_units (Path)** – A path to a TXT-file which includes new unit definitions to be used as pint units. E.g. “car = []” or “mass_CO2_equivalent = [GWP100] = CO2_eq”

static `_translate_process_name_to_ecoinvent_identifier(process_name: str)`

Modifies a process name to match the ecoinvent identifiers.

Deletes special charaters, spaces, commas, dots, etc. and replaces them by underscores. Furthermore secures that there is only one underscore in a row. e.g.:

“passenger car, electric, without battery//[GLO] passenger car production, electric, without battery” becomes “passenger_car_electric_without_battery_GLO_passenger_car_production_electric_without_battery”

static `_translate_units_to_pint`(*json_path: pathlib.Path*)

Modifies units of impacts to match pint unit definitions.

This method simply exchanges some units from the ecoinvent database to fit the corresponding unit definitions in pint. It uses a dictionary to replace the units. This dictionary is provided as a JSON-file. The path to the file has to be given to this method or preferably to the method `load_ecoinvent_database`.

Note: If new units are needed they have to be added to this JSON-file. e.g. like this

“kg PM2.5-.”: “kg PM2_5_eq”

static `load_ecoinvent_subassemblies`(*subassemblies_path: pathlib.Path*)

Loads ecoinvent subassemblies from a directory.

This methods automatically detects all CSV-files in a directory and assumes that all of them are ecoinvent subassemblies.

get_impact_invest(*impact_categories: list, construction_years: list, reference_year: int, invest_years: list*)

This method gets the invest impact of a process in one or multiple impact categories for a specific year.

This method determines the value of the requested impact categories for a process in a specific year. It takes into account when the process capacity was constructed, which year is the reference year of the current optimization horizon and which year is actually the current invest year.

If the impact category is “cost”, the corresponding method `ProcessImpacts._get_cost_invest` is used. In all other cases the impacts of all ecoinvent processes defined by the process itself as the invest phase, are multiplied with their corresponding weight factors and summed up together.

It returns the results as a dictionary of impact categories and their corresponding value, including the correct units.

Parameters

- **impact_categories** (*list*) – A list of one or more impact categories to be determined.
- **construction_year** (*int*) – The year in which the process capacity was build. It is used to determine the actual impact a capacity build in that year has. Earliest year is used, if construction year is earlier than the earliest year in the data.
- **reference_year** (*int*) – The reference year of the current optimization horizon. It is only used to discount the cost of the process during the whole optimization horizon to the reference year value.
- **invest_year** (*int*) – The invest year investigated right now. It is used to determine whether a process still got invest annuities to pay, only maintenance costs remaining or is already beyond its lifetime duration and therefore has no invest impacts anymore at all.

get_impact_operation(*impact_categories: list, construction_years: int, reference_year: int, invest_years: int*)

This method gets the operational impact of a process in one or multiple impact categories for a specific year.

This method determines the value of the requested impact categories for a process in a specific year. It takes into account when the process capacity was constructed, which year is the reference year of the current optimization horizon and which year is actually the current invest year.

If the impact category is “cost”, the corresponding method `ProcessImpacts._get_cost_operation` is used. In all other cases the impacts of all ecoinvent processes defined by the process itself as the operational phase, are multiplied with their corresponding weight factors and summed up together.

It returns the results as a `DataFrame` of impact categories and their corresponding value, including the correct units.

Parameters

- **impact_categories** (*list*) – A list of one or more impact categories to be determined.
- **construction_year** (*int*) – The year in which the process capacity was build. It is used to determine the actual impact a capacity build in that year has. Earliest year is used, if construction year is earlier than the earliest year in the data.
- **reference_year** (*int*) – The reference year of the current optimization horizon. It is only used to discount the cost of the process during the whole optimization horizon to the reference year value.
- **invest_year** (*int*) – The invest year investigated right now. It is only used to discount the cost of the process during the whole optimization horizon to the reference year value.

static `_get_ecoinvent_process_impact`(*construction_year_attribute_name, process, impact_category: str, construction_year: int, impact_year: int*)

Get the impact of a process in a specific impact category and construction year.

Takes in an interable tuple from a process list (e.g. `self._processes_invest`) and returns the corresponding impact value.

Parameters

- **process_list** (*pd.DataFrame*) – A `DataFrame`, which represents a list of processes, e.g. `self._processes_invest`
- **process** – An element of `process_list.itertuple()` which is currently evaluated.
- **impact_category** (*str*) – The name of the evaluated impact category.
- **construction_year** (*int*) – The year of construction for which the impact is determined. If the year of construction is earlier than the earliest year in the data, the earliest year is used.
- **impact_year** (*int*) – The year in which the impact actually occurs. For invest impacts the `impact_year` is equal to the `construction_year`. If the `impact_year` is earlier than the earliest year in the data, the earliest year is used.

Raises **KeyError** – If the searched process does not exist in the ecoinvent database and seems to be used, since its weight factor is not zero.

static `_isSubassembly`(*process_name: str*)

Checks whether a process is actually a subassembly

static `_recursively_get_impact`(*process_list: pandas.core.frame.DataFrame, impact_category: str, construction_year: int, impact_year: int*)

Recursively calculates the impact of a process list

`_abc_impl = <_abc_data object>`

class `secmod.classes.ManualImpact`(*process_path: Optional[pathlib.Path] = None*)

Bases: `secmod.classes.ProcessImpacts`

This is the class for manual impact definitions, when own databases are used. Currently, this is not fully implemented yet.

get_impact_invest(*impact_categories: list, construction_year: int, reference_year: int, invest_year: int*)
Get manual invest impacts.

get_impact_operation(*impact_categories: list, construction_year: int, reference_year: int, invest_year: int*)
Get operational invest impacts.

_abc_impl = <_abc_data object>

class `secmod.classes.Process`(*process_path: pathlib.Path*)

Bases: `secmod.classes.ProcessImpacts`

This is the base class for processes of all kind.

It inherits from ProcessImpacts, since all processes have impacts and costs.

Parameters **process_path** (*Path*) – The path to a process directory.

construction_years = []

locations = []

classmethod **load_processes_from_directory**(*process_category_path: pathlib.Path*)

Loads all processes of a specific class into their class.

get_numerical_property(*property_to_return: str*)

abstract **get_existing_capacity**(*locations: list, invest_years: list*)

abstract **get_potential_capacity**(*locations: list, invest_years: list*)

add_capacity(*location: int, construction_year: int, new_capacity: <module 'pint.quantity' from 'C:\Users\vinolzen\AppData\Roaming\Python\Python38\site-packages\pint\quantity.py'>*)

Abstract method that adds capacity to a specific node or connection in a specific construction year.

classmethod **get_combined_existing_capacity**()

Returns the existing capacity

classmethod **get_combined_potential_capacity**()

Returns the potential capacity

classmethod **setup_construction_years**(*combined_existing_capacity=Series([], dtype: float64)*)

Looks up the required construction years for this process class.

A matrix of combined existing capacity can be provided to speed up the process. Otherwise the method will create a new combined_existing_capacity matrix.

_abc_impl = <_abc_data object>

class `secmod.classes.NodalProcess`(*process_path: pathlib.Path*)

Bases: `secmod.classes.Process`

This is the class for nodal processes. It inherits from Process.

Parameters **process_path** (*Path*) – The path to a process directory.

locations = []

get_existing_capacity(*nodes: list, invest_years: list*)

Returns existing capacity.

get_potential_capacity(*locations: list, invest_years: list*)

Returns potential capacity.

get_secured_capacity_factor(*construction_years: list*)

Returns secured capacity factor.

classmethod get_combined_secured_capacity_factor()

Gets the secured capacity factors for all processes of a process category.

abstract get_technologymatrix(*construction_years: list, products: list*)

classmethod get_combined_technology_matrix(*products: list*)

Returns technology matrix for all production processes.

get_maximum_production_share(*nodes: list, invest_years: list*)

Returns maximum production share.

_abc_impl = <_abc_data object>

class `secmod.classes.ConnectionProcess`(*process_path: pathlib.Path*)

Bases: `secmod.classes.Process`

This is the class for connection processes. It inherits from Process.

Parameters **process_path** (*Path*) – The path to a process directory.

locations = []

get_existing_capacity(*connections: list, invest_years: list*)

Returns existing capacity.

abstract get_potential_capacity(*locations: list, invest_years: list*)

Returns potential capacity.

_abc_impl = <_abc_data object>

class `secmod.classes.ProductionProcess`(*process_path: pathlib.Path*)

Bases: `secmod.classes.NodalProcess`

This is the class for production processes. It inherits from nodal process.

Parameters **process_path** (*Path*) – The path to a process directory.

processes = []

construction_years = []

get_technologymatrix(*construction_years: list, products: list*)

Return technologymatrix for a single process.

For MILP, the technology matrix consists of several part-load sections

get_availability_timeseries()

Returns availability timeseries

static get_combined_availability_timeseries()

Returns combined availability timeseries

get_technology_matrix_timeseries()

Returns time-dependent technologymatrix

static get_combined_technology_matrix_timeseries()

Returns combined availability timeseries

get_cornerpoints()

determines the number of corner points/part-load sections to discretize the process in an MILP

static get_combined_cornerpoints()

determines the number of cornerpoints for all processes”

```

get_reference_products()
    determines the reference product of a production process

static get_combined_reference_product()
    determines the reference products for all production processes

get_gradient_technologymatrix(construction_years: list, products: list)
    determines the gradient between two cornerpoints/in a section for the defined process

get_combined_gradient_technologymatrix(products: list)
    get gradients for all processes

get_relative_partload_capacity(construction_years: list, products: list)
    get the amount of capacity that can be used at each cornerpoint

get_combined_relative_partload_capacity(products: list)
    get the amount of capacity that can be used at each cornerpoint for all processes

get_minimal_capacity(locations: list, invest_years: list)
    get minimal capacity to be built for a production process

classmethod get_combined_minimal_capacity()
    get minimal capacities to be built for all processes

_abc_impl = <_abc_data object>

```

```

class secmod.classes.StorageProcess(process_path: pathlib.Path)
    Bases: secmod.classes.NodalProcess

```

This is the class for storage processes. It inherits from nodal process.

Parameters **process_path** (*Path*) – The path to a process directory.

```

DIRECTIONS_STORAGE = {None: ['deposit', 'withdraw']}
STORAGE_LEVEL_FACTOR = {'deposit': 1, 'withdraw': -1}
processes = []
construction_years = []

get_technologymatrix(construction_years: list, products: list)
    Returns the technologymatrix for storage process

_abc_impl = <_abc_data object>

get_flow_to_storage_capacity_factor(construction_years: list)
    Returns the flow-to-storagecapacity-factor

static get_combined_flow_to_storage_capacity_factor()
    Returns the combined flow to storagecapacity factor

static get_combined_storage_products(products)
    Return a dictionary of all storage processes and their storeable product.

get_minimal_capacity(locations: list, invest_years: list)
    get minimal capacity to be built for a storage process

classmethod get_combined_minimal_capacity()
    get minimal capacity to be built for all storage processes

get_min_rel_storage_level_factor(construction_years: list)
    return minimal relative storage level

static get_combined_min_rel_storage_level_factor()
    return minimal relative storage level for all storage processes

```

get_rel_storage_loss_factor(*construction_years: list*)
returns storage loss between to time steps for a single storage process

static get_combined_rel_storage_loss_factor()
returns storage loss between to time steps for all storage processes

class `secmod.classes.TransshipmentProcess`(*process_path: pathlib.Path*)
Bases: `secmod.classes.ConnectionProcess`

This is the class for transshipment processes. It inherits from connection process.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

`DIRECTIONS_TRANSSSHIPMENT = {None: ['forward', 'backward']}`

`DIRECTION_FACTOR_TRANSSSHIPMENT = {'backward': -1, 'forward': 1}`

`processes = []`

`construction_years = []`

get_transshipment_efficiency()
Return a dataframe with the efficiency of every connection for a transshipment process.

get_potential_capacity(*locations: list, invest_years: list*)
Return potential capacity.

classmethod get_combined_transshipment_efficiency()
Return a dataframe with the efficiencies of every connection for every transshipment process.

classmethod get_combined_transshipment_products()
Returns a dictionary with the names and products of all transshipment processes

class `secmod.classes.TransmissionProcess`(*process_path: pathlib.Path*)
Bases: `secmod.classes.ConnectionProcess`

This is the class for transmission processes. It inherits from connection process.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

`PRODUCTS = {None: ['electricity']}`

`processes = []`

`construction_years = []`

`per_unit_base = <Quantity(500, 'megavolt_ampere')>`

`reference_node = 1`

get_potential_capacity(*locations: list, invest_years: list*)
Returns potential capacity

_get_process_connection_properties(*connections: list, invest_years: list*)
Returns connection properties, such as circuits, power limit, safety margin, and more.

static _get_process_by_voltage(*voltage: <module 'pint.quantity' from 'C:\Users\vmolzen\AppData\Roaming\Python\Python38\site-packages\pint\quantity.py'>*)
Returns processes sorted by voltage.

static get_connection_properties()
Returns connection properties.

static get_combined_power_limit_per_circuit()

Returns a dictionary of the power limits per circuit of all transmission processes.

static get_combined_safety_margin()

Returns a dictionary of the safety margin of all transmission processes.

class `secmod.classes.ProductionProcessEcoinvent`(*process_path: pathlib.Path*)

Bases: `secmod.classes.ProductionProcess`, `secmod.classes.EcoinventImpacts`

This is the class for ecoinvent production processes. It inherits from `ProductionProcess` and `EcoinventImpacts`.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

class `secmod.classes.ProductionProcessManual`(*process_path: pathlib.Path*)

Bases: `secmod.classes.ProductionProcess`, `secmod.classes.ManualImpact`

This is the class for manually defined production processes. It inherits from `ProductionProcess` and `ManualImpact`.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

class `secmod.classes.StorageProcessEcoinvent`(*process_path: pathlib.Path*)

Bases: `secmod.classes.StorageProcess`, `secmod.classes.EcoinventImpacts`

This is the class for ecoinvent storage processes. It inherits from `StorageProcess` and `Ecoinvent Impacts`.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

class `secmod.classes.StorageProcessManual`(*process_path: pathlib.Path*)

Bases: `secmod.classes.StorageProcess`, `secmod.classes.ManualImpact`

This is the class for manually defined storage processes. It inherits from `StorageProcess` and `ManualImpact`.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

class `secmod.classes.TransshipmentProcessEcoinvent`(*process_path: pathlib.Path*)

Bases: `secmod.classes.TransshipmentProcess`, `secmod.classes.EcoinventImpacts`

This is the class for ecoinvent transshipment processes. It inherits from `TransshipmentProcess` and `EcoinventImpacts`.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

class `secmod.classes.TransshipmentProcessManual`(*process_path: pathlib.Path*)

Bases: `secmod.classes.TransshipmentProcess`, `secmod.classes.ManualImpact`

This is the class for manually defined transshipment processes. It inherits from `TransshipmentProcess` and `Ecoinvent Impacts`.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

class `secmod.classes.TransmissionProcessEcoinvent`(*process_path: pathlib.Path*)

Bases: `secmod.classes.TransmissionProcess`, `secmod.classes.EcoinventImpacts`

This is the class for ecoinvent transmission processes. It inherits from `TransmissionProcess` and `EcoinventImpacts`.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

class `secmod.classes.TransmissionProcessManual` (*process_path: pathlib.Path*)

Bases: `secmod.classes.TransmissionProcess`, `secmod.classes.ManualImpact`

This is the class for manually-defined transmission processes. It inherits from `TransmissionProcess` and `EcoinventImpacts`.

Parameters `process_path` (*Path*) – The path to a process directory.

`_abc_impl = <_abc_data object>`

`secmod.classes.calculate_power_line_power_limit` (*power_limit, safety_margin, circuits*)

`secmod.classes.calculate_power_line_resistance_per_unit` (*specific_resistance, distance, circuits, voltage, per_unit_base*)

`secmod.classes.calculate_power_line_susceptance_per_unit` (*reactance_per_unit, resistance_per_unit*)

`secmod.classes.calculate_voltage_switch_power_limit` (*target_power_limit, base_power_limit, safety_margin, circuits*)

`secmod.classes.calculate_voltage_switch_resistance_per_unit` (*target_voltage, base_voltage, target_specific_resistance, base_specific_resistance, per_unit_base, circuits, distance*)

4.2 secmod.data_preprocessing module

`secmod.data_preprocessing.download_datapackage` (*url: str, download_path: pathlib.Path*) → bool

This method is used to download an Open Source Data Package.

Using the URL to the meta data JSON-file of the data package, the complete data package is downloaded into a download folder. If the download has been successful a boolean True is returned.

Parameters

- **url** (*str*) – URL to the directory of the meta data JSON-file of the data package (e.g. https://data.open-power-system-data.org/renewable_power_plants/latest/datapackage.json)
- **download_path** (*Path*) – Path of the directory where the data package is to be saved

`secmod.data_preprocessing.datapackage_update_available` (*url: str, download_path: pathlib.Path*) → bool

This method checks if there is an update available for a remote data package.

First the methods tries to open the remote data package to check its availability. If it is not available, False is returned.

After that the local data package is opened, if it exists. If it does not exist, True is returned.

If both sources are accessible the version information from the meta data is compared. Because the remote source is expected to have the newest version, any difference in the version information returns True.

Parameters

- **url** (*str*) – URL to the directory of the meta data JSON-file of the data package (e.g. https://data.open-power-system-data.org/renewable_power_plants/latest/datapackage.json)

- **download_path** (*Path*) – Path of the directory where the data package is to be saved

`secmod.data_preprocessing.download_file(url: str, save_to: str, expected_size: Optional[int] = None)`

This method downloads files using streaming.

A get request is used to download the file from the URL. The response is streamed into a local file to minimize RAM usage.

Parameters

- **url** (*str*) – URL to the file
- **save_to** (*str*) – local path and filename to be saved to

`secmod.data_preprocessing.is_setup(working_directory: pathlib.Path) → bool`

This method checks if SecMOD has been set up before.

This is achieved by checking if the folder “SecMOD” exists in the working directory and returning a boolean. If it does not exist, the method returns False.

Parameters **working_directory** (*Path*) – Path of the working directory

`secmod.data_preprocessing.create_directory(path: pathlib.Path)`

This method creates a new directory, if it does not exist, yet.

The method uses pathlib to check for the existence of a path. If the path does not exist yet, it get created.

Parameters **path** (*Path*) – Path of the directory to be created

4.3 secmod.data_processing module

`secmod.data_processing.load_raw_input(raw_input_directory: pathlib.Path)`

Loads all data from raw input.

`secmod.data_processing.load_all_processes(processes_directory: pathlib.Path)`

Loads all processes in their corresponding class.

`secmod.data_processing.generate_input_dictionary(computed_input_path: pathlib.Path, load: bool = False)`

Takes all loaded data and generates an input dictionary for the optimization model.

`secmod.data_processing.deunitize_input_dictionary(input_dictionary: dict)`

Removes all units from the input dictionary to make it handle

`secmod.data_processing.get_dimensionalities_from_input_dictionary(input_dictionary: dict)`

Returns a list with all unique dimensionalities and an example unit

4.4 secmod.evaluation module

`class secmod.evaluation.AbstractFrame_evaluation(parent, optimization_results, working_directory, frac_height=0.8, frac_width=0.8)`

Bases: `tkinter.Frame`

This is the abstract GUI Frame class

It is used to define the common parts of the MainFrame and the DetailedFrame.

startPlot()

start setup or plot of figure

setupFigure()

set up figure in north frame

calculateFrameGeometry(*idxrow=0, idxcolumn=0*)

Calculate width and height of frame in GUI

plotFigure()

plot balance

getbalanceUnits(*grouped_balance, unit_time_summation, b_extendedDataframe=False*)

Get units of balance in optimization

selectPlotType()

plots the data according to specified plot type

createLegend()

creates separate Legends

setColors()

return list of colors for plot

highlightMouseSelection(*event*)

highlight selection over which mouse currently hovers

showTable(*extendedDataframe=False*)

Show table with values of dataframe

adjustManualColumnWidths()

Adjust column width of table. Copied from pandastable documentation and cleared line "if w > 200

savePlotData()

opens new Save Frame

groupBalanceByIndex(*input_balance*)

in mainframe no grouping necessary. Method overwritten in Detailed Frame

getNonzeroIndizes(*summed_level=None*)

get indizes of balance whose values are nonzero

abstract getTitleString()

abstract method. Needs to be implemented in Child Class

class `secmod.evaluation.MainFrame_evaluation`(*parent, optimization_results, working_directory*)

Bases: `secmod.evaluation.AbstractFrame_evaluation`

This is the Main Frame of the evaluation GUI

It is a child class of AbstractFrame_evaluation

setUpRadiobuttons(**args*)

Set up and change radiobuttons for product flows/impacts/processes

getDataframeFromSelection(**args*)

Create Dataframe which contains the information representing the selection, i.e. selection of Product, Balance and Impact/Capacities

disableEnableRadiobuttons()

Disable radiobuttons of balance specification, which is not available for product

Only necessary for single product

plotFigure()

plot balance

inherited from AbstractFrame_evaluation but click functionality added

openDetailedWindow(*event*)
 open Detailed Frame to further inspect line/bar/area clicked on

getTitleString()
 Get Title String depended on selection of balance

class `secmod.evaluation.DetailedFrame_evaluation`(*parent, optimization_results, working_directory, reduced_balance, mainframe*)

Bases: `secmod.evaluation.AbstractFrame_evaluation`

This is the Detailed Frame of the evaluation GUI

It is a child class of AbstractFrame_evaluation

disableEnableCheckbuttons()
 Disable checkbuttons of aggregation level, which is not available in balance

groupBalanceByIndex(*input_balance*)
 Group balance by level of aggregation. Index selected by checkbuttons

showTable()
 show table
 select whether detailed or entire dataset shown in table

getTitleString()
 Get Title String depended on selection of balance
 Same Title as MainFrame, only 'Detailed' added

class `secmod.evaluation.SaveFrame`(*parent, mainframe*)

Bases: `object`

New Frame in order to save plot and Data

toggleSaveButton()
 disables the SaveButton if no checkbutton selected

browseFolder()
 Browses Folder and selects new Folder

saveSelection()
 saves the selection as <filename> in <folderpath>

`secmod.evaluation.evaluate`(*working_directory*)
 Evaluates the results of the optimization.

`secmod.evaluation.start_evaluation`(*working_directory*)

4.5 secmod.helpers module

`secmod.helpers.isInteger`(*text: str*)
 Checks if string could be an integer.

`secmod.helpers.unitize_dataframe`(*dataframe: pandas.core.frame.DataFrame, units: pint.registry.UnitRegistry*)
 Multiplied every row of a dataframe with the corresponding unit and drops the unit column.

`secmod.helpers.log_heading`(*message: str*)
 Takes a message and creates a nicely formatted heading log message.

`secmod.helpers.clear_filename_special_character`(*filename: str*)

`secmod.helpers.convert_date(date_old, correct_dateformat, partially_correct_dateformat, wrong_dateformat)`
 converts date to correct dateformat

`secmod.helpers.correct_time_stamp_of_timeseries(timeseries)`
 corrects time stamp of timeseries
 format desired, which is given in config (e.g. dd.mm.yyyy HH:MM)

`secmod.helpers.get_rwth_colors()`
 return RWTH colors as array

4.6 secmod.optimization_MILP module

class `secmod.optimization_MILP.Optimization_MILP`

Bases: `abc.ABC`

This is the class for optimization models.

It includes all declarations of Sets, Parameters, Variables and Constraints. Furthermore it contains all optimization methods.

setupSets()

This method sets up all Sets required by all optimization models.

Some Sets are initialized with default values, if no other values are specified in the input file.

setupParameters()

This method sets up all Parameters required by all optimization models.

setupVariables()

This method sets up all Variables required by all optimization models.

setupConstraints()

This method sets up all Constraints required by all optimization models.

setupObjective()

This method declares the pe.Objective required by all optimization models.

instantiate_model(*input_dict: Optional[dict] = None, filepath: Optional[str] = None, skip_instantiation: bool = False*)

This method creates a model instance and fixes variables

run(*input_dict: Optional[dict] = None, filepath: Optional[str] = None, solver: str = 'glpk', solver_options: Optional[dict] = None, debug: bool = False, skip_instantiation: bool = False*)

This method is used to start an optimization.

To start an optimization it is necessary to provide a file with the necessary input data. This file includes all information required to instantiate the optimization problem.

Parameters

- **input_dict** (*dict*) – Input dictionary that contains all the necessary data for the optimization
- **filename** (*str*) – Filename of the previously generated input file
- **solve** (*str*) – Name of the solve to be used. Standard is ‘gurobi’

fix_transmission_reference_node_phase_difference()

This method is used as rule for fixing the transmission phase difference of one node to zero.

This is necessary for the calculation in the DC load flow model of the transmission grid. Furthermore the reference node should be a node at which no loads of any kind occur. Therefore extractions or feed-in must not take place at this node.

fix_unneeded_variables()

Fixes unneeded variables to 0 for all processes.

static get_dataframe_from_result(model_instance_variable, unstack: Optional[list] = None)

Returns a dataframe with the results from the optimization model.

static get_dataframe_from_parameter(model_instance_variable, unstack: Optional[list] = None)

Returns a dataframe with the results from the optimization model.

get_used_capacity_from_result()

Return a dictionary of dataframes with the used capacities from the optimization.

get_non_served_demand_from_result()

Return a dictionary of dataframes with the non_served_demand from the optimization.

static constraint_product_balance_rule(model, node: int, product: str, year: int, time_slice: str) → pyomo rule

This method is used as rule for the product balance equation.

This expression represents the product balance equation which is unique for each node, product, year and time slice. It can be summarized as:

$$\text{Production} + (\text{Transshipment}) + \text{Storage} + (\text{Transmission}) + \text{Purchase} = \text{Demand} + \text{Selling}$$

Therefore it takes the named above as parameters.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **node** (*int*) – A node at which the pe.Constraint is active
- **product** (*str*) – A product at which is constrained
- **year** (*int*) – A year at which the pe.Constraint is active
- **time_slice** (*str*) – A time_slice at which the pe.Constraint is active

static constraint_restrict_export_rule(model, node: int, product: str, year: int, time_slice: str) → pyomo rule

Restrict exports of products you dont want to sell/you dont want to allow sell.

static constraint_restrict_import_rule(model, node: int, product: str, year: int, time_slice: str) → pyomo rule

This method is used as rule for preventing import of non-importable products.

static constraint_usable_production_rule(model, node: int, process_production: str, year: int, year_construction: int, time_slice: str, vertex: int) → pyomo rule

This method is used to constrain the used production to the available production in each time slice.

static constraint_production_limit_binaries_rule(model, node: int, process_production: str, year: int, year_construction: int, time_slice: str) → pyomo rule

This method is used as rule for limiting the number of binaries which are allowed in partload behaviour of components, i.e. one partload segment can be chosen at maximum.

static constraint_production_limit_by_capacity_upper_rule(*model, node: int,*
process_production: str, year: int,
year_construction: int, time_slice:
str, vertex: int) → pyomo rule

This method constrains the used_production to the upper capacity available in a part-load segment by the parameter processes_rel_partload_capacity.

static constraint_production_limit_by_capacity_lower_rule(*model, node: int,*
process_production: str, year: int,
year_construction: int, time_slice:
str, vertex: int) → pyomo rule

This method constrains the used_production to the lower capacity available in a part-load segment by the parameter processes_rel_partload_capacity.

static constraint_production_potential_capacity_by_input_rule(*model, node: int,*
process_production: str,
year: int, year_construction:
int) → pyomo rule

This constraint sets a limit for the potential capacity of production processes. If existing_capacity_production is >0, this component already exists. Thus, new_capacity_production is fixed to existing_capacity_production in this rule.

static constraint_production_glovers_1a_rule(*model, node: int, process_production: str, year:*
int, year_construction: int, time_slice: str, vertex:
int) → pyomo rule

The glover constraints 1a, 1b, 2a, 2b model part-load behavior using the time-dependent auxiliary capacity xsi_production. Here: If used_production_binary = 0, then xsi_production = 0 OR If used_production_binary = 1, then xsi_production <= potential_capacity_production.

static constraint_production_glovers_1b_rule(*model, node: int, process_production: str, year:*
int, year_construction: int, time_slice: str, vertex:
int) → pyomo rule

The glover constraints 1a, 1b, 2a, 2b model part-load behavior using the time-dependent auxiliary capacity xsi_production. Here: If used_production_binary = 0, then xsi_production = 0 OR If used_production_binary = 1, then then minimal_capacity_production <= xsi_production.

static constraint_production_glovers_2a_rule(*model, node: int, process_production: str, year:*
int, year_construction: int, time_slice: str, vertex:
int) → pyomo rule

The glover constraints 1a, 1b, 2a, 2b model part-load behavior using the time-dependent auxiliary capacity xsi_production. Here: If used_production_binary == 0, then new_capacity_production-potential_capacity_production <= xsi_production <= potential_capacity_production If used_production_binary == 1, then xsi_production=new_capacity_production

static constraint_production_glovers_2b_rule(*model, node: int, process_production: str, year:*
int, year_construction: int, time_slice: str, vertex:
int) → pyomo rule

The glover constraints 1a, 1b, 2a, 2b model part-load behavior using the time-dependent auxiliary capacity xsi_production. Here: If used_production_binary == 0, then new_capacity_production-potential_capacity_production <= xsi_production <= potential_capacity_production If used_production_binary == 1, then xsi_production=new_capacity_production

static constraint_storage_level_limit_by_capacity_rule(*model, node: int, process_storage: str,*
year: int, year_construction: int,
time_slice: str) → pyomo rule

This method is used as rule for limiting the storage_level up to the existing (if exist) or new capacity.

static constraint_storage_level_limit_by_capacity_lower_rule(*model, node: int, process_storage: str, year: int, year_construction: int, time_slice: str*) → pyomo rule

This method is used as rule for setting the lower bound for to the existing (if exist) or new capacity * minial relative storage level.

static constraint_used_storage_limit_by_capacity_rule(*model, node: int, process_storage: str, direction_storage: str, year: int, year_construction: int, time_slice: str*) → pyomo rule

This abstract method is used as rule for limiting the output and input of storage processes.

The maximum flow out of or into a storage instance is limited by the available storage capacity multiplied with a flow-to-storage-capacity factor:

$$\text{storage usage} \leq \text{flow-to-storage-capacity factor} * (\text{existing capacity})$$

or:

$\text{storage usage} \leq \text{flow-to-storage-capacity factor} * (\text{new capacity})$
--

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **node** (*int*) – A node at which the pe.Constraint is active
- **process_storage** (*str*) – A storage process which is constrained
- **direction_storage** (*str*) – A direction of “withdraw” or “deposit” a product
- **year** (*int*) – A year at which the pe.Constraint is active
- **year_construction** (*int*) – The year of construction of a specific capacity
- **time_slice** (*str*) – A time_slice at which the pe.Constraint is active

static constraint_storage_potential_capacity_by_input_rule(*model, node: int, process_storage: str, year: int, year_construction: int*) → pyomo rule

This method is used as rule for limiting the capacity of the storage processes by the maximum potential, if the storage exists.

static constraint_storage_minimal_capacity_by_input_rule(*model, node: int, process_storage: str, year: int, year_construction: int*) → pyomo rule

This method is used as rule for setting a lower limit to a newly built storage process, if the storage unit is built.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **node** (*int*) – A node at which the pe.Constraint is active
- **process_storage** (*str*) – A storage process which is constrained
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_storage_deposit_withdraw_decision_rule(*model*, *node*: *int*, *process_storage*: *str*, *direction_storage*: *str*, *year*: *int*, *year_construction*: *int*, *time_slice*: *str*) → pyomo rule

This method restricts the storage usage to deposit or withdraw depending on the existing capacity/potential.

static constraint_storage_restrict_storing_per_timeslice_rule(*model*, *node*: *int*, *process_storage*: *str*, *year*: *int*, *year_construction*: *int*, *time_slice*: *str*) → pyomo rule

This method is used as a rule for deposit/withdrawal decision.

We can either withdraw or deposit in the storage, but never both at the same time in a time step.

static constraint_storage_level_rule(*model*, *node*: *int*, *process_storage*: *str*, *year*: *int*, *year_construction*: *int*, *time_slice*: *str*) → pyomo rule

This method is used as rule for the calculation of the storage level.

Since the storage should not act as a source or sink, this method calculates the storage level after each *time_slice* and couples it with the storage level at the beginning of the next time slice.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **node** (*int*) – A node at which the pe.Constraint is active
- **process_storage** (*str*) – A storage process which is constrained
- **year** (*int*) – A year at which the pe.Constraint is active
- **year_construction** (*int*) – The year of construction of a specific capacity
- **time_slice** (*str*) – A time_slice at which the pe.Constraint is active

static constraint_transshipment_limit_by_capacity_rule(*model*, *connection*: *int*, *process_transshipment*: *str*, *direction_transshipment*: *str*, *year*: *int*, *year_construction*: *int*, *time_slice*: *str*) → pyomo rule

This method is used rule for limiting the used transmission up to the sum of existing and new capacity.

The rule is applied to every year of operation AND every year of construction. This means that for every year the capacity of existing infrastructure is used as limit, creating a constraint for every year of construction as well. This way the operation can differentiate between older and newer capacities which might have different impacts. Additionally for the years of construction which also are investment years, the newly build capacities are added to the limit.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **process_transshipment** (*str*) – A transshipment process which is constrained
- **direction_transshipment** (*str*) – A transshipment direction which is constrained
- **year** (*int*) – A year at which the pe.Constraint is active
- **year_construction** (*int*) – The year of construction of a specific capacity
- **time_slice** (*str*) – A time_slice at which the pe.Constraint is active

```
static constraint_transshipment_potential_capacity_by_input_rule(model, connection: int,
                                                                process_transshipment:
                                                                str, year: int) → pyomo
                                                                rule
```

This method is used as rule for limiting the capacity of transshipment processes by limits from the input e.g. maximum potential.

Some transshipment technologies like pipelines have local capacity limits based on the availability of resources or regulatory restrictions. These limits are given to the optimization as parameter and can change by time.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **connection** (*int*) – A connection of the grid
- **process_transshipment** (*str*) – A transshipment process which is constrained
- **year** (*int*) – A year at which the pe.Constraint is active

```
static constraint_transmission_limit_by_capacity_positive_rule(model, connection: int,
                                                                year: int, time_slice: str)
                                                                → pyomo rule
```

This method is used as rule for limiting the load flow through a transmission power line.

The load flow needs to be limited by the available capacity, which is why this abstract rule exists and is used for a constraint in the base class for all optimizations. Although the available capacity can be defined differently, e.g.:

```
load flow <= installed capacity
```

or:

```
load flow <= installed capacity + capacity from switching + capacity from new_
↳ power lines
```

Because of this, the abstract pe.Constraint rule needs to be defined in a subclass.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **connection** (*int*) – A power line which is constrained
- **year** (*int*) – A year at which the pe.Constraint is active
- **time_slice** (*str*) – A time_slice at which the pe.Constraint is active

```
static constraint_transmission_limit_by_capacity_negative_rule(model, connection: int,
                                                                year: int, time_slice: str)
                                                                → pyomo rule
```

This method is used as rule for limiting the load flow through a transmission power line.

The load flow needs to be limited by the available capacity, which is why this abstract rule exists and is used for a constraint in the base class for all optimizations. Although the available capacity can be defined differently, e.g.:

```
load flow <= installed capacity
```

or:

```
load flow <= installed capacity + capacity from switching + capacity from new_
↳power lines
```

Because of this, the abstract `pe.Constraint` rule needs to be defined in a subclass.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **power_line** (*int*) – A power line which is constrained
- **year** (*int*) – A year at which the `pe.Constraint` is active
- **time_slice** (*str*) – A `time_slice` at which the `pe.Constraint` is active

```
static constraint_transmission_potential_capacity_by_input_rule_rule(model, connection:
int, process_transmission:
str, year: int) →
pyomo rule
```

This method is used as rule for limiting the capacity of transmission processes by limits from the input e.g. maximum potential.

Some transmission technologies have local capacity limits based on the availability of resources or regulatory restrictions. These limits are given to the optimization as parameter and can change by time.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **connection** (*int*) – A connection of the grid
- **process_transmission** (*str*) – A transmission process which is constrained
- **year** (*int*) – A year at which the `pe.Constraint` is active

```
static constraint_operational_nodal_impact_rule(model, node: int, impact_category: str, year:
int) → pyomo rule
```

This method is used as rule for the calculation of the operational impact vector summed over all processes at each node.

The operational impact of one node over all time slices is calculated in this method. By summing over the construction years of all capacities, the corresponding impact matrices from the actual construction year can be used. Furthermore the impact calculated for each time slice is multiplied with the weight in hours of a year the time slice has.

The operational impact of production processes equals the sum of the products of process usage and the corresponding impact matrix. The same method is used for calculating the impact of transshipment processes, but the resulting impact is shared equally between the corresponding nodes.

The operational impact of storage processes is calculated based on the **withdrawn** product flow.

For the electric transmission grid it is assumed that the operational impact is negligible.

Furthermore, a credit is given for sold products.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **node** (*int*) – A node at which the `pe.Constraint` is active
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the `pe.Constraint` is active

static constraint_operational_nodal_impact_limits_rule(*model, node: int, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the limitation of the operational impact vector summed over all processes at each node.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **node** (*int*) – A node at which the pe.Constraint is active
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_operational_impact_rule(*model, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the calculation of the operational impact vector summed over all processes and nodes.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_operational_impact_limits_rule(*model, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the limitation of the operational impact vector summed over all processes and nodes.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_invest_nodal_impact_rule(*model, node: int, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the calculation of the invest impact vector summed over all processes at each node. The method consider the existing and new capacities.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_invest_nodal_impact_limits_rule(*model, node: int, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the limitation of the invest impact vector summed over all processes at each node.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **node** (*int*) – A node at which the pe.Constraint is active
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_invest_impact_rule(*model, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the calculation of the invest impact vector summed over all processes and nodes

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_invest_impact_limits_rule(*model, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the limitation of the invest impact vector summed over all processes and nodes

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_total_nodal_impact_rule(*model, node: int, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the calculation of the total impact vector summed over all processes at each node.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **node** (*int*) – A node at which the pe.Constraint is active
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_total_nodal_impact_limits_rule(*model, node: int, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the limitation of the total impact vector summed over all processes at each node.

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **node** (*int*) – A node at which the pe.Constraint is active
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_total_impact_rule(*model, impact_category: str, year: int*) → pyomo rule

This method is used as rule for the calculation of the total impact vector summed over all processes and nodes

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static constraint_total_impact_limits_rule(*model*, *impact_category*: *str*, *year*: *int*) → pyomo rule
 This method is used as rule for the limitation of the total impact vector summed over all processes and nodes

Parameters

- **model** – The equivalent of “self” in pyomo optimization models
- **impact_category** (*str*) – An impact category which is assessed
- **year** (*int*) – A year at which the pe.Constraint is active

static objectiveRule(*model*)

This method is used as an pe.Objective for the optimization, using weight factors for all impact categories.

Parameters model – The equivalent of “self” in pyomo optimization models

_abc_impl = <_abc_data object>

4.7 secmod.setup module

secmod.setup.setup(*working_directory*: *pathlib.Path*, *reset*: *bool* = *False*, *download_ext_datapackages*: *bool* = *False*)

This methods sets up SecMOD for the first use in the working directory.

This method copies the embedded sample data to the working directory. Furthermore it downloads the external data packages for the first time. Last but not least it creates a python file and a batch file to easily start SecMOD.

Please note that you need to activate the download of external data packages here.

Args:

working_directory (Path): The folder which is used as working directory by SecMOD

reset (bool): Boolean value which decides whether an existing working directory is overwritten

download_external_datapackages (bool): Boolean value which decides whether external datapackages are downloaded

secmod.setup.copy_sample_data(*working_directory*: *pathlib.Path*)

This method copies sample data from the package to the working directory.

Parameters working_directory (Path) – The folder which is used as working directory by SecMOD

secmod.setup.create_startup_helper(*working_directory*: *pathlib.Path*)

This method creates startup helpers in the working directory.

The startup helpers created by this method allow to run SecMOD with one click.

Parameters working_directory (Path) – The folder which is used as working directory by SecMOD

secmod.setup.download_external_datapackages(*working_directory*: *pathlib.Path*)

TROUBLESHOOTING

SecMOD is programmed to handle most things internally, if possible. If you still encounter an error, SecMOD mostly provides you with an error message which should help to resolve the issue.

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

`secmod.classes`, 9
`secmod.data_preprocessing`, 24
`secmod.data_processing`, 25
`secmod.evaluation`, 25
`secmod.helpers`, 27
`secmod.optimization_MILP`, 28
`secmod.setup`, 37

Symbols

- `_abc_impl` (*secmod.classes.ConnectionProcess* attribute), 20
- `_abc_impl` (*secmod.classes.EcoinventImpacts* attribute), 18
- `_abc_impl` (*secmod.classes.ManualImpact* attribute), 19
- `_abc_impl` (*secmod.classes.NodalProcess* attribute), 20
- `_abc_impl` (*secmod.classes.Process* attribute), 19
- `_abc_impl` (*secmod.classes.ProcessImpacts* attribute), 16
- `_abc_impl` (*secmod.classes.ProductionProcess* attribute), 21
- `_abc_impl` (*secmod.classes.ProductionProcessEcoinvent* attribute), 23
- `_abc_impl` (*secmod.classes.ProductionProcessManual* attribute), 23
- `_abc_impl` (*secmod.classes.StorageProcess* attribute), 21
- `_abc_impl` (*secmod.classes.StorageProcessEcoinvent* attribute), 23
- `_abc_impl` (*secmod.classes.StorageProcessManual* attribute), 23
- `_abc_impl` (*secmod.classes.TransmissionProcess* attribute), 22
- `_abc_impl` (*secmod.classes.TransmissionProcessEcoinvent* attribute), 24
- `_abc_impl` (*secmod.classes.TransmissionProcessManual* attribute), 24
- `_abc_impl` (*secmod.classes.TransshipmentProcess* attribute), 22
- `_abc_impl` (*secmod.classes.TransshipmentProcessEcoinvent* attribute), 23
- `_abc_impl` (*secmod.classes.TransshipmentProcessManual* attribute), 23
- `_abc_impl` (*secmod.optimization_MILP.Optimization_MILP* attribute), 37
- `_get_cost_invest()` (*secmod.classes.ProcessImpacts* method), 15
- `_get_cost_non_served_demand()` (*secmod.classes.Product* method), 11
- `_get_cost_operation()` (*secmod.classes.ProcessImpacts* method), 15
- `_get_ecoinvent_process_impact()` (*secmod.classes.EcoinventImpacts* static method), 18
- `_get_invest_impact_limits()` (*secmod.classes.ImpactCategory* method), 13
- `_get_invest_nodal_impact_limits()` (*secmod.classes.ImpactCategory* method), 13
- `_get_investment_period_duration()` (*secmod.classes.ProcessImpacts* static method), 15
- `_get_objective_factor_impact()` (*secmod.classes.ImpactCategory* method), 14
- `_get_objective_factor_impact_overshoot()` (*secmod.classes.ImpactCategory* method), 14
- `_get_operational_impact_limits()` (*secmod.classes.ImpactCategory* method), 13
- `_get_operational_nodal_impact_limits()` (*secmod.classes.ImpactCategory* method), 13
- `_get_price_time_series()` (*secmod.classes.Product* method), 11
- `_get_process_by_voltage()` (*secmod.classes.TransmissionProcess* static method), 22
- `_get_process_connection_properties()` (*secmod.classes.TransmissionProcess* method), 22
- `_get_revenues_add_served_demand()` (*secmod.classes.Product* method), 11
- `_get_selling_price_time_series()` (*secmod.classes.Product* method), 11
- `get_single_combined_impact_matrix()` (*secmod.classes.ProcessImpacts* static method), 16
- `_get_total_impact_limits()` (*secmod.classes.ImpactCategory* method), 13
- `_get_total_nodal_impact_limits()` (*secmod.classes.ImpactCategory* method), 13
- `_isSubassembly()` (*secmod.classes.EcoinventImpacts* static method), 18
- `_recursively_get_impact()` (*secmod.classes.EcoinventImpacts* static method), 18

- `_selected` (*secmod.classes.Grid* attribute), 9, 10
 - `_translate_process_name_to_ecoinvent_identifier()` (*secmod.classes.EcoinventImpacts* static method), 16
 - `_translate_units_to_pint()` (*secmod.classes.EcoinventImpacts* static method), 17
- A**
- `AbstractFrame_evaluation` (class in *secmod.evaluation*), 25
 - `add_capacity()` (*secmod.classes.Process* method), 19
 - `adjustManualColumnWidths()` (*secmod.evaluation.AbstractFrame_evaluation* method), 26
- B**
- `browseFolder()` (*secmod.evaluation.SaveFrame* method), 27
- C**
- `calculate_distances()` (*secmod.classes.Grid* static method), 10
 - `calculate_power_line_power_limit()` (in module *secmod.classes*), 24
 - `calculate_power_line_resistance_per_unit()` (in module *secmod.classes*), 24
 - `calculate_power_line_susceptance_per_unit()` (in module *secmod.classes*), 24
 - `calculate_voltage_switch_power_limit()` (in module *secmod.classes*), 24
 - `calculate_voltage_switch_resistance_per_unit()` (in module *secmod.classes*), 24
 - `calculateFrameGeometry()` (*secmod.evaluation.AbstractFrame_evaluation* method), 26
 - `clear_filename_special_character()` (in module *secmod.helpers*), 27
 - `CONNECTION_NODES` (*secmod.classes.Grid* attribute), 10
 - `ConnectionProcess` (class in *secmod.classes*), 20
 - `connections` (*secmod.classes.Grid* property), 10
 - `constraint_invest_impact_limits_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 36
 - `constraint_invest_impact_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 35
 - `constraint_invest_nodal_impact_limits_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 35
 - `constraint_invest_nodal_impact_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 35
 - `constraint_operational_impact_limits_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 35
 - `constraint_operational_impact_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 35
 - `constraint_operational_nodal_impact_limits_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 34
 - `constraint_operational_nodal_impact_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 34
 - `constraint_product_balance_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 29
 - `constraint_production_glovers_1a_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 30
 - `constraint_production_glovers_1b_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 30
 - `constraint_production_glovers_2a_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 30
 - `constraint_production_glovers_2b_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 30
 - `constraint_production_limit_binaries_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 29
 - `constraint_production_limit_by_capacity_lower_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 30
 - `constraint_production_limit_by_capacity_upper_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 29
 - `constraint_production_potential_capacity_by_input_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 30
 - `constraint_restrict_export_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 29
 - `constraint_restrict_import_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 29
 - `constraint_storage_deposit_withdraw_decision_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 31
 - `constraint_storage_level_limit_by_capacity_lower_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 30
 - `constraint_storage_level_limit_by_capacity_rule()` (*secmod.optimization_MILP.Optimization_MILP* static method), 30

constraint_storage_level_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 32
 constraint_storage_minimal_capacity_by_input_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 31
 constraint_storage_potential_capacity_by_input_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 31
 constraint_storage_restrict_storing_per_timeslice_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 32
 constraint_total_impact_limits_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 36
 constraint_total_impact_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 36
 constraint_total_nodal_impact_limits_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 36
 constraint_total_nodal_impact_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 36
 constraint_transmission_limit_by_capacity_negative_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 33
 constraint_transmission_limit_by_capacity_positive_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 33
 constraint_transmission_potential_capacity_by_input_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 34
 constraint_transshipment_limit_by_capacity_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 32
 constraint_transshipment_potential_capacity_by_input_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 32
 constraint_usable_production_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 29
 constraint_used_storage_limit_by_capacity_rule() (sec-mod.optimization_MILP.Optimization_MILP static method), 31
 construction_years (secmod.classes.Process attribute), 19
 construction_years (secmod.classes.ProcessImpacts attribute), 14
 construction_years (secmod.classes.ProductionProcess attribute), 20
 construction_years (secmod.classes.StorageProcess attribute), 21
 construction_years (secmod.classes.TransmissionProcess attribute), 22
 construction_years (secmod.classes.TransshipmentProcess attribute), 22
 copy_sample_data() (in module secmod.setup), 37
 correct_time_stamp_of_timeseries() (in module secmod.helpers), 28
 create_directory() (in module sec-mod.data_preprocessing), 25
 create_startup_helper() (in module secmod.setup), 37
 createLegend() (sec-mod.evaluation.AbstractFrame_evaluation method), 26
D
 database (secmod.classes.EcoinventImpacts attribute), 16
 datapackage_update_available() (in module sec-mod.data_preprocessing), 24
 DEFAULT_COST (secmod.classes.Product attribute), 11
 DetailedFrame_evaluation (class in sec-mod.evaluation), 27
 deunitize_input_dictionary() (in module sec-mod.data_processing), 25
 DIRECTION_FACTOR_TRANSSHIPMENT (secmod.classes.TransshipmentProcess attribute), 22
 DIRECTIONS_STORAGE (secmod.classes.StorageProcess attribute), 21
 DIRECTIONS_TRANSSHIPMENT (secmod.classes.TransshipmentProcess attribute), 22
 disableEnableCheckbuttons() (sec-mod.evaluation.DetailedFrame_evaluation method), 27
 disableEnableRadiobuttons() (sec-mod.evaluation.MainFrame_evaluation method), 26
 download_datapackage() (in module sec-mod.data_preprocessing), 24
 download_external_datapackages() (in module sec-mod.setup), 37
 download_file() (in module sec-mod.data_preprocessing), 25
E
 ecoinvent_name (secmod.classes.ImpactCategory property), 13
 EcoinventImpacts (class in secmod.classes), 16

economic_period (*secmod.classes.ProcessImpacts* attribute), 14
evaluate() (in module *secmod.evaluation*), 27
F
fix_transmission_reference_node_phase_difference() (*secmod.optimization_MILP.Optimization_MILP* method), 28
fix_unneeded_variables() (*secmod.optimization_MILP.Optimization_MILP* method), 29
FRAMEWORK (*secmod.classes.ImpactCategory* attribute), 12
G
generate_input_dictionary() (in module *secmod.data_processing*), 25
get_availability_timeseries() (*secmod.classes.ProductionProcess* method), 20
get_combined_availability_timeseries() (*secmod.classes.ProductionProcess* static method), 20
get_combined_cornerpoints() (*secmod.classes.ProductionProcess* static method), 20
get_combined_demand_time_series() (*secmod.classes.Product* static method), 12
get_combined_existing_capacity() (*secmod.classes.Process* class method), 19
get_combined_flow_to_storage_capacity_factor() (*secmod.classes.StorageProcess* static method), 21
get_combined_gradient_technologymatrix() (*secmod.classes.ProductionProcess* method), 21
get_combined_impact_add_served_demand() (*secmod.classes.Product* static method), 12
get_combined_impact_matrix() (*secmod.classes.ProcessImpacts* class method), 15
get_combined_impact_non_served_demand() (*secmod.classes.Product* static method), 12
get_combined_invest_impact_limits() (*secmod.classes.ImpactCategory* static method), 13
get_combined_invest_nodal_impact_limits() (*secmod.classes.ImpactCategory* static method), 13
get_combined_lifetime_duration() (*secmod.classes.ProcessImpacts* class method), 16
get_combined_min_rel_storage_level_factor() (*secmod.classes.StorageProcess* static method), 21
get_combined_minimal_capacity() (*secmod.classes.ProductionProcess* class method), 21
get_combined_minimal_capacity() (*secmod.classes.StorageProcess* class method), 21
get_combined_objective_factor_impact() (*secmod.classes.ImpactCategory* static method), 14
get_combined_objective_factor_impact_overshoot() (*secmod.classes.ImpactCategory* static method), 14
get_combined_operational_impact_limits() (*secmod.classes.ImpactCategory* static method), 13
get_combined_operational_nodal_impact_limits() (*secmod.classes.ImpactCategory* static method), 13
get_combined_potential_capacity() (*secmod.classes.Process* class method), 19
get_combined_power_limit_per_circuit() (*secmod.classes.TransmissionProcess* static method), 22
get_combined_reference_product() (*secmod.classes.ProductionProcess* static method), 21
get_combined_rel_storage_loss_factor() (*secmod.classes.StorageProcess* static method), 22
get_combined_relative_partload_capacity() (*secmod.classes.ProductionProcess* method), 21
get_combined_required_nodal_secured_capacity() (*secmod.classes.Product* static method), 12
get_combined_required_total_secured_capacity() (*secmod.classes.Product* static method), 12
get_combined_safety_margin() (*secmod.classes.TransmissionProcess* static method), 23
get_combined_secured_capacity_factor() (*secmod.classes.NodalProcess* class method), 20
get_combined_storage_products() (*secmod.classes.StorageProcess* static method), 21
get_combined_technology_matrix() (*secmod.classes.NodalProcess* class method), 20
get_combined_technology_matrix_timeseries() (*secmod.classes.ProductionProcess* static method), 20
get_combined_total_impact_limits() (*secmod.classes.ImpactCategory* static method),

13
get_combined_total_nodal_impact_limits() (secmod.classes.ImpactCategory static method), 13
get_combined_transshipment_products() (secmod.classes.TransshipmentProcess class method), 22
get_combined_transshipment_efficiency() (secmod.classes.TransshipmentProcess class method), 22
get_connection_properties() (secmod.classes.TransmissionProcess static method), 22
get_cornerpoints() (secmod.classes.ProductionProcess method), 20
get_dataframe_from_parameter() (secmod.optimization_MILP.Optimization_MILP static method), 29
get_dataframe_from_result() (secmod.optimization_MILP.Optimization_MILP static method), 29
get_dimensionalities_from_input_dictionary() (in module secmod.data_processing), 25
get_existing_capacity() (secmod.classes.ConnectionProcess method), 20
get_existing_capacity() (secmod.classes.NodalProcess method), 19
get_existing_capacity() (secmod.classes.Process method), 19
get_flow_to_storage_capacity_factor() (secmod.classes.StorageProcess method), 21
get_gradient_technologymatrix() (secmod.classes.ProductionProcess method), 21
get_impact_add_served_demand() (secmod.classes.Product method), 12
get_impact_invest() (secmod.classes.EcoinventImpacts method), 17
get_impact_invest() (secmod.classes.ManualImpact method), 18
get_impact_invest() (secmod.classes.ProcessImpacts method), 15
get_impact_non_served_demand() (secmod.classes.Product method), 12
get_impact_operation() (secmod.classes.EcoinventImpacts method), 17
get_impact_operation() (secmod.classes.ManualImpact method), 19
get_impact_operation() (secmod.classes.ProcessImpacts method), 15
get_lifetime_duration() (secmod.classes.ProcessImpacts method), 16
get_list_of_active_impact_categories() (secmod.classes.ImpactCategory static method), 13
get_list_of_connection_ids() (secmod.classes.Grid static method), 10
get_list_of_names_of_active_impact_categories() (secmod.classes.ImpactCategory static method), 13
get_list_of_node_ids() (secmod.classes.Grid static method), 10
get_maximum_production_share() (secmod.classes.NodalProcess method), 20
get_min_rel_storage_level_factor() (secmod.classes.StorageProcess method), 21
get_minimal_capacity() (secmod.classes.ProductionProcess method), 21
get_minimal_capacity() (secmod.classes.StorageProcess method), 21
get_nodal_demand_time_series() (secmod.classes.Product method), 12
get_non_served_demand_from_result() (secmod.optimization_MILP.Optimization_MILP method), 29
get_numerical_property() (secmod.classes.Process method), 19
get_potential_capacity() (secmod.classes.ConnectionProcess method), 20
get_potential_capacity() (secmod.classes.NodalProcess method), 19
get_potential_capacity() (secmod.classes.Process method), 19
get_potential_capacity() (secmod.classes.TransmissionProcess method), 22
get_potential_capacity() (secmod.classes.TransshipmentProcess method), 22
get_reference_products() (secmod.classes.ProductionProcess method), 20
get_rel_storage_loss_factor() (secmod.classes.StorageProcess method), 21
get_relative_partload_capacity() (secmod.classes.ProductionProcess method), 21
get_required_nodal_secured_capacity() (secmod.classes.Product method), 12
get_required_total_secured_capacity() (secmod.classes.Product method), 12
get_rwth_colors() (in module secmod.helpers), 28

`get_secured_capacity_factor()` (*secmod.classes.NodalProcess* method), 19
`get_technology_matrix_timeseries()` (*secmod.classes.ProductionProcess* method), 20
`get_technologymatrix()` (*secmod.classes.NodalProcess* method), 20
`get_technologymatrix()` (*secmod.classes.ProductionProcess* method), 20
`get_technologymatrix()` (*secmod.classes.StorageProcess* method), 21
`get_transshipment_efficiency()` (*secmod.classes.TransshipmentProcess* method), 22
`get_used_capacity_from_result()` (*secmod.optimization_MILP.Optimization_MILP* method), 29
`getbalanceUnits()` (*secmod.evaluation.AbstractFrame_evaluation* method), 26
`getDataframeFromSelection()` (*secmod.evaluation.MainFrame_evaluation* method), 26
`getNonzeroIndizes()` (*secmod.evaluation.AbstractFrame_evaluation* method), 26
`getTitleString()` (*secmod.evaluation.AbstractFrame_evaluation* method), 26
`getTitleString()` (*secmod.evaluation.DetailedFrame_evaluation* method), 27
`getTitleString()` (*secmod.evaluation.MainFrame_evaluation* method), 27
Grid (*class in secmod.classes*), 9
grids (*secmod.classes.Grid* attribute), 9, 10
`groupBalanceByIndex()` (*secmod.evaluation.AbstractFrame_evaluation* method), 26
`groupBalanceByIndex()` (*secmod.evaluation.DetailedFrame_evaluation* method), 27

H

`highlightMouseSelection()` (*secmod.evaluation.AbstractFrame_evaluation* method), 26

I

impact_categories (*secmod.classes.ImpactCategory* attribute), 12

IMPACT_SOURCES (*secmod.classes.ProcessImpacts* attribute), 14
ImpactCategory (*class in secmod.classes*), 12
`instantiate_model()` (*secmod.optimization_MILP.Optimization_MILP* method), 28
interest_rate (*secmod.classes.ProcessImpacts* attribute), 14
invest_years (*secmod.classes.ProcessImpacts* attribute), 14
`is_setup()` (*in module secmod.data_preprocessing*), 25
`isInteger()` (*in module secmod.helpers*), 27

L

`load_all_processes()` (*in module secmod.data_processing*), 25
`load_ecoinvent_database()` (*secmod.classes.EcoinventImpacts* static method), 16
`load_ecoinvent_subassemblies()` (*secmod.classes.EcoinventImpacts* static method), 17
`load_grids_from_directory()` (*secmod.classes.Grid* class method), 10
`load_impact_categories_from_directory()` (*secmod.classes.ImpactCategory* class method), 13
`load_processes_from_directory()` (*secmod.classes.Process* class method), 19
`load_products_from_directory()` (*secmod.classes.Product* class method), 11
`load_raw_input()` (*in module secmod.data_processing*), 25
locations (*secmod.classes.ConnectionProcess* attribute), 20
locations (*secmod.classes.NodalProcess* attribute), 19
locations (*secmod.classes.Process* attribute), 19
`log_heading()` (*in module secmod.helpers*), 27

M

MainFrame_evaluation (*class in secmod.evaluation*), 26
MANUAL_IMPACT_CATEGORY_SELECTION (*secmod.classes.ImpactCategory* attribute), 12
ManualImpact (*class in secmod.classes*), 18
module
secmod.classes, 9
secmod.data_preprocessing, 24
secmod.data_processing, 25
secmod.evaluation, 25
secmod.helpers, 27
secmod.optimization_MILP, 28
secmod.setup, 37

N

name (*secmod.classes.Grid* property), 10
 name (*secmod.classes.ImpactCategory* property), 12
 name (*secmod.classes.ProcessImpacts* property), 14
 name (*secmod.classes.Product* property), 11
 NodalProcess (*class in secmod.classes*), 19
 nodes (*secmod.classes.Grid* property), 10

O

objectiveRule() (*secmod.optimization_MILP.Optimization_MILP* static method), 37
 openDetailedWindow() (*secmod.evaluation.MainFrame_evaluation* method), 26
 Optimization_MILP (*class in secmod.optimization_MILP*), 28

P

per_unit_base (*secmod.classes.TransmissionProcess* attribute), 22
 plotFigure() (*secmod.evaluation.AbstractFrame_evaluation* method), 26
 plotFigure() (*secmod.evaluation.MainFrame_evaluation* method), 26
 Process (*class in secmod.classes*), 19
 processes (*secmod.classes.ProcessImpacts* attribute), 14
 processes (*secmod.classes.ProductionProcess* attribute), 20
 processes (*secmod.classes.StorageProcess* attribute), 21
 processes (*secmod.classes.TransmissionProcess* attribute), 22
 processes (*secmod.classes.TransshipmentProcess* attribute), 22
 ProcessImpacts (*class in secmod.classes*), 14
 Product (*class in secmod.classes*), 11
 ProductionProcess (*class in secmod.classes*), 20
 ProductionProcessEcoinvent (*class in secmod.classes*), 23
 ProductionProcessManual (*class in secmod.classes*), 23
 products (*secmod.classes.Product* attribute), 11
 PRODUCTS (*secmod.classes.TransmissionProcess* attribute), 22

R

reference_node (*secmod.classes.TransmissionProcess* attribute), 22
 run() (*secmod.optimization_MILP.Optimization_MILP* method), 28

S

SaveFrame (*class in secmod.evaluation*), 27
 savePlotData() (*secmod.evaluation.AbstractFrame_evaluation* method), 26
 saveSelection() (*secmod.evaluation.SaveFrame* method), 27
 secmod.classes module, 9
 secmod.data_preprocessing module, 24
 secmod.data_processing module, 25
 secmod.evaluation module, 25
 secmod.helpers module, 27
 secmod.optimization_MILP module, 28
 secmod.setup module, 37
 select() (*secmod.classes.Grid* method), 11
 selected() (*secmod.classes.Grid* static method), 11
 selectPlotType() (*secmod.evaluation.AbstractFrame_evaluation* method), 26
 setColors() (*secmod.evaluation.AbstractFrame_evaluation* method), 26
 setup() (*in module secmod.setup*), 37
 setup_construction_years() (*secmod.classes.Process* class method), 19
 setup_construction_years() (*secmod.classes.ProcessImpacts* class method), 16
 setup_distances() (*secmod.classes.Grid* method), 10
 setup_years() (*secmod.classes.ProcessImpacts* method), 16
 setupConstraints() (*secmod.optimization_MILP.Optimization_MILP* method), 28
 setupFigure() (*secmod.evaluation.AbstractFrame_evaluation* method), 25
 setupObjective() (*secmod.optimization_MILP.Optimization_MILP* method), 28
 setupParameters() (*secmod.optimization_MILP.Optimization_MILP* method), 28
 setUpRadiobuttons() (*secmod.evaluation.MainFrame_evaluation* method), 26
 setupSets() (*secmod.optimization_MILP.Optimization_MILP* method), 28

`setupVariables()` (*secmod.optimization_MILP.Optimization_MILP method*), 28

`showTable()` (*secmod.evaluation.AbstractFrame_evaluation method*), 26

`showTable()` (*secmod.evaluation.DetailedFrame_evaluation method*), 27

`start_evaluation()` (*in module secmod.evaluation*), 27

`startPlot()` (*secmod.evaluation.AbstractFrame_evaluation method*), 25

`STORAGE_LEVEL_FACTOR` (*secmod.classes.StorageProcess attribute*), 21

`StorageProcess` (*class in secmod.classes*), 21

`StorageProcessEcoinvent` (*class in secmod.classes*), 23

`StorageProcessManual` (*class in secmod.classes*), 23

`subassemblies` (*secmod.classes.EcoinventImpacts attribute*), 16

T

`toggleSaveButton()` (*secmod.evaluation.SaveFrame method*), 27

`TransmissionProcess` (*class in secmod.classes*), 22

`TransmissionProcessEcoinvent` (*class in secmod.classes*), 23

`TransmissionProcessManual` (*class in secmod.classes*), 24

`TransshipmentProcess` (*class in secmod.classes*), 22

`TransshipmentProcessEcoinvent` (*class in secmod.classes*), 23

`TransshipmentProcessManual` (*class in secmod.classes*), 23

U

`unitize_dataframe()` (*in module secmod.helpers*), 27