City Energy Analyst Documentation

Release 2.1

Architecture and Building Systems

April 11, 2017

Contents:

CHAPTER 1

Getting started

The City Energy Analyst 2.1 is stored in a public repository in Github under the name of CEAforArcGIS.

Folder Structure

The repository contains the following folders:

- cea
  - analysis
  - databases
  - demand
  - geometry
  - optimization
  - plots
  - resources
  - technologies
  - utilities
  - examples
  - tests
- docs
- bin
- euler
- setup
- tests

cea

Contains the source code (human readable) of the different modules of CEA. This is the core of CEA and is divided in the next sub-folder structure:

cea/analysis

Contains scripts to analyze the results of all simulations in CEA.

cea/databases

Contains default data of weather, technology, occupancy, costs etc.,

cea/demand

Contains scripts to calculate the demand of energy in buildings

cea/geometry

Contains scripts to manipulate and read building geometry.

cea/optimization

Contains scripts aiming to optimize the energy system of a district.

cea/plots

Contains scripts to plot information.

cea/resources

Contains scripts to analyze the potential of energy sources.

cea/technologies

Contains scripts for the simulation of different technologies.

cea/utilities

Contains functions needed by any other script frequently.

cea/examples

Contains an open source case study to test the cea.

cea/tests

Contains unitest data and scripts necessary ot run our automatic code checker in Github.

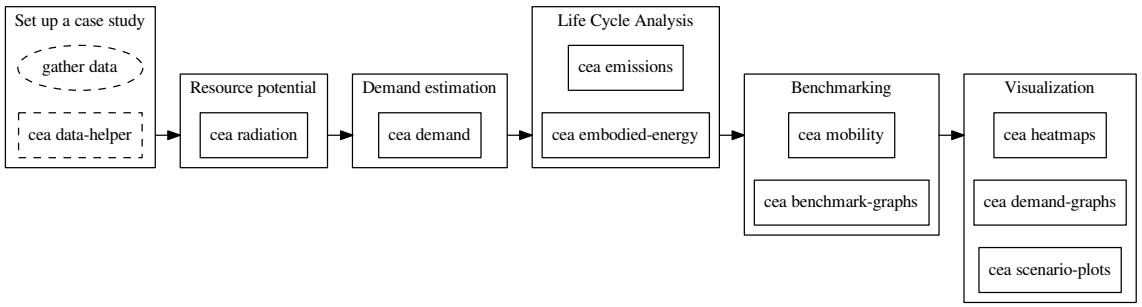
docs

Contains the developers manual of the CEA.

euler

Contains source code needed to connect to the cluster of 50K cores called Euler at ETH Zurich (Only researchers at ETH Zurich can use this).

CEA workflow



The main workflow of CEA is:

Set up a case study

If you want to run a case study different from those available in the 'examples' folder. This step entails preparing a case study to have:

1. the same folder structure as one of our case studies in the 'examples' folder.
2. the exact number, names and attributes tables of the input shapefiles.
3. the exact name and content of the digital elevation model of the terrain.

The CEA users manual includes a great deal of hints to both gather and format these data for CEA using ArcGIS or any other geographical information system.

Further, the command `cea data-helper` can apply archetype information to a subset of the input files to generate the others.

Resource potential

Once a case study is created the solar radiation incident in every surface is calculated. - run `cea radiation`

Demand estimation

Calculate the demand of energy services for every building in the area. - run `cea demand`

Life Cycle Analysis

Calculate the emissions and primary energy needed due to the construction, operation and dismantling of buildings in the area.

- run `cea emissions`
- run `cea embodied-energy`

Benchmarking

In case you have more than one scenario inside the case study, this step calculates targets of performance according to the 2000-Watt Society approach. The approach also calculates the LCA of vehicles in the area.

- run `cea mobility`
- run `cea benchmark`

Visualization

There are different ways to visualize and plot all the raw data described until now. You can either map it using ArcGIS (we expect you to know how through our user's manual), or run the different scripts we included for this.

- for heatmaps of demand or LCA run `cea heatmaps` - currently, you will need ArcGIS for this.
- for plots of demand run `cea demand-graphs`
- for plots of benchmarking run `cea scenario plots`

CHAPTER 2

Installation guide

The version 2.1 of the City Energy Analyst is dependent on ArcGIS 10.4 for its visuals. As such it is restricted to Windows-based OS's for some features.

Installation on Windows

Installing the CEA on Windows is easiest with Anaconda (or Miniconda) as the CEA depends on the geopandas module.

Installation follows the following basic steps:

1. create a conda environment and activate it (optional)
2. `conda install -c conda-forge geopandas`
3. `pip install cityenergyanalyst`
4. `cea install-toolbox` (installs the CEA as an ArcGIS 10.4 toolbox)

The following subsections contain additional information for variations on the above theme.

Creating a conda environment

Creating a conda environment is an optional step, but probably a good habit to get into: This creates a python environment separate from your other python environments - that way, version mismatches between packages don't bleed into your other work. Follow these steps to create a new conda environment for the CEA:

1. `conda create -n cea python=2.7` (you can use any name, when creating an environment - "cea" is just an example)
2. `activate cea` (do this before any cea-related work on the commandline)
3. work with CEA, python is now set up to use your conda environment
4. `deactivate cea` (you can also just close the shell)

Connecting to Arcpy

The command `cea install-toolbox` (see step 4 in the basic installation steps above) attempts to connect the CEA with the ArcGIS environment. You should not need to do anything else.

If, however, you get error messages like `ImportError: no module named arcpy` you can check your home directory for a file called `cea_arcgis.pth` containing these three lines:

```
C:\Program Files (x86)\ArcGIS\Desktop10.4\bin
C:\Program Files (x86)\ArcGIS\Desktop10.4\arcpy
C:\Program Files (x86)\ArcGIS\Desktop10.4\Scripts
```

Edit these folders to point to the appropriate ArcGIS folders as documented in the ArcGIS manuals.

Installation from GitHub repository

If you'd prefer to track the newest version of the City Energy Analyst, replace step 3 in the basic installation steps above with a git clone of the CEA GitHub repository and run:

```
python setup.py install
```

to install the CEA, including the `cea` tool. Note, you will still need to follow the guide above to create a conda environment and install geopandas.

Installation of the development environment

When installing the City Energy Analyst for development, clone the repository to your computer and run:

```
python setup.py develop
```

This will install the `cea` tool to your path and set it up to use the version in the repository. Note, you will need to follow the guide above to create a conda environment and install geopandas.

Recommended software

- GitHub Desktop (or your favorite `git`)
- Anaconda distribution (x86) - other pythons can work, but this is really recommended
- PyCharm community edition - you can use your own favorite editor, but this is what we use
- Git Large File Storage - only for working with the reference case repository (you need to be a core developer to have access to the private reference case repository)

Setting up PyCharm

The developer team uses PyCharm Community edition as default. Here are the instructions to get PyCharm up and running:

1. Access PyCharm and open project CEAforArcGIS
2. Open File>Settings>Project:CEAforArcGIS>Project Interpreter>Project Interpreter
3. Click on settings>addlocal and point to the location of your python installation in the environment `cea`. It should be located in something like `C:\Users\your_name\Anaconda2\envs\cea/python.exe`
4. Click apply changes and your are done!

To set the custom dictionary used in PyCharm, do:

1. Open File>Settings>Editor>Spelling

2. Open the Dictionaries tab
3. Add a new Custom Dictionaries Folder
4. Select the root source folder for CEAforArcGIS. It should be located in something like `C:\Users\your_name\Documents\GitHub\CEAforArcGIS`.
5. Click "Apply".

Installation on the Euler cluster

It is possible to install the CEA on the Euler cluster by following the following guide: `installation-on-euler`.

CHAPTER 3

License

The `core` of the City Energy Analyst is registered under The MIT License (MIT).

for V0.3c

Copyright (c) 2016, 'Jimeno A. Fonseca <http://www.fcl.ethz.ch/person/dr-jimeno-a-fonseca/>'__, 'Daren Thomas <http://www.systems.arch.ethz.ch/about-us/team/team-zurich/daren-thomas.html>'__, 'Gabriel Happle <http://www.fcl.ethz.ch/person/gabriel-happle/>'__, 'Shanshan Hsieh <http://www.fcl.ethz.ch/person/hsieh-shan-shan/>'__, 'Martin Mosteiro <http://www.systems.arch.ethz.ch/about-us/team/team-zurich/martin-mosteiro-romero.html>'__, 'Amr Elesawy <http://www.systems.arch.ethz.ch/about-us/team/team-zurich/amr-elesawy.html>'__, 'Architecture and Building Systems - ETH Zurich <http://www.systems.arch.ethz.ch>'__

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

for V0.1

Copyright (c) 2015, 'Jimeno A. Fonseca <http://www.fcl.ethz.ch/person/dr-jimeno-a-fonseca/>'__, 'Daren Thomas <http://www.systems.arch.ethz.ch/about-us/team/team-zurich/daren-thomas.html>'__, 'Architecture and Building Systems - ETH Zurich <http://www.systems.arch.ethz.ch>'__

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

CHAPTER 4

Disclaimer

The City Energy Analyst is provided “as is”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

CHAPTER 5

Architecture

The architecture of the CEA is still a bit in flux, but some main components have already been developed and will be explained in this chapter. The following figure shows a high-level view of the main components of the CEA:

|CEA architecture objects|

Demand calculation

At the core of the CEA is the demand calculation. The demand calculation retrieves inputs from the scenario folder and stores outputs back to the scenario folder. A preprocessing step can be used to add archetype data to a scenario as a first guess of certain parameters.

The demand calculation uses a special variable called `tsd` to store information about the timestep data during the calculation of thermal loads for each building. The data structure used is a simple python dictionary of NumPy arrays. Each of these arrays has the length 8760, to total number of hours of the year. The keys of the `tsd` dictionary are the names of the state variables of the simulation.

The demand calculation also uses a variable `bpr` to store building properties of a single building.

InputLocator

The `cea.inputlocator.InputLocator` class encapsulates the code for creating paths for input and output to the archetypes and the contents of the scenario (input and output files). An instance of this class is found in most of the code and is always named `locator`, unless multiple `InputLocator` objects are used, e.g. for comparing scenarios.

Each method of the `locator` starts with `get_*` and returns a string containing the full path to the resource requested. These `get_*` methods should be the only way to obtain file- and folder names in the CEA - files and folders should especially not be concatenated with strings and backslashes (`\`). Instead, new paths should be introduced as methods of the `InputLocator` class.

One of the main benefits of doing this is that it makes documentation of what files are read / written by what module of the CEA easier. The `funcionlogger` module can be used to trace these calls for generating documentation.

The private method `_ensure_folder(*paths)` is used to join path components and at the same time ensure that the folders are present in the scenario folder, creating them with `os.makedirs` if necessary.

NOTE: The list of `get_*` methods is getting very long. We might end up creating a namespace hierarchy for grouping related paths together.

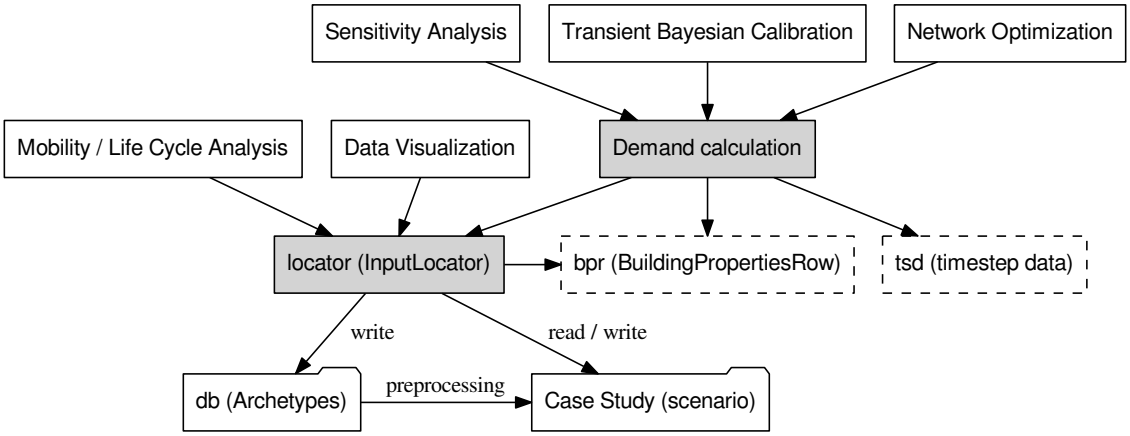
Analysis and Visualization

Separate modules exist for analyzing different aspects of a scenario. Some of the analysis modules operate only on the input data (LCA for embedded emissions, mobility) and others operate on the output of the demand module (LCA for emissions due to operation). These modules are grouped in the folder `cea/analysis`.

The folder `cea/plots` contains modules for plotting outputs of the calculations.

"Higher order" modules

Some of the modules in the CEA use the demand calculation to calculate variants of a scenario. This includes the sensitivity analysis, the calibration and the network optimization code. All these modules call the demand calculation as part of their process.



CHAPTER 6

User Interfaces

The CEA code exposes multiple interfaces as an API:

- CLI (Command Line Interface) - each module in the CEA implements a CLI for calling it from the command line.
- ArcGIS - the CEA implements an ArcGIS Toolbox (in the folder `cea/GUI`) to run the modules from within ArcScene 10.4
- euler - a set of scripts for running the CEA sensitivity analysis on the ETH Euler cluster is provided in the folder `euler` and can be used as a starting point for running the analysis on similar clusters and / or linux machines.

The Command Line Interface

The most portable way to interact with the CEA is via the CLI. Type the following command in your shell to see the list of commands available:

```
> cea --help
usage: cea [-h] [-s SCENARIO]
       {demand,demand-helper,emissions,embodied-energy,mobility,
       benchmark-graphs,weather-files,weather-path,
       locate,demand-graphs, scenario-plots,latitude,longitude,
       radiation,install-toolbox, heatmaps}
       ...

positional arguments:
{demand,demand-helper,emissions,embodied-energy,mobility,
benchmark-graphs,weather-files,weather-path,locate,demand-graphs,
scenario-plots,latitude,longitude,radiation,install-toolbox,heatmaps}

optional arguments:
-h, --help            show this help message and exit
-s SCENARIO, --scenario SCENARIO
                      Path to the scenario folder
```

Most commands work on a scenario folder, provided with the global option `-s` and defaults to the current directory - if you `cd` to the scenario folder, you can ommit the `-s` option.

Each sub-command (one of `demand`, `demand-helper` etc.) may provide additional arguments that are required to run that sub-command. For example, the `embodied-energy` sub-command has an (optional) option `--year-to-calculate`

```
> cea embodied-energy --help
usage: cea embodied-energy [-h] [--year-to-calculate YEAR_TO_CALCULATE]

optional arguments:
-h, --help            show this help message and exit
--year-to-calculate YEAR_TO_CALCULATE
                      Year to calculate for (default: 2017)
```

As you can see, you can get help on any sub-command by typing `cea SUBCOMMAND --help`. This will list the expected arguments as well as the default values used.

The sub-commands can be grouped into three groups: Main commands that work on a single scenario, main commands that work on multiple scenarios and auxillary commands. Main commands that work on multiple scenarios have a `--scenarios` option for specifying the scenarios to work on - these commands ignore the `-s` global option for specifying the scenario folder.

Main commands

Main commands that work on a single scenario are:

- demand (calculate the demand load of a scenario)
- demand-helper (apply archetypes to a scenario)
- emissions (calculate emissions due to operation)
- embodied-energy (calculate embodied energy)
- mobility (calculate emissions due to mobility)
- demand-graphs (create graphs for demand output variables per building)
- radiation (create radiation data as input to the demand calculation)
- heatmaps (create heatmaps based on demand and emissions output)
- extract-reference-case (extracts a sample reference case that can be used to test-drive the CEA)

Main commands that work on multiple scenarios:

- benchmark-graphs (create graphs for the 2000 Watt society benchmark for multiple scenarios)
- scenario-plots (plots various scenarios against each other)

Auxillary commands

- weather-files (list the weather names shipped with the CEA)
- weather-path (given a weather name {see above} return the path to the file)
- latitude (try to guess the latitude of a scenario based on it's building geometry shapefile)
- longitude (try to guess the longitude of a scenario based on it's building geometry shapefile)
- install-toolbox (install the ArcGIS interface)
- locate (gives access to the InputLocator class for finding paths)

Commands for developers

- test (runs a set of tests - requires access to the private repository *cea-reference-case*)

To run the `cea test` tool, you will need to provide authentication for the *cea-reference-case* repository. The options `--user` and `--token` should be set to your GitHub username and a Personal Access Token. These will be stored in your home folder in a file called `cea_github.auth`. The first line is the username, the second contains the token. See this page on how to create such a token: https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/ In the scopes section, select "repo (Full control of private repositories)" for the token.

Planned interfaces

The following interfaces are planned for the near future:

- Rhino/Grasshopper - provide a set of components for running CEA modules inside Grasshopper
- VisTrails - provide a set of VisTrails modules for running the CEA

Further ideas

Other possible interfaces include:

- Kepler - a set of modules for the Kepler Scientific Workflow software
- REST - a REST server for executing the CEA in the cloud

CHAPTER 7

Databases

The current version of the CEA uses the next three databases:

Weather data

Weather data files are stored as `energyPlus:.epw` inside the folder `..cea/DB/Weather`. The files can be downloaded for more than 2100 locations in the planet. You can also create your own `.epw` file with tools such as Meteonorm. The City Energy Analyst V1.0b ships with hourly data of a Typical Meteorological Year for the locations of Zurich, Zug and Singapore.

Archetypes

An Archetype is a database relating the typology of buildings to historical data about their construction, energy systems, operation etc. The function of this database is to help the user to retrieve inputs for his analysis when data is scarce.

We can call it as default data for any type of simulation. Bear in mind that this data is highly dependent of local context. The City Energy Analyst V1.0b ships with data for the Swiss-European context as a default for up to 15 types of building typologies. Each building typology is classified according to a year of construction and/or renovation and a type of occupancy in the building.

Note: You are encouraged to build an Archetypes database for your local area. If you decide to use the existing please remember to acknowledge the authors:

- Fonseca, J. A., & Schlueter, A. (2015). Integrated model for characterization of spatiotemporal building energy consumption patterns in neighborhoods and city districts. Applied Energy, 142, 247–265.
- *Thoma, E., J. A. Fonseca, and A. Schlueter. "Estimation of base-values for Grey Energy, Primary Energy, Global Warming Potential (GWP 100A) and Umweltbelastungspunkte (UBP 2006) for Swiss constructions from before 1920 until today." Contemporary Urban Issues. 2014. 17. '*

Conversion Systems

Data about carbon accounting of building energy supply systems are stored in `..db/CH/Systems/supply_systems.xls`. The data are classified in a spreadsheet according the next type of energy services attended.

- `dhw`: domestic hot water.

- `heating`: Single Dwelling Unit.
- `cooling`: space cooling.
- `electricity`: space cooling.

The spreadsheet `ARCHITECTURE` relates architecture properties of buildings to a building category. It contains the next attributes:

Variable	Type	Unit	Description	Valid Values	Ref.
Code	string	-	code of type of main energy supply system (e.g., solar collector, natural gas in district heating network etc.)	T0,T1,T2,...Tn	•
PEN	float	MJ-oil/m2.yr	Non-renewable Primary energy factor (only fossil fuels contribution)	(0.0.....4)	[1]
CO2	float	kg CO2-eq/m2.yyr	CO22 equivalent emissions factor (only fossil fuels contribution)	(0.0.....0.2)	[1]

References

[1] Schweizerischer Ingenieur- und Architektenverein (SIA). (2006). Standard-Nutzungsbedingungen für die Energie- und Gebäudetechnik Merkbatt 2024. Zürich.

CHAPTER 8

cea package

Subpackages

cea.analysis package

Subpackages

cea.analysis.sensitivity package

Submodules

cea.analysis.sensitivity.sensitivity_demand_analyze module

cea.analysis.sensitivity.sensitivity_demand_count module Return the count a list of samples in a specified folder as input for the demand sensitivity analysis.

This reads in the *samples.npy* file produced by the script *sensitivity_demand_samples.py* and prints out the number of samples contained. This can be used for scripting the demand simulations with a load sharing facility system like the Euler cluster.

cea.analysis.sensitivity.sensitivity_demand_count.**count_samples**(*samples_path*)

Read in the *samples.npy* numpy array from disk in the *samples_path* and report the row count (each row in the array is a sample to simulate for either the morris or the sobol method.

Parameters **samples_path** (*str*) – path to folder with the samples - see sensitivity_demand_samples.py

Returns number of samples in the samples folder.

cea.analysis.sensitivity.sensitivity_demand_samples module

cea.analysis.sensitivity.sensitivity_demand_simulate module

cea.analysis.sensitivity.sensitivity_optimization module
Note: documentation pending

Module contents Sensitivity analysis for demand_main.py

These scripts use the morris algorithm (morris 1991)(campologo 2011) and Sobol Algorithm Sltalli 20110 to screen the most sensitive variables of a selection of parameters of the CEA.

The morris method serves to do basic screening of input variables and it is based on the family of One-at-a-time screening methods (OAT). morris provides a ranking but not a quantitative measure of the importance of each parameter.

The Sobol method serves for a complete sensitivity analysis of input variables. It is based on variance methods.

Submodules

cea.analysis.benchmark module

cea.analysis.embodied module

cea.analysis.mcda module

cea.analysis.mobility module

cea.analysis.operation module

Module contents

cea.databases package

Subpackages

cea.databases.CH package

Module contents

Module contents

cea.demand package

Subpackages

cea.demand.calibration package

Subpackages

cea.demand.calibration.clustering package

Submodules

cea.demand.calibration.clustering.clustering_main module

cea.demand.calibration.clustering.sax module Symbolic Aggregate approximation (SAX) in python. Based on the paper "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms" by J. Lin, E, Keogh, S. Lonardi & B. Chiu. 2003.

Adapted from work og N. Hoffman published under MIT license. The original code can be found in https://github.com/nphoff/saxpy

class cea.demand.calibration.clustering.sax.**SAX**(*word_size=8*, *alphabet_size=7*, *epsilon=1e-06*)

Bases: object

This class is for computing common things with the Symbolic Aggregate approXimation method. In short, this translates a series of data to a string, which can then be compared with other such strings using a lookup table.

alphabetize(*paaX*)
Converts the Piecewise Aggregate Approximation of x to a series of letters.

batch_compare(*xStrings*, *refString*)

build_letter_compare_dict()
Builds up the lookup table to determine numeric distance between two letters given an alphabet size. Entries for both 'ab' and 'ba' will be created and will have identical values.

compare_letters(*la*, *lb*)
Compare two letters based on letter distance return distance between

compare_strings(*sA*, *sB*)
Compares two strings based on individual letter distance Requires that both strings are the same length

normalize(*x*)
Function will normalize an array (give it a mean of 0, and a standard deviation of 1) unless it's standard deviation is below epsilon, in which case it returns an array of zeros the length of the original array.

set_scaling_factor(*scalingFactor*)

set_window_size(*windowSize*)

sliding_window(*x*, *num_subsequences=None*, *overlapping_fraction=None*)

to_PAA(*x*)
Function performs Piecewise Aggregate Approximation on data set, reducing the dimension of the dataset x to w discrete levels. returns the reduced dimension data set, as well as the indicies corresponding to the original data for each reduced dimension

to_letter_representation(*x*)
Function takes a series of data, x, and transforms it to a string representation

cea.demand.calibration.clustering.sax_optimization module

Module contents

Module contents

cea.demand.preprocessing package

Submodules

cea.demand.preprocessing.properties module building properties algorithm

cea.demand.preprocessing.properties.**calc_category**(*a*, *x*, *y*)

cea.demand.preprocessing.properties.**calc_comparison**(*array_min*, *array_max*)

cea.demand.preprocessing.properties.**calc_mainuse**(*uses_df*, *uses*)

cea.demand.preprocessing.properties.**get_database**(*path_database*, *sheet*)

cea.demand.preprocessing.properties.**properties**(*locator*, *prop_architecture_flag*, *prop_hvac_flag*, *prop_comfort_flag*, *prop_internal_loads_flag*)

algorithm to query building properties from statistical database Archetypes_HVAC_properties.csv. for more info check the integrated demand model of Fonseca et al. 2015. Appl. energy.

Parameters

- **locator** (InputLocator) – an InputLocator instance set to the scenario to work on
- **prop_architecture_flag** (*boolean*) – if True, get properties about the construction and architecture.
- **prop_comfort_flag** (*boolean*) – if True, get properties about thermal comfort.
- **prop_hvac_flag** (*boolean*) – if True, get properties about types of HVAC systems, otherwise False.
- **prop_internal_loads_flag** (*boolean*) – if True, get properties about internal loads, otherwise False.

The following files are created by this script, depending on which flags were set:

•**building_HVAC: .dbf** describes the queried properties of HVAC systems.

•**architecture.dbf** describes the queried properties of architectural features

•**building_thermal: .shp** describes the queried thermal properties of buildings

•**indoor_comfort.shp** describes the queried thermal properties of buildings

cea.demand.preprocessing.properties.**run_as_script**(*scenario_path=None*, *prop_thermal_flag=True*, *prop_architecture_flag=True*, *prop_hvac_flag=True*, *prop_comfort_flag=True*, *prop_internal_loads_flag=True*)

Run the properties script with input from the reference case and compare the results. This ensures that changes made to this script (e.g. refactorings) do not stop the script from working and also that the results stay the same.

Module contents

Submodules

cea.demand.airconditioning_model module

Contains debugged version of HVAC model from *[Kämpf2009]*

- originally coded by J. Fonseca
- debugged by G. Happle

Note: this is not really true anymore. The procedure now is just loosely based on *[Kämpf2009]*.

cea.demand.airconditioning_model.**calc_hvac_cooling**(*tsd*, *hoy*, *gv*)
Calculate AC air mass flows, energy demand and temperatures For the cooling case for AC systems with demand controlled ventilation air flows (mechanical ventilation) and conditioning of recirculated air (outdoor air flows are not modified)

Parameters

- **tsd** (*Dict[str, numpy.ndarray[numpy.float64]]*) – time series data dict
- **hoy** (*int*) – time step
- **gv** (cea.globalvar.GlobalVariables) – global variables

Returns AC air mass flows, energy demand and temperatures for the cooling case

Return type Dict[str, numpy.float64]

cea.demand.airconditioning_model.**calc_hvac_heating**(*tsd*, *hoy*, *gv*)
Calculate AC air mass flows, energy demand and temperatures for the heating case For AC system with demand controlled ventilation air flows (mechanical ventilation) and conditioning of recirculated air (outdoor air flows are not modified)

Parameters

- **tsd** (*Dict[str, numpy.ndarray[numpy.float64]]*) – time series data dict
- **hoy** (*int*) – time step
- **gv** (cea.globalvar.GlobalVariables) – global variables

Returns AC air mass flows, energy demand and temperatures for the heating case

Return type Dict[str, numpy.float64]

cea.demand.airconditioning_model.**calc_w3_cooling_case**(*t5*, *w2*, *t3*, *w5*)
Algorithm 2 Determination of the room's supply moisture content (w3) for the cooling case from Kaempf's HVAC model for non-evaporative cooling

Source: *[Kämpf2009]*

Parameters **t5** (*numpy.float64*) – temperature 5 in (°C)

:param w2 : moisture content 2 in (kg/kg dry air) :type w2: numpy.float64

Parameters

- **t3** (*numpy.float64*) – temperature 3 in (°C)
- **w5** (*numpy.float64*) – moisture content 5 in (kg/kg dry air)

Returns w3, moisture content of HVAC supply air in (kg/kg dry air)

Return type numpy.float64

cea.demand.airconditioning_model.**calc_w3_heating_case**(*t5*, *w2*, *w5*, *t3*, *gv*)
Algorithm 1 Determination of the room's supply moisture content (w3) for the heating case from Kaempf's HVAC model *[Kämpf2009]*

Parameters

- **t5** (*numpy.float64*) – temperature 5 in (°C)
- **w2** (*numpy.float64*) – moisture content 2 in (kg/kg dry air)
- **w5** (*numpy.float64*) – moisture content 5 in (kg/kg dry air)
- **t3** (*numpy.float64*) – temperature 3 in (°C)
- **gv** (cea.globalvar.GlobalVariables) – global variables

Returns w3, moisture content of HVAC supply air in (kg/kg dry air)

Return type numpy.float64

cea.demand.control_heating_cooling_systems module

cea.demand.control_heating_cooling_systems.**cooling_system_is_ac**(*bpr*)
determines whether a building's heating system is ac or not

Param bpr: building properties row object

Return type bool

cea.demand.control_heating_cooling_systems.**has_cooling_system**(*bpr*)
determines whether a building has a cooling system installed or not

Parameters **bpr** – building properties row object

Return type bool

cea.demand.control_heating_cooling_systems.**has_heating_system**(*bpr*)
determines whether a building has a heating system installed or not

bpr : building properties row object

Returns

Return type bool

cea.demand.control_heating_cooling_systems.**heating_system_is_ac**(*bpr*)
determines whether a building's heating system is ac or not

Parameters **bpr** – building properties row object

Return type bool

cea.demand.control_heating_cooling_systems.**is_active_cooling_system**(*bpr*, *tsd*, *t*)

cea.demand.control_heating_cooling_systems.**is_active_heating_system**(*bpr*, *tsd*, *t*)

cea.demand.control_ventilation_systems module

cea.demand.control_ventilation_systems.**has_mechanical_ventilation**(*bpr*)

cea.demand.control_ventilation_systems.**has_mechanical_ventilation_economizer**(*bpr*)

cea.demand.control_ventilation_systems.**has_mechanical_ventilation_heat_recovery**(*bpr*)

cea.demand.control_ventilation_systems.**has_night_flushing**(*bpr*)

cea.demand.control_ventilation_systems.**has_window_ventilation**(*bpr*)

cea.demand.control_ventilation_systems.**is_economizer_active**(*bpr*, *tsd*, *t*)
Control of activity of economizer of mechanical ventilation system Economizer of mechanical ventilation is controlled via zone set point temperatures, indoor air temperature and outdoor air temperature. Economizer is active during cooling season if the indoor air temperature exceeds the set point and the outdoor temperatures are lower than the set point. Economizer increases mechanical ventilation flow rate to the maximum.

Author: Gabriel Happle Date: APR 2017

Parameters

- **bpr** (*BuildingPropertiesRow*) – Building Properties
- **tsd** (*dict*) – Time series data of building
- **t** (*int*) – time step / hour of the year

Returns Economizer ON/OFF status

Return type bool

cea.demand.control_ventilation_systems.**is_mechanical_ventilation_active**(*bpr*, *tsd*, *t*)

cea.demand.control_ventilation_systems.**is_mechanical_ventilation_heat_recovery_active**(*bpr*, *tsd*, *t*)
Control of activity of heat exchanger of mechanical ventilation system

Author: Gabriel Happle Date: APR 2017

Parameters

- **bpr** (*BuildingPropertiesRow*) – Building Properties
- **tsd** (*dict*) – Time series data of building
- **t** (*int*) – time step / hour of the year

Returns Heat exchanger ON/OFF status

Return type bool

cea.demand.control_ventilation_systems.**is_night_flushing_active**(*bpr*, *tsd*, *t*)

cea.demand.control_ventilation_systems.**is_window_ventilation_active**(*bpr*, *tsd*, *t*)

cea.demand.datacenter_loads module

datacenter loads

cea.demand.datacenter_loads.**calc_Qcdataf**(*Edataf*)

cea.demand.demand_main module

cea.demand.demand_writers module

A collection of classes that write out the demand results files. The *cea.globalvar.GlobalVariables.demand_writer* variable references the *DemandWriter* to use. The default is *HourlyDemandWriter*. A *MonthlyDemandWriter* is provided that sums the values up monthly. See the *cea.analysis.sensitivity.sensitivity_demand* module for an example of using the *MonthlyDemandWriter*.

class cea.demand.demand_writers.**DemandWriter**(*gv*)
Bases: object

This is meant to be an abstract base class: Use the subclasses of this class instead.

Subclasses are expected to:

- set the *gv* field to a *cea.globalvar.GlobalVariables* instance in the constructor
- set the *vars_to_print* field in the constructor (FIXME: describe the *vars_to_print* structure.
- implement the *write_to_csv* method

results_to_csv(*tsd*, *bpr*, *locator*, *date*, *building_name*)

class cea.demand.demand_writers.**HourlyDemandWriter**(*gv*)
Bases: *cea.demand.demand_writers.DemandWriter*

Write out the hourly demand results

write_to_csv(*building_name*, *columns*, *hourly_data*, *locator*)

write_totals_csv(*building_properties*, *locator*)
read in the temporary results files and append them to the Totals.csv file.

class cea.demand.demand_writers.**MonthlyDemandWriter**(*gv*)
Bases: *cea.demand.demand_writers.DemandWriter*

Write out the monthly demand results

write_to_csv(*building_name*, *columns*, *hourly_data*, *locator*)

write_totals_csv(*building_properties*, *locator*)
read in the temporary results files and append them to the Totals.csv file.

cea.demand.electrical_loads module

Electrical loads

cea.demand.electrical_loads.**average_appliances_lighting_schedule**(*list_uses*, *schedules*, *building_uses*)
Calculate the schedule to use for lighting and appliances based on the building uses from the schedules defined for the project unsing a weighted average.

Parameters

- **list_uses** (*List[str]*) – the schedule names for the *schedules* parameter
- **schedules** (*List[Tuple[List[float], List[float], List[float]]]*) – the schedules, one for each name in *list_uses*. Each schedule is a list of 8760 floats...

- **building_uses** (*Dict[str, float]*) – A set of weights for the schedules as they apply to this particular building.

Returns appliances lighting schedule as a weighted average of the schedules for a specific building.

Return type numpy.ndarray[numpy.float64]

cea.demand.electrical_loads.**calc_Eauxf**(*Ll*, *Lw*, *Mww*, *Qcsf*, *Qcsf_0*, *Qhsf*, *Qhsf_0*, *Qww*, *Qwwf*, *Qwwf_0*, *Tcs_re*, *Tcs_sup*, *Ths_re*, *Ths_sup*, *Vw*, *Year*, *fforma*, *gv*, *nf_ag*, *nfp*, *sys_e_cooling*, *sys_e_heating*, *Ehs_lat_aux*, *tsd*)

cea.demand.electrical_loads.**calc_Eauxf_cs_dis**(*Qcsf*, *Qcsf0*, *Imax*, *deltaP_des*, *b*, *ts*, *tr*, *cpw*)

cea.demand.electrical_loads.**calc_Eauxf_fw**(*freshw*, *nf*, *gv*)

cea.demand.electrical_loads.**calc_Eauxf_hs_dis**(*Qhsf*, *Qhsf0*, *Imax*, *deltaP_des*, *b*, *ts*, *tr*, *cpw*)

cea.demand.electrical_loads.**calc_Eauxf_ve**(*tsd*, *gv*)
calculation of auxiliary electricity consumption of mechanical ventilation and AC fans

Parameters

- **tsd** (*dict*) – Time series data of building
- **gv** – global variables

:type gv :return: electrical energy for fans of mechanical ventilation in [Wh/h] :rtype: float

cea.demand.electrical_loads.**calc_Eauxf_ww**(*Qww*, *Qwwf*, *Qwwf0*, *Imax*, *deltaP_des*, *b*, *qV_des*)

cea.demand.electrical_loads.**calc_Edataf**(*schedule*, *Ed_Wm2*, *Aef*)
Calculates the final electricity consumption in data centers.

Parameters

- **schedule** (*ndarray*) – The data center schedule as calculated by *calc_Ea_El_Edata_Eref_schedule* but just for the SERVERROOM portion of the occupancy.
- **Ed_Wm2** (*float64*) – The maximum electrical consumption due to servers per unit of gross floor area (as taken from the building properties / internal loads file)
- **Aef** (*float64*) – The floor area with electricity in [m2]

Returns final electricity consumption in data centers per hour in [W]

Return type numpy.ndarray[numpy.float64]

cea.demand.electrical_loads.**calc_Eint**(*tsd*, *bpr*, *list_uses*, *schedules*)
Calculate final internal electrical loads (without auxiliary loads)

Parameters

- **tsd** (*Dict[str, numpy.ndarray]*) – Timestep data
- **bpr** (*cea.demand.thermal_loads.BuildingPropertiesRow*) – building properties
- **list_uses** (*list*) – The list of uses used in the project
- **schedules** (*List[numpy.ndarray]*) – The list of schedules defined for the project - in the same order as *list_uses*

- **building_uses** (*Dict[str, numpy.ndarray]*) – for each use in *list_uses*, the percentage of that use for this building. Sum of values is 1.0

Returns *tsd* with new keys: *['Eaf', 'Elf', 'Ealf', 'Edataf', 'Eref', 'Eprof']*

Return type Dict[str, numpy.ndarray]

cea.demand.electrical_loads.**calc_Eprof**(*schedule*, *Epro_Wm2*, *Aef*)

cea.demand.electrical_loads.**calc_Eprof_schedule**(*list_uses*, *schedules*, *building_uses*)
Calculate a weighted average of the schedules as defined in *list_uses* and *schedules*, using the weights in *building_uses* (which is taken from *bpr.occupancy*).

Parameters

- **list_uses** (*list of str*) – the schedule names for the *schedules* parameter
- **schedules** (*list of list of float*) – the schedules, one for each name in *list_uses*. Each schedule is a list of 8760 floats...
- **building_uses** (*dict of (str, float)*) – A set of weights for the schedules as they apply to this particular building.

Returns A weighted average of the schedules for a specific building.

Return type numpy.ndarray[numpy.float64]

cea.demand.electrical_loads.**calc_Eref**(*schedule*, *Ere_Wm2*, *Aef*)

cea.demand.hotwater_loads module

Hotwater load (it also calculates fresh water needs)

cea.demand.hotwater_loads.**calc_Qww**(*mww*, *Tww_sup_0*, *Tww_re*, *Cpw*)

cea.demand.hotwater_loads.**calc_Qww_dis_ls_nr**(*tair*, *Qww*, *Lvww_dis*, *Lvww_c*, *Y*, *Qww_0*, *V*, *Flowtap*, *twws*, *Cpw*, *Pwater*, *Bf*, *te*, *gv*)

cea.demand.hotwater_loads.**calc_Qww_dis_ls_r**(*Tair*, *Qww*, *lsww_dis*, *lcww_dis*, *Y*, *Qww_0*, *V*, *Flowtap*, *twws*, *Cpw*, *Pwater*, *gv*)

cea.demand.hotwater_loads.**calc_Qww_schedule**(*list_uses*, *schedules*, *building_uses*)
Algoithm to calculate the schedule of Qww use

Parameters

- **list_uses** –
- **schedules** –
- **building_uses** –

cea.demand.hotwater_loads.**calc_Qww_st_ls**(*T_ext*, *Ta*, *Qww*, *Vww*, *Qww_dis_ls_r*, *Qww_dis_ls_nr*, *gv*)

cea.demand.hotwater_loads.**calc_Qwwf**(*Af*, *Lcww_dis*, *Lsww_dis*, *Lvww_c*, *Lvww_dis*, *T_ext*, *Ta*, *Tww_re*, *Tww_sup_0*, *Y*, *gv*, *Vww_lpd*, *Vw_lpd*, *Occ_m2p*, *list_uses*, *schedules*, *building_uses*)
This function calculates the distribution heat loss and final energy consumption of domestic hot water. Final energy consumption of dhw includes dhw demand, sensible heat loss in hot water storage tank, and heat loss in the distribution network. :param Af: Conditioned floor area in m2. :param Lcww_dis: Length of dhw usage circulation pipeline in m. :param Lsww_dis: Length of dhw usage distribution pipeline in m. :param Lvww_c: Length of dhw heating circulation pipeline in m. :param Lvww_dis: Length of dhw heating distribution pipeline in m. :param T_ext: Ambient temperature in C. :param Ta: Room temperature in C. :param Tww_re: Domestic

hot water tank return temperature in C, this temperature is the ground water temperature, set according to norm. :param Tww_sup_0: Domestic hot water suppply set point temperature. :param vw: specific fresh water consumption in m3/hr*m2. :param vww: specific domestic hot water consumption in m3/hr*m2. :return:

cea.demand.hotwater_loads.**calc_disls**(*tamb*, *hotw*, *Flowtap*, *V*, *twws*, *Lsww_dis*, *p*, *cpw*, *Y*, *gv*)

cea.demand.hotwater_loads.**calc_mw**(*schedule*, *Vw_lpd*, *Occ_m2p*, *Af*, *Pwater*)

cea.demand.hotwater_loads.**calc_mww**(*schedule*, *Vww_lpd*, *Occ_m2p*, *Af*, *Pwater*)
Algorithm to calculate the hourly mass flow rate of domestic hot water

Parameters

- **schedule** – hourly DHW demand profile [1/h]
- **Vww_lpd** – DHW demand per person per day in [L/person/day]
- **Occ_m2p** – Occupant density in [m2/person]
- **Af** – Total floor area per building [m2]
- **Pwater** – water density [kg/m3]

cea.demand.occupancy_model module

Query schedules according to database

cea.demand.occupancy_model.**calc_occ**(*list_uses*, *schedules*, *bpr*)
Calculate the occupancy in number of people for the whole building per timestep.

Parameters

- **list_uses** (*list*) – The list of uses used in the project
- **schedules** (*list[ndarray[float]]*) – The list of schedules defined for the project - in the same order as *list_uses*
- **bpr** (*cea.demand.thermal_loads.BuildingPropertiesRow*) – The properties of the building to calculate

Returns Occupancy as number of persons per timestep for the whole building

Return type ndarray

cea.demand.occupancy_model.**calc_occ_schedule**(*list_uses*, *schedules*, *building_uses*)
Given schedule data for archetypical building uses, *calc_occ_schedule* calculates the schedule for a building with possibly a mixed schedule as defined in *building_uses* using a weighted average approach.

Parameters

- **list_uses** (*list*) – The list of uses used in the project
- **schedules** (*list[ndarray[float]]*) – The list of schedules defined for the project - in the same order as *list_uses*
- **building_uses** (*dict[str, float]*) – for each use in *list_uses*, the percentage of that use for this building. Sum of values is 1.0

Returns

Return type ndarray

cea.demand.occupancy_model.**read_schedules**(*use*, *x*)

cea.demand.occupancy_model.**schedule_maker**(*dates*, *locator*, *list_uses*)

cea.demand.rc_model_SIA module

cea.demand.rc_model_SIA.**calc_f_ic**(*a_t*, *a_m*, *h_ec*)

Parameters

- **a_t** – see bpr.rc_model['Atot']
- **a_m** – see bpr.rc_model['Am']
- **h_ec** – see calc_h_ec

Returns

cea.demand.rc_model_SIA.**calc_f_im**(*a_t*, *a_m*)

Parameters

- **a_t** – see bpr.rc_model['Atot']
- **a_m** – see bpr.rc_model['Am']

Returns

cea.demand.rc_model_SIA.**calc_f_sc**(*a_t*, *a_m*, *a_w*, *h_ec*)

Parameters

- **a_t** – see bpr.rc_model['Atot']
- **a_m** – see bpr.rc_model['Am']
- **a_w** – see bpr.rc_model['Aw']
- **h_ec** – see calc_h_ec

Returns

cea.demand.rc_model_SIA.**calc_f_sm**(*a_t*, *a_m*, *a_w*)

Parameters

- **a_t** – bpr.rc_model['Atot']
- **a_m** – bpr.rc_model['Am']
- **a_w** – bpr.rc_model['Aw']

Returns

cea.demand.rc_model_SIA.**calc_h_1**(*h_ea*, *h_ac*)

cea.demand.rc_model_SIA.**calc_h_2**(*h_1*, *h_ec*)

cea.demand.rc_model_SIA.**calc_h_3**(*h_2*, *h_mc*)

cea.demand.rc_model_SIA.**calc_h_ac**(*a_t*)

Parameters **a_t** – equivalent to bpr.rc_model['Atot']

Returns

cea.demand.rc_model_SIA.**calc_h_ea**(*m_ve_mech*, *m_ve_window*, *m_ve_inf_simple*)

cea.demand.rc_model_SIA.**calc_h_ec**(*Htr_w*)

cea.demand.rc_model_SIA.**calc_h_em**(*h_op_m*, *h_mc*)

cea.demand.rc_model_SIA.**calc_h_j_em**()

cea.demand.rc_model_SIA.**calc_h_mc**(*a_m*)

Parameters **a_m** – see `bpr.rc_model['Am']`

Returns

cea.demand.rc_model_SIA.**calc_h_op_m**(*Htr_op*)

cea.demand.rc_model_SIA.**calc_h_tabs**()

cea.demand.rc_model_SIA.**calc_phi_a**(*phi_hc_cv*, *phi_i_l*, *phi_i_a*, *phi_i_p*, *I_sol*)

cea.demand.rc_model_SIA.**calc_phi_c**(*phi_hc_r*, *phi_i_l*, *phi_i_a*, *phi_i_p*, *I_sol*, *f_ic*, *f_sc*)

cea.demand.rc_model_SIA.**calc_phi_hc_cv**(*phi_hc*, *f_hc_cv*)

cea.demand.rc_model_SIA.**calc_phi_hc_r**(*phi_hc*, *f_hc_cv*)

cea.demand.rc_model_SIA.**calc_phi_i_a**(*Eaf*, *Qcdataf*, *Qcref*)

cea.demand.rc_model_SIA.**calc_phi_i_l**(*Elf*)

cea.demand.rc_model_SIA.**calc_phi_i_p**(*Qs_Wp*, *people*)

cea.demand.rc_model_SIA.**calc_phi_m**(*phi_hc_r*, *phi_i_l*, *phi_i_a*, *phi_i_p*, *I_sol*, *f_im*, *f_sm*)

cea.demand.rc_model_SIA.**calc_phi_m_tot**(*phi_m*, *phi_a*, *phi_c*, *theta_ea*, *theta_em*, *theta_ec*, *h_1*, *h_2*, *h_3*, *h_ec*, *h_ea*, *h_em*)

cea.demand.rc_model_SIA.**calc_phi_m_tot_tabs**()

cea.demand.rc_model_SIA.**calc_phi_tabs**()

cea.demand.rc_model_SIA.**calc_rc_model_temperatures**(*phi_hc_cv*, *phi_hc_r*, *bpr*, *tsd*, *t*)

cea.demand.rc_model_SIA.**calc_rc_model_temperatures_cooling**(*phi_hc*, *bpr*, *tsd*, *t*)
This function executes the equations of SIA 2044 R-C-Building-Model to calculate the node temperatures for a given cooling energy demand

:py:func: *cea.demand.rc_model_SIA.lookup_f_hc_cv_cooling* :py:func: *cea.demand.rc_model_SIA.calc_phi_hc_cv* :py:func: *cea.demand.rc_model_SIA.calc_phi_hc_r* :py:func: *cea.demand.rc_model_SIA.calc_rc_model_temperatures*

Author: Gabriel Happle Date: FEB 2017

Parameters

- **phi_hc** (*float*) – Heating or cooling energy demand of building
- **bpr** (*BuildingPropertiesRow*) – Building Properties
- **tsd** (*dict*) – Time series data of building
- **t** (*int*) – time step / hour of the year

Returns R-C-Building-Model node temperatures

Return type dict

cea.demand.rc_model_SIA.**calc_rc_model_temperatures_heating**(*phi_hc*, *bpr*, *tsd*, *t*)
This function executes the equations of SIA 2044 R-C-Building-Model to calculate the node temperatures for a given heating energy demand

:py:func: *cea.demand.rc_model_SIA.lookup_f_hc_cv_heating* :py:func: *cea.demand.rc_model_SIA.calc_phi_hc_cv* :py:func: *cea.demand.rc_model_SIA.calc_phi_hc_r* :py:func: *cea.demand.rc_model_SIA.calc_rc_model_temperatures*

Author: Gabriel Happle Date: FEB 2017

Parameters

- **phi_hc** (*float*) – Heating or cooling energy demand of building
- **bpr** (*BuildingPropertiesRow*) – Building Properties

- **tsd** (*dict*) – Time series data of building
- **t** (*int*) – time step / hour of the year

Returns R-C-Building-Model node temperatures

Return type dict

cea.demand.rc_model_SIA.**calc_rc_model_temperatures_no_heating_cooling**(*bpr*, *tsd*, *t*)

Calculates R-C-Model temperatures are calculated with zero heating/cooling power according to SIA 2044 procedure.

:py:func: *cea.demand.rc_model_SIA.calc_rc_model_temperatures_no_heating_cooling*

Author: Gabriel Happle Date: FEB 2017

Parameters

- **bpr** (*BuildingPropertiesRow*) – Building Properties
- **tsd** (*dict*) – Time series data of building
- **t** (*int*) – time step / hour of the year

Returns R-C-Model node temperatures

Return type dict

cea.demand.rc_model_SIA.**calc_theta_a**(*phi_a*, *theta_ea*, *theta_c*, *h_ac*, *h_ea*)

cea.demand.rc_model_SIA.**calc_theta_c**(*phi_a*, *phi_c*, *theta_ea*, *theta_ec*, *theta_m*, *h_1*, *h_mc*, *h_ec*, *h_ea*)

cea.demand.rc_model_SIA.**calc_theta_e_star**()

cea.demand.rc_model_SIA.**calc_theta_ea**(*m_ve_mech*, *m_ve_window*, *m_ve_inf_simple*, *theta_ve_mech*, *T_ext*)

cea.demand.rc_model_SIA.**calc_theta_ec**(*T_ext*)

cea.demand.rc_model_SIA.**calc_theta_em**(*T_ext*)

cea.demand.rc_model_SIA.**calc_theta_m**(*theta_m_t*, *theta_m_t_1*)

cea.demand.rc_model_SIA.**calc_theta_m_t**(*phi_m_tot*, *theta_m_t_1*, *h_em*, *h_3*, *c_m*)

cea.demand.rc_model_SIA.**calc_theta_o**(*theta_a*, *theta_c*)

cea.demand.rc_model_SIA.**calc_theta_tabs_su**()

cea.demand.rc_model_SIA.**has_cooling_demand**(*bpr*, *tsd*, *t*)

This function checks whether the building R-C-Model has a cooling demand according to the procedure in SIA 2044. R-C-Model temperatures are calculated with zero cooling power and checked versus the set-point temperature. Function includes a temperature tolerance according to the precision of the result reporting.

:py:func: *cea.demand.rc_model_SIA.calc_rc_model_temperatures_no_heating_cooling*

Author: Gabriel Happle Date: FEB 2017

Parameters

- **bpr** (*BuildingPropertiesRow*) – Building Properties
- **tsd** (*dict*) – Time series data of building
- **t** (*int*) – time step / hour of the year

Returns True or False

Return type bool

cea.demand.rc_model_SIA.**has_heating_demand**(*bpr*, *tsd*, *t*)
This function checks whether the building R-C-Model has a heating demand according to the procedure in SIA 2044. R-C-Model temperatures are calculated with zero heating power and checked versus the set-point temperature. Function includes a temperature tolerance according to the precision of the result reporting.

:py:func: *cea.demand.rc_model_SIA.calc_rc_model_temperatures_no_heating_cooling*

Author: Gabriel Happle Date: FEB 2017

Parameters

- **bpr** (*BuildingPropertiesRow*) – Building Properties
- **tsd** (*dict*) – Time series data of building
- **t** (*int*) – time step / hour of the year

Returns True or False

Return type bool

cea.demand.rc_model_SIA.**lookup_f_hc_cv_cooling**(*bpr*)

cea.demand.rc_model_SIA.**lookup_f_hc_cv_heating**(*bpr*)

cea.demand.rc_model_crank_nicholson_procedure module

cea.demand.rc_model_crank_nicholson_procedure.**calc_rc_model_demand_heating_cooling**(*bpr*, *tsd*, *t*, *gv*)
Crank-Nicholson Procedure to calculate heating / cooling demand of buildings following the procedure in 2.3.2 in SIA 2044 / Korrigenda C1 zum Merkblatt SIA 2044:2011 / Korrigenda C2 zum Mekblatt SIA 2044:2011

Special procedures for updating ventilation air AC-heated and AC-cooled buildings

Author: Gabriel Happle Date: 01/2017

Parameters

- **bpr** – building properties row object
- **tsd** – time series data dict
- **t** – time step / hour of year [0..8760]
- **gv** – globalvars

Returns updates values in tsd

cea.demand.rc_model_crank_nicholson_procedure.**update_tsd_no_cooling**(*tsd*, *t*)
updates NaN values in tsd for case of no cooling demand

Author: Gabriel Happle Date: 01/2017

Parameters

- **tsd** – time series data dict
- **t** – time step / hour of year [0..8760]

Returns updates tsd values

cea.demand.rc_model_crank_nicholson_procedure.**update_tsd_no_heating**(*tsd*, *t*)
updates NaN values in tsd for case of no heating demand

Author: Gabriel Happle Date: 01/2017

Parameters

- **tsd** – time series data dict
- **t** – time step / hour of year [0..8760]

Returns updates tsd values

cea.demand.refrigeration_loads module

refrigeration loads

cea.demand.refrigeration_loads.**calc_Qcref**(*Eref*)

cea.demand.sensible_loads module

Sensible space heating and space cooling loads EN-13970

cea.demand.sensible_loads.**calc_Asol**(*t*, *bpr*, *gv*)
This function calculates the effective collecting solar area accounting for use of blinds according to ISO 13790, for the sake of simplicity and to avoid iterations, the delta is calculated based on the last time step.

Parameters

- **t** – time of the year
- **bpr** – building properties object
- **gv** – global variables class

Returns

cea.demand.sensible_loads.**calc_I_rad**(*t*, *tsd*, *bpr*, *Rse*)
This function calculates the solar radiation re-irradiated from a building to the sky according to ISO 13790

Parameters

- **t** – hour of the year
- **tsd** – time series dataframe
- **bpr** – building properties object
- **gv** – global variables class

Returns I_rad: vector solar radiation re-irradiated to the sky.

cea.demand.sensible_loads.**calc_I_sol**(*t*, *bpr*, *tsd*, *gv*)
This function calculates the net solar radiation (incident -reflected - re-irradiated) according to ISO 13790

Parameters

- **t** – hour of the year
- **bpr** – building properties object
- **tsd** – time series dataframe
- **gv** – global variables class

Returns I_sol: vector of net solar radiation to the building I_rad: vector solar radiation re-irradiated to the sky.

cea.demand.sensible_loads.**calc_Qgain_lat**(*people*, *X_ghp*, *sys_e_cooling*, *sys_e_heating*)

cea.demand.sensible_loads.**calc_Qgain_sen**(*t*, *tsd*, *bpr*, *gv*)

cea.demand.sensible_loads.**calc_Qhs_Qcs_dis_ls**(*tair*, *text*, *Qhs*, *Qcs*, *tsh*, *trh*, *tsc*, *trc*, *Qhs_max*, *Qcs_max*, *D*, *Y*, *SystemH*, *SystemC*, *Bf*, *Lv*)

calculates distribution losses based on ISO 15316

cea.demand.sensible_loads.**calc_Qhs_Qcs_sys_max**(*Af*, *prop_HVAC*)

cea.demand.sensible_loads.**calc_hr**(*emissivity*, *theta_ss*)

This function calculates the external radiative heat transfer coefficient according to ISO 13790

Parameters

- **emissivity** – emissivity of the considered surface
- **theta_ss** – delta of temperature between building surface and the sky.

Returns hr:

cea.demand.sensible_loads.**calc_temperatures_emission_systems**(*tsd*, *bpr*, *Qcsf_0*, *Qhsf_0*, *gv*)

cea.demand.space_emission_systems module

Space emission systems (heating and cooling) EN 15316-2 prEN 15316-2:2014

cea.demand.space_emission_systems.**calc_delta_theta_int_inc_cooling**(*cooling_system*, *control_system*)

Model of losses in the emission and control system for space heating and cooling.

Correction factor for the heating and cooling setpoints. Extracted from EN 15316-2

(see ceadatabasesCHSystemsemission_systems.xls for valid values for the heating and cooling system values)

T0 means there's no heating/cooling systems installed, therefore, also no control systems for heating/cooling. In short, when the input system is T0, the output set point correction should be 0.0. So if there is no cooling systems, the setpoint_correction_for_space_emission_systems function input: (T1, T0, T1) (type_hs, type_cs, type_ctrl), return should be (2.65, 0.0), the control system is only specified for the heating system. In another case with no heating systems: input: (T0, T3, T1) return: (0.0, -2.0), the control system is only specified for the heating system.

Parameters

- **heating_system** (*str*) – The heating system used. Valid values: T0, T1, T2, T3, T4
- **cooling_system** (*str*) – The cooling system used. Valid values: T0, T1, T2, T3
- **control_system** (*str*) – The control system used. Valid values: T1, T2, T3, T4 - as defined in the contributors manual under Databases / Archetypes / Building Properties / Mechanical systems. T1 for none, T2 for PI control, T3 for PI control with optimum tuning, and T4 for room temperature control (electromagnetically/electronically).

Returns two delta T to correct the set point temperature, dT_heating, dT_cooling

Return type tuple(double, double)

cea.demand.space_emission_systems.**calc_delta_theta_int_inc_heating**(*heating_system*, *control_system*)

Model of losses in the emission and control system for space heating and cooling.

Correction factor for the heating and cooling setpoints. Extracted from EN 15316-2

(see ceadatabasesCHSystemsemission_systems.xls for valid values for the heating and cooling system values)

T0 means there's no heating/cooling systems installed, therefore, also no control systems for heating/cooling. In short, when the input system is T0, the output set point correction should be 0.0. So if there is no cooling systems, the setpoint_correction_for_space_emission_systems function input: (T1, T0, T1) (type_hs, type_cs, type_ctrl), return should be (2.65, 0.0), the control system is only specified for the heating system. In another case with no heating systems: input: (T0, T3, T1) return: (0.0, -2.0), the control system is only specified for the heating system.

Parameters

- **heating_system** (*str*) – The heating system used. Valid values: T0, T1, T2, T3, T4
- **cooling_system** (*str*) – The cooling system used. Valid values: T0, T1, T2, T3
- **control_system** (*str*) – The control system used. Valid values: T1, T2, T3, T4 - as defined in the contributors manual under Databases / Archetypes / Building Properties / Mechanical systems. T1 for none, T2 for PI control, T3 for PI control with optimum tuning, and T4 for room temperature control (electromagnetically/electronically).

Returns two delta T to correct the set point temperature, dT_heating, dT_cooling

Return type tuple(double, double)

cea.demand.space_emission_systems.**calc_q_em_ls**(*q_em_out*, *delta_theta_int_inc*, *theta_int_inc*, *theta_e_comb*, *q_em_max*)

Eq. (8) in [prEN 15316-2:2014]

With modification of capping emission losses at system capacity [Happle 01/2017]

Parameters

- **q_em_out** – heating power of emission system (W)
- **delta_theta_int_inc** – delta temperature caused by all losses (K)
- **theta_int_inc** – equivalent room temperature (°C)
- **theta_e_comb** – ?comb? outdoor temperature (°C)

Returns

cea.demand.space_emission_systems.**calc_q_em_ls_cooling**(*bpr*, *tsd*, *hoy*)
calculation procedure for space emissions losses in the cooling case [prEN 15316-2:2014]

Returns

cea.demand.space_emission_systems.**calc_q_em_ls_heating**(*bpr*, *tsd*, *hoy*)
calculation procedure for space emissions losses in the heating case [prEN 15316-2:2014]

Returns

cea.demand.space_emission_systems.**calc_theta_e_comb_cooling**(*theta_e*, *bpr*)
Eq. (10) in [prEN 15316-2:2014]

Returns

cea.demand.space_emission_systems.**calc_theta_e_comb_heating**(*theta_e*)
Eq. (9) in [prEN 15316-2:2014]

Returns

cea.demand.space_emission_systems.**calc_theta_int_inc**(*theta_int_ini*, *delta_theta_int_inc*)
Eq. (1) in [prEN 15316-2:2014]

Parameters **theta_int_ini** –

Returns

cea.demand.space_emission_systems.**get_delta_theta_e_sol**(*bpr*)
Appendix B.7 in [prEN 15316-2:2014]

delta_theta_e_sol = 8K – for medium window fraction or internal loads (e.g. residential) delta_theta_e_sol = 12K – for large window fraction or internal loads (e.g. office)

Parameters **bpr** –

Returns

cea.demand.thermal_loads module

cea.demand.ventilation_air_flows_detailed module

Ventilation according to *[DIN-16798-7]* and *[ISO-9972]*

Convention: all temperature inputs in (°C)

cea.demand.ventilation_air_flows_detailed.**allocate_default_leakage_paths**(*coeff_lea_zone*, *area_facade_zone*, *area_roof_zone*, *height_zone*)
Allocate default leakage paths according to B.1.3.17 in [1]

Parameters

- **coeff_lea_zone** – leakage coefficient of zone
- **area_facade_zone** – facade area of zone (m2)
- **area_roof_zone** – roof area of zone (m2)
- **height_zone** – height of zone (m)

Returns

- coeff_lea_path : coefficients of default leakage paths
- height_lea_path : heights of default leakage paths (m)
- orientation_lea_path : orientation index of default leakage paths (-)

cea.demand.ventilation_air_flows_detailed.**allocate_default_ventilation_openings**(*coeff_vent_zone*, *height_zone*)
Allocate default ventilation openings according to B.1.3.13 in [1]

:param coeff_vent_zone : coefficient of ventilation openings of zone :param height_zone : height of zone (m)

Returns

- coeff_vent_path : coefficients of default ventilation opening paths
- height_vent_path : heights of default ventilation opening paths (m)

- orientation_vent_path : orientation index of default ventilation opening paths (-)

cea.demand.ventilation_air_flows_detailed.**calc_air_flow_mass_balance**(*p_zone_ref*, *temp_zone*, *u_wind_10*, *temp_ext*, *dict_props_nat_vent*, *option*)

Air flow mass balance for iterative calculation according to 6.4.3.9 in [1]

:param p_zone_ref : zone reference pressure (Pa) :param temp_zone : air temperature in ventilation zone (°C) :param u_wind_10 : meteorological wind velocity (m/s) :param temp_ext : exterior air temperature (°C) :param dict_props_nat_vent : dictionary containing natural ventilation properties of zone :param option : 'minimize' = returns sum of air mass flows, 'calculate' = returns air mass flows

Returns sum of air mass flows in and out of zone in (kg/h)

cea.demand.ventilation_air_flows_detailed.**calc_air_flows**(*temp_zone*, *u_wind*, *temp_ext*, *dict_props_nat_vent*)

Minimization of variable air flows as a function of zone gauge

Parameters

- **temp_zone** – zone indoor air temperature (°C)
- **u_wind** – wind velocity (m/s)
- **temp_ext** – exterior air temperature (°C)
- **dict_props_nat_vent** – dictionary containing natural ventilation properties of zone

qm_sum_in : total air mass flow rates into zone (kg/h) qm_sum_out : total air mass flow rates out of zone (kg/h)

cea.demand.ventilation_air_flows_detailed.**calc_area_window_cros**(*dict_windows_building*, *r_window_arg*)

Calculate cross-ventilation window area according to the procedure in 6.4.3.5.4.3 in [1]

:param dict_windows_building : dictionary containing information of all windows in building :param r_window_arg : fraction of window opening (-)

Returns area_window_cros : effective window area for cross ventilation (m2)

cea.demand.ventilation_air_flows_detailed.**calc_area_window_free**(*area_window_max*, *r_window_arg*)

Calculate free window opening area according to 6.4.3.5.2 in [1]

:param area_window_max : area of single operable window (m2) :param r_window_arg : fraction of window opening (-)

Returns area_window_free : open area of window (m2)

cea.demand.ventilation_air_flows_detailed.**calc_area_window_tot**(*dict_windows_building*, *r_window_arg*)

Calculation of total open window area according to 6.4.3.5.2 in [1]

:param dict_windows_building : dictionary containing information of all windows in building :param r_window_arg : fraction of window opening (-)

Returns area_window_tot = total open area of windows in building (m2)

cea.demand.ventilation_air_flows_detailed.**calc_coeff_lea_zone**(*qv_delta_p_lea_ref*)

Calculate default leakage coefficient of zone according to B.1.3.16 in [1]

Parameters **qv_delta_p_lea_ref** – air volume flow rate at reference pressure (m3/h)

Returns coeff_lea_zone : leakage coefficient of zone

cea.demand.ventilation_air_flows_detailed.**calc_coeff_vent_zone**(*area_vent_zone*)
Calculate air volume flow coefficient of ventilation openings of zone according to 6.4.3.6.4 in [1]

:param area_vent_zone : total area of ventilation openings of zone (cm2)

:returns coeff_vent_zone : coefficient of ventilation openings of zone

cea.demand.ventilation_air_flows_detailed.**calc_delta_p_path**(*p_zone_ref*, *height_path*, *temp_zone*, *coeff_wind_pressure_path*, *u_wind_site*, *temp_ext*)
Calculation of indoor-outdoor pressure difference at air path according to 6.4.2.4 in [1]

Parameters

- **p_zone_ref** – zone reference pressure (Pa)
- **height_path** – height of ventilation path (m)
- **temp_zone** – air temperature of ventilation zone in (°C)
- **coeff_wind_pressure_path** – wind pressure coefficient of ventilation path (-)
- **u_wind_site** – wind velocity (m/s)
- **temp_ext** – external air temperature (°C)

Returns delta_p_path, pressure difference across ventilation path (Pa)

cea.demand.ventilation_air_flows_detailed.**calc_effective_stack_height**(*dict_windows_building*)
Calculation of effective stack height for window ventilation according to 6.4.3.4.1 in [1]

:param dict_windows_building : dictionary containing information of all windows in building

Returns height_window_stack : effective stack height of windows of building (m)

cea.demand.ventilation_air_flows_detailed.**calc_qm_arg**(*factor_cros*, *temp_ext*, *dict_windows_building*, *u_wind_10*, *temp_zone*, *r_window_arg*)
Calculation of cross ventilated and non-cross ventilated window ventilation according to procedure in 6.4.3.5.4 in [1]

:param factor_cros : cross ventilation factor [0,1] :param temp_ext : exterior temperature (°C) :param dict_windows_building : dictionary containing information of all windows in building :param u_wind_10 : wind velocity (m/s) :param temp_zone : zone temperature (°C) :param r_window_arg : fraction of window opening (-)

Returns window ventilation air mass flows in (kg/h)

cea.demand.ventilation_air_flows_detailed.**calc_qm_lea**(*p_zone_ref*, *temp_zone*, *temp_ext*, *u_wind_site*, *dict_props_nat_vent*)
Calculation of leakage infiltration and exfiltration air mass flow as a function of zone indoor reference pressure

Parameters

- **p_zone_ref** – zone reference pressure (Pa)
- **temp_zone** – air temperature in ventilation zone (°C)

- **temp_ext** – exterior air temperature (°C)
- **u_wind_site** – wind velocity (m/s)
- **dict_props_nat_vent** – dictionary containing natural ventilation properties of zone

Returns

- qm_lea_in : air mass flow rate into zone through leakages (kg/h)
- qm_lea_out : air mass flow rate out of zone through leakages (kg/h)

cea.demand.ventilation_air_flows_detailed.**calc_qm_vent**(*p_zone_ref*, *temp_zone*, *temp_ext*, *u_wind_site*, *dict_props_nat_vent*)

Calculation of air flows through ventilation openings in the facade

:param p_zone_ref : zone reference pressure (Pa) :param temp_zone : zone air temperature (°C) :param temp_ext : exterior air temperature (°C) :param u_wind_site : wind velocity (m/s) :param dict_props_nat_vent : dictionary containing natural ventilation properties of zone

Returns

- qm_vent_in : air mass flow rate into zone through ventilation openings (kg/h)
- qm_vent_out : air mass flow rate out of zone through ventilation openings (kg/h)

cea.demand.ventilation_air_flows_detailed.**calc_qv_delta_p_ref**(*n_delta_p_ref*, *vol_building*)

Calculate airflow at reference pressure according to 6.3.2 in [2]

Parameters

- **n_delta_p_ref** – air changes at reference pressure [1/h]
- **vol_building** – building_volume [m3]

Returns qv_delta_p_ref : air volume flow rate at reference pressure (m3/h)

cea.demand.ventilation_air_flows_detailed.**calc_qv_lea_path**(*coeff_lea_path*, *delta_p_lea_path*)

Calculate volume air flow of single leakage path according to 6.4.3.6.5 in [1]

Parameters

- **coeff_lea_path** – coefficient of leakage path
- **delta_p_lea_path** – pressure difference across leakage path (Pa)

Returns qv_lea_path : volume flow rate across leakage path (m3/h)

cea.demand.ventilation_air_flows_detailed.**calc_qv_vent_path**(*coeff_vent_path*, *delta_p_vent_path*)

Calculate volume air flow of single ventilation opening path according to 6.4.3.6.4 in [1]

:param coeff_vent_path : ventilation opening coefficient of air path :param delta_p_vent_path : pressure difference across air path (Pa)

Returns qv_vent_path : air volume flow rate across air path (m3/h)

cea.demand.ventilation_air_flows_detailed.**calc_u_wind_site**(*u_wind_10*)

Adjusts meteorological wind velocity to site surroundings according to 6.4.2.2 in [1]

Parameters **u_wind_10** – meteorological wind velocity (m/s)

Returns u_wind_site, site wind velocity (m/s)

cea.demand.ventilation_air_flows_detailed.**get_properties_natural_ventilation**(*bpr*, *gv*)

gdf_geometry_building : GeoDataFrame containing geometry properties of single building gdf_architecture_building : GeoDataFrame containing architecture props of single building :param gv: globalvars :param bpr: building propert row

Returns dictionary containing natural ventilation properties of zone

cea.demand.ventilation_air_flows_detailed.**lookup_coeff_wind_pressure**(*height_path*, *class_shielding*, *orientation_path*, *slope_roof*, *factor_cros*)

Lookup default wind pressure coefficients for air leakage paths according to B.1.3.3 in [1]

Parameters

- **height_path** –
- **class_shielding** –
- **orientation_path** –
- **slope_roof** –
- **factor_cros** –

Returns wind pressure coefficients (-)

Conventions:

class_shielding = 0 : open terrain class_shielding = 1 : normal class_shielding = 2 : shielded

orientation_path = 0 [facade facing wind] 1 : facade not facing wind 2 : roof

factor_cros = 0 [cross ventilation not possible] = 1 : cross ventilation possible

cea.demand.ventilation_air_flows_detailed.**testing**()

cea.demand.ventilation_air_flows_simple module

cea.demand.ventilation_air_flows_simple.**calc_air_mass_flow_mechanical_ventilation**(*bpr*, *tsd*, *t*)

Calculates mass flow rate of mechanical ventilation at time step t according to ventilation control options and building systems properties

Author: Gabriel Happle Date: 01/2017

Parameters

- **bpr** – Building properties row object
- **tsd** – Time series data dict
- **t** – time step [0..8760]

Returns updates tsd

cea.demand.ventilation_air_flows_simple.**calc_air_mass_flow_window_ventilation**(*bpr*, *tsd*, *t*)

Calculates mass flow rate of window ventilation at time step t according to ventilation control options and building systems properties

Author: Gabriel Happle Date: 01/2017

Parameters

- **bpr** – Building properties row object
- **tsd** – Time series data dict
- **t** – time step [0..8760]

Returns updates tsd

cea.demand.ventilation_air_flows_simple.**calc_m_ve_leakage**()

cea.demand.ventilation_air_flows_simple.**calc_m_ve_leakage_simple**(*bpr*, *tsd*, *gv*)

Calculates mass flow rate of leakage at time step t according to ventilation control options and building systems properties

Estimation of infiltration air volume flow rate according to Eq. (3) in DIN 1946-6

Author: Gabriel Happle Date: 01/2017

Parameters

- **bpr** – Building properties row object
- **tsd** – Time series data dict
- **gv** – globalvars

Returns updates tsd

cea.demand.ventilation_air_flows_simple.**calc_m_ve_required**(*bpr*, *tsd*)
Calculate required outdoor air ventilation rate according to occupancy

Author: Legacy Date: old

Parameters

- **bpr** – Building properties row object
- **tsd** – Time series data dict

Returns updates tsd

cea.demand.ventilation_air_flows_simple.**calc_theta_ve_mech**(*bpr*, *tsd*, *t*, *gv*)

Calculates supply temperature of mechanical ventilation system according to ventilation control options and building systems properties

Author: Gabriel Happle Date: 01/2017

Parameters

- **bpr** – Building properties row object
- **tsd** – Time series data dict
- **t** – time step [0..8760]
- **gv** – globalvars

Returns updates tsd

Module contents

cea.geometry package

Submodules

cea.geometry.geometry_reader module

algorithms for manipulation of building geometry

cea.geometry.geometry_reader.**create_windows**(*df_prop_surfaces*, *gdf_building_architecture*)

Creates windows on exposed building surfaces according to building win-wall-ratio

Parameters

- **df_prop_surfaces** (*DataFrame*) – DataFrame containing all exposed building surfaces (this is the *properties_surfaces.csv* file from the radiation calculation)
- **gdf_building_architecture** (*GeoDataFrame*) – GeoDataFrame containing building architecture - this is the *architecture.shp* file from the scenario input, containing the *win_wall* column with the window to wall ratio.

Returns DataFrame containing all windows of all buildings

Return type DataFrame

Sample rows of output::

```
            angle_window  area_window  height_window_above_ground          0            90     1.9108
    1             90     2.276739                          1.5
    2             90     2.276739                          4.5
    3             90     2.276739                          7.5
    4             90     2.276739                         10.5

       height_window_in_zone name_building  orientation_window
    0                    1.5       B140589                   0
    1                    1.5       B140590                 180
    2                    4.5       B140590                 180
    3                    7.5       B140590                 180
    4                   10.5       B140590                 180

    [5 rows x 6 columns]
```

cea.geometry.geometry_reader.**get_building_geometry_ventilation**(*gdf_building_geometry*)

:param gdf_building_geometry : GeoDataFrame contains single building

Returns building properties for natural ventilation calculation

Module contents

cea.plots package

Submodules

cea.plots.graphs_demand module

graphs algorithm

cea.plots.graphs_demand.**create_demand_graph_for_building**(*analysis_fields*, *area_df*, *color_palette*, *fields_date*, *locator*, *name*, *total_demand*)

cea.plots.graphs_demand.**demand_graph_fields**(*scenario_path*)
Lists the available fields for the demand graphs - these are fields that are present in both the building demand results files as well as the totals file (albeit with different units).

cea.plots.graphs_demand.**graphs_demand**(*locator*, *analysis_fields*, *gv*)
algorithm to print graphs in PDF concerning the dynamics of each and all buildings

Parameters

- **locator** (inputlocator.InputLocator) – an InputLocator set to the scenario to compute
- **analysis_fields** (*list[string]*) – list of fields (column names in Totals.csv) to analyse

Returns

- Graphs of each building and total: .Pdf
- heat map file per variable of interest n.

cea.plots.graphs_demand.**run_as_script**(*scenario_path=None*, *analysis_fields=['Ealf_kWh', 'Qhsf_kWh', 'Qwwf_kWh', 'Qcsf_kWh']*)

cea.plots.graphs_optimization module

Note: documentation pending

cea.plots.graphs_solar_potential module

Solar graphs

cea.plots.graphs_solar_potential.**calc_graph_I_sol**(*hourlydata_groups*)

cea.plots.graphs_solar_potential.**calc_graph_PV**(*results*, *results_perarea*)

cea.plots.graphs_solar_potential.**calc_graph_SC**(*result*, *Tin*)

cea.plots.heatmaps module

cea.plots.scenario_plots module

scenario_plots.py

Create a list of plots for comparing multiple scenarios.

cea.plots.scenario_plots.**create_page_demand**(*locators*, *pdf*, *scenario_names*)
Create Page one: Demand :param locators: list of InputLocators, one for each scenario :param pdf: the PdfFile to write the page to :param scenario_names: list of scenario names :return: None

cea.plots.scenario_plots.**create_page_lca_embodied**(*locators*, *pdf*, *scenario_names*)
Create Page Two: LCA Embodied :param locators: list of InputLocators, one for each scenario :param pdf: the PdfFile to write the page to :param scenario_names: list of scenario names :return: None

cea.plots.scenario_plots.**create_page_lca_operation**(*locators*, *pdf*, *scenario_names*)
Create Page Three: LCA Operation :param locators: list of InputLocators, one for each scenario :param pdf: the PdfFile to write the page to :param scenario_names: list of scenario names :return: None

cea.plots.scenario_plots.**plot_demand**(*ax*, *locators*, *scenario_names*, *column*, *title*)

cea.plots.scenario_plots.**plot_lca_embodied**(*ax*, *locators*, *scenario_names*, *column*, *title*, *unit*)

cea.plots.scenario_plots.**plot_lca_operation**(*ax*, *locators*, *scenario_names*, *column*, *title*, *unit*)

cea.plots.scenario_plots.**plot_scenarios**(*scenarios*, *output_file*)
List each scenario in the folder *scenario_root* and plot demand and lca (operations, embodied) data.

Parameters

- **scenarios** – A list of scenario folders.
- **output_file** – The filename (pdf) to save the results as.

Returns (None)

cea.plots.scenario_plots.**run_as_script**(*scenario_folders=None*, *output_file=None*)

cea.plots.sensitivity_demand_graphs module

Graphs for sensitivity_demand.py

cea.plots.sensitivity_demand_graphs.**graph**(*locator*, *parameters*, *method*, *samples*)

Parameters

- **locator** – locator class
- **parameters** – list of output parameters to analyse
- **method** – 'morris' or 'sobol' methods
- **samples** – number of samples to calculate

Returns .pdf file per output_parameter stored in locator.get_sensitivity_plots_file()

cea.plots.sensitivity_demand_graphs.**run_as_script**()

cea.plots.timeseries_interactive_graph module

Module contents

cea.resources package

Submodules

cea.resources.geothermal module

cea.resources.geothermal.**calc_ground_temperature**(*T_ambient*, *gv*)
Calculates hourly ground temperature fluctuation over a year following [Kusuda, T. et al., 1965]_.

Parameters

- **T_ambient** (*np array*) – vector with outdoor temperature
- **gv** – globalvar.py

Return Tg vector with ground temperatures in [K]

Rtype Tg np array

..[Kusuda, T. et al., 1965] Kusuda, T. and P.R. Achenbach (1965). Earth Temperatures and Thermal Diffusivity at Selected Stations in the United States. ASHRAE Transactions. 71(1):61-74

cea.resources.natural_gas module

natural gas

cea.resources.natural_gas.**calc_Cinv_gas**(*PnomGas*, *gV*)
Calculate investment cost of natural gas connections.

Parameters

- **PnomGas** (*float*) – peak natural gas supply in [W]
- **gV** – globalvar.py

Returns InvCa

Rtype InvCa

cea.resources.radiation module

Module contents

cea.technologies package

Submodules

cea.technologies.blinds module

blinds

cea.technologies.blinds.**calc_blinds_activation**(*radiation*, *g_gl*, *Rf_sh*)
This function calculates the blind operation according to ISO 13790.

Parameters

- **radiation** – radiation in [W/m2]
- **g_gl** – window g value
- **Rf_sh** – shading factor

cea.technologies.boilers module

condensing boilers

cea.technologies.boilers.**calc_Cinv_boiler**(*Q_design*, *Q_annual*, *gV*)
Calculates the annual cost of a boiler (based on A+W cost of oil boilers) [CHF / a] and Faz. 2012 data

:type Q_design : float :param Q_design: Design Load of Boiler in [W]

:type Q_annual : float :param Q_annual: Annual thermal load required from Boiler in [Wh]

Parameters **gV** – globalvar.py

:rtype InvCa : float :returns InvCa: Annualized investment costs in CHF/a including Maintenance Cost

cea.technologies.boilers.**calc_Cop_boiler**(*Q_load*, *Q_design*, *T_return_to_boiler*)
This function calculates efficiency for operation of condensing Boilers based on LHV. This efficiency accounts for boiler efficiency only (not plant efficiency!)

operational efficiency after: http://www.greenshootscontrols.net/?p=153

Parameters

- **Q_load** (*float*) – Load of time step
- **Q_design** (*float*) – Design Load of Boiler

:type T_return_to_boiler : float :param T_return_to_boiler: Return Temperature of the network to the boiler [K]

Retype boiler_eff float

Returns boiler_eff efficiency of Boiler (Lower Heating Value), in abs. numbers

cea.technologies.boilers.**cond_boiler_op_cost**(*Q_therm*, *Q_design*, *T_return_to_boiler*, *BoilerFuelType*, *ElectricityType*, *gV*)
Calculates the operation cost of a Condensing Boiler (only operation, not annualized cost)

:type Q_therm : float :param Q_therm: Load of time step

Parameters **Q_design** (*float*) – Design Load of Boiler

:type T_return_to_boiler : float :param T_return_to_boiler: return temperature to Boiler (from DH network)

Parameters **gV** – globalvar.py

:rtype C_boil_therm : float :returns C_boil_therm: Total generation cost for required load (per hour) in CHF

:rtype C_boil_per_Wh : float :returns C_boil_per_Wh: cost per Wh in CHF / kWh

:rtype Q_primary : float :returns Q_primary: required thermal energy per hour (in Wh Natural Gas)

Rtype E_aux_Boiler float

Returns E_aux_Boiler auxiliary electricity of boiler operation

cea.technologies.boilers.**cond_boiler_operation**(*Q_load*, *Q_design*, *T_return_to_boiler*)
This function calculates efficiency for operation of condensing Boilers at DH plant based on LHV. This efficiency accounts for boiler efficiency only (not plant efficiency!)

operational efficiency after: http://www.greenshootscontrols.net/?p=153

Parameters

- **Q_load** (*float*) – Load of time step
- **Q_design** (*float*) – Design Load of Boiler

:type T_return_to_boiler : float :param T_return_to_boiler: Return Temperature of the network to the boiler [K]

Retype boiler_eff float

Returns boiler_eff efficiency of Boiler (Lower Heating Value), in abs. numbers

cea.technologies.chillers module

Vapor-compressor chiller

cea.technologies.chillers.**calc_Cinv_VCC**(*qcold*, *gV*)
Annualized investment costs for the vapor compressor chiller

:type qcold : float :param qcold: peak cooling demand in [W] :param gV: globalvar.py

Returns InvCa annualized chiller investment cost in CHF/a

Rtype InvCa float

cea.technologies.chillers.**calc_VCC**(*mdot*, *tsup*, *tret*, *gV*)
For the operation of a Vapor-compressor chiller between a district cooling network and a condenser with fresh water to a cooling tower following [D.J. Swider, 2003]_.

:type mdot : float :param mdot: plant supply mass flow rate to the district cooling network :type tsup : float :param tsup: plant supply temperature to DCN :type tret : float :param tret: plant return temperature from DCN :param gV: globalvar.py

:rtype wdot : float :returns wdot: chiller electric power requirement :rtype qhotdot : float :returns qhotdot: condenser heat rejection

..[D.J. Swider, 2003] D.J. Swider (2003). A comparison of empirically based steady-state models for vapor-compression liquid chillers. Applied Thermal Engineering.

cea.technologies.cogeneration module

cogeneration (combined heat and power)

cea.technologies.cogeneration.**CC_Op**(*wdot*, *gt_size*, *fuel*, *tDH*, *gV*)
Operation Function of Combined Cycle at given electricity Demand (wdot). The gas turbine (GT) exhaust gas is used by the steam turbine (ST).

:type wdot : float :param wdot: Electric load that is demanded to the gas turbine (only GT output, not CC output!) :type gt_size : float :param gt_size: size of the gas turbine and (not CC)(P_el_max) :type fuel : string :param fuel: fuel used, either 'NG' (natural gas) or 'BG' (biogas) :type tDH : float :param tDH: plant supply temperature to district heating network (hot) :param gV: globalvar.py

:rtype wtot : float :returns wtot: total electric power output from the combined cycle (both GT + ST !) :rtype qdot : float :returns qdot: thermal output from teh combined cycle :rtype eta_elec : float :returns eta_elec: total electric efficiency :rtype eta_heat : float :returns eta_heat: total thermal efficiency :rtype eta_all : float :returns eta_all: sum of total electric and thermal efficiency

cea.technologies.cogeneration.**GT_fullLoadParam**(*gt_size*, *fuel*, *gV*)
Calculates gas turbine efficiency and exhaust gas mass flow rate at full load.

:type gt_size : float :param gt_size: Maximum electric load that is demanded to the gas turbine :type fuel : string :param fuel: fuel used, either NG (Natural Gas) or BG (Biogas) :param gV: globalvar.py

:rtype eta0 : float :returns eta0: efficiency at full load :rtype mdot0 : float :returns mdot0: exhaust gas mass flow rate at full load

..[C. Weber, 2008] C.Weber, Multi-objective design and optimization of district energy systems including poly-generation energy conversion technologies., PhD Thesis, EPFL

cea.technologies.cogeneration.**GT_partLoadParam**(*wdot*, *gt_size*, *eta0*, *mdot0*, *fuel*, *gV*)
Calculates GT operational parameters at part load

:type wdot : float :param wdot: GT electric output (load) :type gt_size : float :param gt_size: Maximum electric load that is demanded to the gas turbine :type eta0 : float :param eta0: GT part-load electric efficiency :type mdot0 : float :param mdot0: GT part-load exhaust gas mass flow :type fuel : string :param fuel: fuel used, either 'NG' (natural gas) or 'BG' (biogas)

:rtype eta : float :returns eta: GT part-load electric efficiency :rtype mdot : float :returns mdot: GT part-load exhaust gas mass flow rate :rtype texh : float :returns texh: exhaust gas temperature :rtype mdotfuel : float :returns mdotfuel: mass flow rate of fuel(gas) requirement

..[C. Weber, 2008] C.Weber, Multi-objective design and optimization of district energy systems including poly-generation energy conversion technologies., PhD Thesis, EPFL

cea.technologies.cogeneration.**ST_Op**(*mdot*, *texh*, *tDH*, *fuel*, *gV*)
Operation of a double pressure (LP,HP) steam turbine connected to a district heating network following [C. Weber, 2008]_

:type mdot : float :param mdot: GT part-load exhaust gas mass flow rate :type texh : float :param texh: GT exhaust gas temperature :type tDH : float :param tDH: plant supply temperature to district heating network (hot) :param fuel: fuel used, either 'NG' (natural gas) or 'BG' (biogas) :type tDH : float :param gV: globalvar.py

:rtype qdot : float :returns qdot: heat power supplied to the DHN :rtype wdotfin : float :returns wdotfin: electric power generated from the steam cycle

..[C. Weber, 2008] C.Weber, Multi-objective design and optimization of district energy systems including poly-generation energy conversion technologies., PhD Thesis, EPFL

cea.technologies.cogeneration.**calc_Cinv_CCT**(*CC_size*, *gV*)
Annualized investment costs for the Combined cycle

:type CC_size : float :param CC_size: Electrical size of the CC

:rtype InvCa : float :returns InvCa: annualized investment costs in CHF

..[C. Weber, 2008] C.Weber, Multi-objective design and optimization of district energy systems including poly-generation energy conversion technologies., PhD Thesis, EPFL

cea.technologies.cogeneration.**calc_Cinv_FC**(*P_design*, *gV*)
Calculates the investment cost of a Fuel Cell in CHF

http://hexis.com/sites/default/files/media/publikationen/140623_hexis_galileo_ibb_profitpaket.pdf?utm_source=HEXIS+Mitarbe
1_Newsletter_2014_Mitarbeitende_DE&utm_medium=email&utm_term=0_e97bc1703e-06d2c528a5-

:type P_design : float :param P_design: Design thermal Load of Fuel Cell [W_th]

Rtype InvCa float

Returns InvCa annualized investment costs in CHF

cea.technologies.cogeneration.**calc_Cop_CCT**(*GT_SIZE*, *T_DH_Supply*, *fuel*, *gV*)
The function iterate the CCT operation between its nominal capacity and minimum load and generate linear functions of the GT operation.

This generated function calculates Operation Point and associated costs of the cogeneration at given thermal load (Q_therm_requested).

How to use the return functions : input Q_therm_requested into the output interpolation functions Conditions: not below or above boundaries Q_therm_min & Q_therm_max

:type GT_SIZE : float :param GT_SIZE: Nominal capacity of Gas Turbine (only GT not cogeneration)

:type T_DH_Supply : float :param T_DH_Supply: CHP plant supply temperature to DHN

:type fuel : string :param fuel: type of fuel, either "NG" or "BG"

Parameters **gV** – globalvar.py

:rtype wdot_interpol : function :returns wdot_interpol: interpolation function for part load electricity requirement for given Q_therm_requested

Rtype Q_used_prim_interpol function

Returns Q_used_prim_interpol interpolation function, primary energy used for given Q_therm_requested

:rtype cost_per_Wh_th_incl_el_interpol : function :returns cost_per_Wh_th_incl_el_interpol: interpolation function, operation cost per thermal energy generated at Q_therm_requested

:rtype Q_therm_min : float :returns Q_therm_min: minimum thermal energy output

:rtype Q_therm_max : float :returns Q_therm_max: maximum thermal energy output

Rtype eta_elec_interpol function

Returns eta_elec_interpol interpolation function, electrical efficiency at Q_therm_requested

..[C. Weber, 2008] C.Weber, Multi-objective design and optimization of district energy systems including polygeneration energy conversion technologies., PhD Thesis, EPFL

cea.technologies.cogeneration.**calc_eta_FC**(*Q_load*, *Q_design*, *phi_threshold*, *approach_call*)

Efficiency for operation of a SOFC (based on LHV of NG) including all auxiliary losses Valid for Q_load in range of 1-10 [kW_el]

Modeled after:

Approach A (NREL Approach): http://energy.gov/eere/fuelcells/distributedstationary-fuel-cell-systems and NREL : p.5 of [M. Zolot et al., 2004]_

Approach B (Empiric Approach): [Iain Staffell]_

:type Q_load : float :param Q_load: Load at each time step

:type Q_design : float :param Q_design: Design Load of FC

:type phi_threshold : float :param phi_threshold: where Maximum Efficiency is reached, used for Approach A

:type approach_call : string :param appraoch_call: choose "A" or "B": A = NREL-Approach, B = Empiric Approach

:rtype eta_el : float :returns eta_el: electric efficiency of FC (Lower Heating Value), in abs. numbers

:rtype Q_fuel : float :returns Q_fuel: Heat demand from fuel (in Watt)

..[M. Zolot et al., 2004] M. Zolot et al., Analysis of Fuel Cell Hybridization and Implications for Energy Storage Devices, NREL, 4th International Advanced Automotive Battery. http://www.nrel.gov/vehiclesandfuels/energystorage/pdfs/36169.pdf

..[Iain Staffell, 2009] Iain Staffell, For Domestic Heat and Power: Are They Worth It?, PhD Thesis, Birmingham: University of Birmingham. http://etheses.bham.ac.uk/641/1/Staffell10PhD.pdf

cea.technologies.controllers module

controllers

cea.technologies.controllers.**calc_simple_temp_control**(*tsd*, *prop_comfort*, *limit_inf_season*, *limit_sup_season*, *weekday*)

cea.technologies.controllers.**calc_simple_ventilation_control**(*ve*, *people*, *Af*, *gv*, *hour_day*, *hour_year*, *n50*)

Modified version of calc_simple_ventilation_control from functions. Fixed infiltration according to schedule is only considered for mechanically ventilated buildings.

ve : required ventilation rate according to schedule (?) people : occupancy schedules (pax?) Af : conditioned floor area (m2) gv : globalvars hour_day : hour of the day [0..23] hour_year : hour of the year [0..8760] n50 : building envelope leakiness from archetypes

q_req : required ventilation rate schedule (m3/s)

cea.technologies.controllers.**calc_ventialtion_HVAC_buildings**(*area_envelope*, *HVAC_on*, *Tin*, *Tout*, *ws*, *method*)

infiltration according to energy plus, blast, or DOE2 tools

cea.technologies.controllers.**temperature_control_tabs**(*bpr*, *tsd*, *hoy*, *gv*, *control*)

Controls for TABS operating temperature based on the operating parameters defined by Koschenz and Lehmann "Thermoaktive Bauteilsysteme (TABS)" (2000), that is: maximum surface temperature and maximum temperature difference between TABS surface and air. If either of these is exceeded, they are set to the maximum and all temperatures are recalculated. The formulas below are simply reformulations of the calculations in the R-C model.

cea.technologies.cooling_tower module

System Modeling: Cooling tower

cea.technologies.cooling_tower.**calc_CT**(*qhotdot*, *Qdesign*, *gV*)

For the operation of a water condenser + direct cooling tower based on [B. Stephane, 2012]_

:type qhotdot : float :param qhotdot: heating power to condenser, From Model_VCC :type Qdesign : float :param Qdesign: Max cooling power

:type wdot : float :param wdot: electric power needed for the variable speed drive fan

..[B. Stephane, 2012] B. Stephane (2012), Evidence-Based Model Calibration for Efficient Building Energy Services. PhD Thesis, University de Liege, Belgium

cea.technologies.cooling_tower.**calc_Cinv_CT**(*CT_size*, *gV*)

Annualized investment costs for the Combined cycle

:type CT_size : float :param CT_size: Size of the Cooling tower in [W]

:rtype InvCa : float :returns InvCa: annualized investment costs in Dollars

cea.technologies.furnace module

furnaces

cea.technologies.furnace.**calc_Cinv_furnace**(*Q_design*, *Q_annual*, *gv*)

Calculates the annualized investment cost of a Furnace based on Bioenergy 2020 (AFO) and POLYCITY Ostfildern

:type Q_design : float :param Q_design: Design Load of Boiler

:type Q_annual : float :param Q_annual: annual thermal Power output [Wh]

Parameters **gV** – globalvar.py

:rtype InvC_return : float :returns InvC_return: total investment Cost for building the plant

:rtype InvCa : float :returns InvCa: annualized investment costs in [CHF] including O&M

cea.technologies.furnace.**calc_eta_furnace**(*Q_load*, *Q_design*, *T_return_to_boiler*, *MOIST_TYPE*, *gv*)

Efficiency for Furnace Plant (Wood Chip CHP Plant, Condensing Boiler) based on LHV.

Capacity : 1-10 [MW], Minimum Part Load: 30% of P_design Source: POLYCITY HANDBOOK 2012

:type Q_load : float :param Q_load: Load of time step

:type Q_design : float :param Q_design: Design Load of Boiler

:type T_return_to_boiler : float :param T_return_to_boiler: return temperature to the boiler

:type MOIST_TYPE : float :param MOIST_TYPE: moisture type of the fuel, set in MasterToSlaveVariables ('wet' or 'dry')

Parameters **gV** – globalvar.py

up to 6MW_therm_out Capacity proven! = 8 MW th (burner)

:rtype eta_therm : float :returns eta_therm: thermal Efficiency of Furnace (LHV), in abs. numbers

:rtype eta_el : float :returns eat_el: electric efficiency of Furnace (LHV), in abs. numbers

:rtyp;e Q_aux : float :returns Q_aux: auxiliary power for Plant operation [W]

cea.technologies.furnace.**furnace_op_cost**(*Q_therm*, *Q_design*, *T_return_to_boiler*, *MOIST_TYPE*, *gv*)

Calculates the operation cost of a furnace plant (only operation, no annualized cost!)

:type Q_therm : float :param Q_therm: thermal energy required from furnace plant in [Wh]

:type Q_design : float :param Q_design: Design Load of Boiler [W]

:type T_return_to_boiler : float :param T_return_to_boiler: return temperature to the boiler

:type MOIST_TYPE : float :param MOIST_TYPE: moisture type of the fuel, set in MasterToSlaveVariables ('wet' or 'dry')

Parameters **gV** – globalvar.py

:rtype C_furn : float :returns C_furn: Total generation cost for required load (per hour) in [CHF], including profits from electricity sold

:rtype C_furn_per_kWh : float :returns C_furn_per_kWh: cost generation per kWh thermal energy produced in [Rp / kWh], including profits from electricity sold

:rtype Q_primary : float :returns Q_primary: required thermal energy per hour [Wh] of wood chips

:rtype E_furn_el_produced : float :returns E_furn_el_produced: electricity produced by furnace plant in [Wh]

cea.technologies.heat_exchangers module

heat exchangers

cea.technologies.heat_exchangers.**calc_Cinv_HEX**(*Q_design*, *gV*)

Calculates the cost of a heat exchanger (based on A+W cost of oil boilers) [CHF / a]

:type Q_design : float :param Q_design: Design Load of Boiler

Parameters **gV** – globalvar.py

:rtype InvC_return : float :returns InvC_return: total investment Cost in [CHF]

:rtype InvCa : float :returns InvCa: annualized investment costs in [CHF/a]

cea.technologies.heating_coils module

Heating and cooling coils of Air handling units

cea.technologies.heating_coils.**calc_cooling_coil**(*Qcsf*, *Qcsf_0*, *Ta_sup_cs*, *Ta_re_cs*, *Tcs_sup_0*, *Tcs_re_0*, *ma_sup_cs*, *ma_sup_0*, *Ta_sup_0*, *Ta_re_0*, *Cpa*, *gv*)

cea.technologies.heating_coils.**calc_heating_coil**(*Qhsf*, *Qhsf_0*, *Ta_sup_hs*, *Ta_re_hs*, *Ths_sup_0*, *Ths_re_0*, *ma_sup_hs*, *ma_sup_0*, *Ta_sup_0*, *Ta_re_0*, *Cpa*, *gv*)

cea.technologies.heatpumps module

heatpumps

cea.technologies.heatpumps.**GHP_InvCost**(*GHP_Size*, *gV*)
Calculates the annualized investment costs for the geothermal heat pump

:type GHP_Size : float :param GHP_Size: Design electrical size of the heat pump in [Wel]

InvCa [float] annualized investment costs in EUROS/a

cea.technologies.heatpumps.**GHP_Op_max**(*tsup*, *tground*, *nProbes*, *gV*)
For the operation of a Geothermal heat pump (GSHP) at maximum capacity supplying DHN.

:type tsup : float :param tsup: supply temperature to the DHN (hot) :type tground : float :param tground: ground temperature :type nProbes: float :param nProbes: bumber of probes :param gV: globalvar.py

Rtype qhotdot float

Returns qhotdot heating energy provided from GHSP

Rtype COP float

Returns COP coefficient of performance of GSHP

cea.technologies.heatpumps.**GHP_op_cost**(*mdot*, *tsup*, *tret*, *gV*, *COP*)
Operation cost of GSHP supplying DHN

:type mdot : float :param mdot: supply mass flow rate to the DHN :type tsup : float :param tsup: supply temperature to the DHN (hot) :type tret : float :param tret: return temeprature from the DHN (cold) :type COP: float :param COP: coefficient of performance of GSHP :param gV: globalvar.py

Rtype C_GHP_el float

Returns C_GHP_el electricity cost of GSHP operation

Rtype wdot float

Returns wdot electricty required for GSHP operation

Rtype qcoldot float

Returns qcoldot cold power requirement

Rtype q_therm float

Returns q_therm thermal energy supplied to DHN

cea.technologies.heatpumps.**HPLake_Op**(*mdot*, *tsup*, *tret*, *tlake*, *gV*)
For the operation of a Heat pump between a district heating network and a lake

:type mdot : float :param mdot: supply mass flow rate to the DHN :type tsup : float :param tsup: supply temperature to the DHN (hot) :type tret : float :param tret: return temeprature from the DHN (cold) :type tlake : float :param tlake: lake temperature :param gV: globalvar.py

:rtype wdot_el : float :returns wdot_el: total electric power requirement for compressor and auxiliary el. :rtype qcolddot : float :returns qcolddot: cold power requirement

..[L. Girardin et al., 2010] L. Girardin, F. Marechal, M. Dubuis, N. Calame-Darbellay, D. Favrat (2010). EnerGis: a geographical information based system for the evaluation of integrated energy conversion systems in urban areas, Energy.

..[C. Montagud et al., 2014] C. Montagud, J.M. Corberan, A. Montero (2014). In situ optimization methodology for the water circulation pump frequency of ground source heat pump systems. Energy and Buildings

cea.technologies.heatpumps.**HPLake_op_cost**(*mdot*, *tsup*, *tret*, *tlake*, *gV*)
For the operation of lake heat pump supplying DHN

:type mdot : float :param mdot: supply mass flow rate to the DHN :type tsup : float :param tsup: supply temperature to the DHN (hot) :type tret : float :param tret: return temeprature from the DHN (cold) :type tlake : float :param tlake: lake temperature :param gV: globalvar.py

Rtype C_HPL_el float

Returns C_HPL_el electricity cost of Lake HP operation

Rtype wdot float

Returns wdot electricty required for Lake HP operation

Rtype Q_cold_primary float

Returns Q_cold_primary cold power requirement

Rtype Q_therm float

Returns Q_therm thermal energy supplied to DHN

cea.technologies.heatpumps.**HPSew_op_cost**(*mdot*, *tsup*, *tret*, *tsupsew*, *gV*)
Operation cost of sewage water HP supplying DHN

:type mdot : float :param mdot: supply mass flow rate to the DHN :type tsup : float :param tsup: supply temperature to the DHN (hot) :type tret : float :param tret: return temeprature from the DHN (cold) :type tsupsew : float :param tsupsew: sewage supply temperature :param gV: globalvar.py

Rtype C_HPSew_el_pure float

Returns C_HPSew_el_pure electricity cost of sewage water HP operation

Rtype C_HPSew_per_kWh_th_pure float

Returns C_HPSew_per_kWh_th_pure electricity cost per kWh thermal energy produced from sewage water HP

Rtype qcoldot float

Returns qcoldot cold power requirement

Rtype q_therm float

Returns q_therm thermal energy supplied to DHN

Rtype wdot float

Returns wdot electricty required for sewage water HP operation

..[L. Girardin et al., 2010] L. Girardin, F. Marechal, M. Dubuis, N. Calame-Darbellay, D. Favrat (2010). EnerGis: a geographical information based system for the evaluation of integrated energy conversion systems in urban areas, Energy.

cea.technologies.heatpumps.**calc_Cinv_GHP**(*GHP_Size*, *gV*)
Calculates the annualized investment costs for the geothermal heat pump

:type GHP_Size : float :param GHP_Size: Design electrical size of the heat pump in [Wel]

:type InvCa : float :returns InvCa: annualized investment costs in [EUROS/a]

..[D. Bochatay et al., 2005] D. Bochatay, I. Blanc, O. Jolliet, F. Marechal, T. Manasse-Ratmandresy (2005). Project PACOGEN Evaluation economique et environmentale de systemes energetiques a usage residentiel., EPFL.

cea.technologies.heatpumps.**calc_Cinv_HP**(*HP_Size*, *gV*)
Calculates the annualized investment costs for the heat pump

:type HP_Size : float :param HP_Size: Design thermal size of the heat pump in [W]

:rtype InvCa : float :returns InvCa: annualized investment costs in [CHF/a]

..[C. Weber, 2008] C.Weber, Multi-objective design and optimization of district energy systems including polygeneration energy conversion technologies., PhD Thesis, EPFL

cea.technologies.heatpumps.**calc_Cop_GHP**(*mdot*, *tsup*, *tret*, *tground*, *gV*)
For the operation of a Geothermal heat pump (GSHP) supplying DHN.

:type mdot : float :param mdot: supply mass flow rate to the DHN :type tsup : float :param tsup: supply temperature to the DHN (hot) :type tret : float :param tret: return temeprature from the DHN (cold) :type tground : float :param tground: ground temperature :param gV: globalvar.py

:rtype wdot_el : float :returns wdot_el: total electric power requirement for compressor and auxiliary el. :rtype qcolddot : float :returns qcolddot: cold power requirement :rtype qhotdot_missing : float :returns qhotdot_missing: deficit heating energy from GSHP :rtype tsup2 : :returns tsup2: supply temperature after HP (to DHN)

..[O. Ozgener et al., 2005] O. Ozgener, A. Hepbasli (2005). Experimental performance analysis of a solar assisted ground-source heat pump greenhouse heating system, Energy Build. ..[C. Montagud et al., 2014] C. Montagud, J.M. Corberan, A. Montero (2014). In situ optimization methodology for the water circulation pump frequency of ground source heat pump systems. Energy and Buildings

cea.technologies.photovoltaic module

photovoltaic

cea.technologies.photovoltaic.**Calc_PV_power**(*S*, *Tcell*, *eff_nom*, *areagroup*, *Bref*, *misc_losses*)

cea.technologies.photovoltaic.**Calc_Sm_PV**(*te*, *I_sol*, *I_direct*, *I_diffuse*, *tilt*, *Sz*, *teta*, *tetad*, *tetaeg*, *n*, *Pg*, *K*, *NOCT*, *a0*, *a1*, *a2*, *a3*, *a4*, *L*)

cea.technologies.photovoltaic.**Calc_diffuseground_comp**(*tilt_radians*)

cea.technologies.photovoltaic.**calc_Cinv_pv**(*P_peak*)
P_peak in kW result in CHF Lifetime 20 y

cea.technologies.photovoltaic.**calc_Crem_pv**(*E_nom*)
Calculates KEV (Kostendeckende Einspeise - Verguetung) for solar PV and PVT. Therefore, input the nominal capacity of EACH installation and get the according KEV as return in Rp/kWh

Parameters **E_nom** – Nominal Capacity of solar panels (PV or PVT) in Wh

Returns KEV_obtained_in_RpPerkWh : float KEV remuneration in Rp / kWh

cea.technologies.photovoltaic.**calc_properties_PV**(*type_PVpanel*)

cea.technologies.photovoltaic.**calc_pv_generation**(*type_panel*, *hourly_radiation*, *Number_groups*, *number_points*, *prop_observers*, *weather_data*, *g*, *Sz*, *Az*, *ha*, *latitude*, *misc_losses*)

cea.technologies.photovoltaic.**calc_pv_main**(*locator*, *sensors_data*, *radiation*, *latitude*, *longitude*, *year*, *gv*, *weather_path*)

cea.technologies.photovoltaic.**test_photovoltaic**()

cea.technologies.photovoltaic_thermal module

Photovoltaic thermal panels

cea.technologies.photovoltaic_thermal.**Calc_PVT_module**(*tilt_angle*, *IAM_b_vector*, *I_direct_vector*, *I_diffuse_vector*, *Te_vector*, *n0*, *c1*, *c2*, *mB0_r*, *mB_max_r*, *mB_min_r*, *C_eff*, *t_max*, *IAM_d*, *Area_a*, *dP1*, *dP2*, *dP3*, *dP4*, *Tin*, *Leq*, *Le*, *Nseg*, *eff_nom*, *Bref*, *Sm_pv*, *Tcell_pv*, *misc_losses*, *area-group*)

cea.technologies.photovoltaic_thermal.**calc_Cinv_PVT**(*P_peak*)
P_peak in kW result in CHF

cea.technologies.photovoltaic_thermal.**calc_PVT**(*locator*, *sensors_data*, *radiation*, *latitude*, *longitude*, *year*, *gv*, *weather_path*)

cea.technologies.photovoltaic_thermal.**calc_PVT_generation**(*type_panel*, *hourly_radiation*, *Number_groups*, *number_points*, *prop_observers*, *weather_data*, *g*, *Sz*, *Az*, *ha*, *latitude*, *misc_losses*, *type_SCpanel*, *Tin*, *height*)

cea.technologies.photovoltaic_thermal.**test_PVT**()

cea.technologies.pumps module

pumps

cea.technologies.pumps.**Pump_Cost**(*deltaP*, *mdot*, *eta_pumping*, *gV*)
Calculates the cost of a pumping device. if the nominal load (electric) > 375kW, a new pump is installed if the nominal load (electric) < 500W, a pump with Pel_design = 500W is assumed

Investment costs are calculated upon the life time of a GHP (20y) and a GHP- related interest rate of 6%

:type deltaP : float :param deltaP: nominal pressure drop that has to be overcome with the pump

:type mdot : float :param mdot: nominal mass flow

:type eta_pumping : float :param pump efficiency: (set 0.8 as standard value, eta = E_pumping / E_elec)

:rtype InvCa : float :returns InvCa: annualized investment costs in CHF/year

cea.technologies.pumps.**Pump_operation**(*P_design*)

Modeled after: 05_merkblatt_wirtschaftlichkeit_14.pdf 23_merkblatt_pumpen_web.pdf ER_2010_11_Heizungspumpen.pdf MerkblattPreiseFU2010_2011.pdf MerkblattPreiseMotoren2010_2011.pdf

P_design [float] Load of time step

eta_el [float] electric efficiency of Pumping operation in abs. numbers (e.g. 0.93)

cea.technologies.pumps.**calc_Cinv_pump**(*deltaP*, *mdot*, *eta_pumping*, *gV*)
Calculates the cost of a pumping device. if the nominal load (electric) > 375kW, a new pump is installed if the nominal load (electric) < 500W, a pump with Pel_design = 500W is assumed

Investement costs are calculated upon the life time of a GHP (20y) and a GHP- related interest rate of 6%

:type deltaP : float :param deltaP: nominal pressure drop that has to be overcome with the pump

:type mdot : float :param mdot: nominal mass flow

:type eta_pumping : float :param pump efficiency: (set 0.8 as standard value, eta = E_pumping / E_elec)

:rtype InvC_return : float :returns InvC_return: total investment Cost in CHF

:rtype InvCa : float :returns InvCa: annualized investment costs in CHF/year

cea.technologies.pumps.**calc_Ctot_pump**(*dicoSupply*, *buildList*, *network_results_folder*, *ntwFeat*, *gV*)
Computes the total pump investment cost

:type dicoSupply : class context :type buildList : list :param buildList: list of buildings in the district :type network_results_folder : string :param network_results_folder: path to network results folder :type ntwFeat : class ntwFeatures

:rtype pumpCosts : float :returns pumpCosts: pumping cost

cea.technologies.radiators module

heating radiators

cea.technologies.radiators.**calc_radiator**(*Qh*, *tair*, *Qh0*, *tair0*, *tsh0*, *trh0*)

cea.technologies.radiators.**fh**(*x*, *mCw0*, *k2*, *Qh0*, *tair*, *LMRT*, *nh*)

cea.technologies.radiators.**lmrt**(*tair0*, *trh0*, *tsh0*)

cea.technologies.sewage_heat_exchanger module

Sewage source heat exchanger

cea.technologies.sewage_heat_exchanger.**calc_Sewagetemperature**(*Qwwf*, *Qww*, *tsww*, *trww*, *totwater*, *mcpww*, *cp*, *density*, *SW_ratio*)

Calculate sewage temperature and flow rate released from DHW usages and Fresh Water (FW) in buildings.

Parameters

- **Qwwf** (*float*) – final DHW heat requirement
- **Qww** (*float*) – DHW heat requirement
- **tsww** (*float*) – DHW supply temperature
- **trww** (*float*) – DHW return temperature
- **totwater** (*float*) – fresh water flow rate
- **mcpww** (*float*) – DHW heat capacity
- **cp** (*float*) – water specific heat capacity
- **density** – water density
- **SW_ratio** (*float*) – ratio of waste water to fresh water production.

Returns mcp_combi sewage water heat capacity [kWh/K]

Rtype mcp_combi float

Returns t_to_sewage sewage water temperature

Rtype t_to_sewage float

cea.technologies.sewage_heat_exchanger.**calc_sewage_heat_exchanger**(*locator*, *Length_HEX_available*, *gv*)

Calaculate the heat extracted from the sewage HEX.

Parameters

- **locator** – an InputLocator instance set to the scenario to work on
- **Length_HEX_available** (*float*) – HEX length available
- **gv** – globalvar.py

Save the results to *SWP.csv*

cea.technologies.sewage_heat_exchanger.**calc_sewageheat**(*mcp*, *tin*, *w_HEX*, *Vf*, *cp*, *h0*, *min_m*, *L_HEX*, *tmin*, *ATmin*)

Calculates the operation of sewage heat exchanger.

Parameters

- **mcp** (*float*) – heat capacity of total sewage in a zone
- **tin** (*float*) – sewage inlet temperature of a zone
- **w_HEX** (*float*) – width of the sewage HEX
- **Vf** (*float*) – sewage flow rate [m/s]

- **cp** (*float*) – water specific heat capacity
- **h0** (*float*) – sewage heat transfer coefficient
- **min_m** (*float*) – sewage minimum flow rate in [lps]
- **L_HEX** (*float*) – HEX length available
- **tmin** (*float*) – minimum temperature of extraction
- **ATmin** (*float*) – minimum area of heat exchange

Returns Q_source heat supplied by sewage

Return type float

Returns t_source sewage heat supply temperature

Rtype t_source float

Returns tb2 sewage return temperature

Rtype tbs float

Returns ta1 temperature inlet of the cold stream (from the HP)

Rtype ta1 float

Returns ta2 temperature outlet of the cold stream (to the HP)

Rtype ta2 float

..[J.A. Fonseca et al., 2016] J.A. Fonseca, Thuy-An Nguyen, Arno Schlueter, Francois Marechal (2016). City Enegy Analyst (CEA): Integrated framework for analysis and optimization of building energy systems in neighborhoods and city districts. Energy and Buildings.

cea.technologies.sewage_heat_exchanger.**test_ss_heatpump**()

cea.technologies.solar_collector module

solar collectors

cea.technologies.solar_collector.**Calc_SC_module2**(*radiation*, *tilt_angle*, *IAM_b_vector*, *I_direct_vector*, *I_diffuse_vector*, *Te_vector*, *n0*, *c1*, *c2*, *mB0_r*, *mB_max_r*, *mB_min_r*, *C_eff*, *t_max*, *IAM_d*, *Area_a*, *dP1*, *dP2*, *dP3*, *dP4*, *Tin*, *Leq*, *Le*, *Nseg*)

cea.technologies.solar_collector.**Calc_incidenteangleB**(*g*, *lat*, *ha*, *tilt*, *teta_z*)

cea.technologies.solar_collector.**Calc_qloss_net**(*Mfl*, *Le*, *Area_a*, *Tm*, *Te*, *maxmsc*)

cea.technologies.solar_collector.**SC_generation**(*type_SCpanel*, *group_radiation*, *prop_observers*, *number_points*, *weather_data*, *g*, *Sz*, *Az*, *ha*, *latitude*, *Tin*, *height*)

cea.technologies.solar_collector.**Selectminimumenergy2**(*m*, *q*, *dp*)

cea.technologies.solar_collector.**SelectminimumenergySc**(*q1*, *q2*, *q3*, *q4*, *E1*, *E2*, *E3*, *E4*, *m1*, *m2*, *m3*, *m4*, *dP1*, *dP2*, *dP3*, *dP4*, *Area_a*)

cea.technologies.solar_collector.**calc_Cinv_SC**(*Area*)
Lifetime 35 years

cea.technologies.solar_collector.**calc_Eaux_SC**(*qV_des*, *Dp_collector*, *Leq*, *Aa*)
auxiliary electricity solar collectort

Parameters

- **qV_des** –
- **Dp_collector** –
- **Leq** –
- **Aa** –

Returns

cea.technologies.solar_collector.**calc_SC**(*locator*, *sensors_data*, *radiation*, *latitude*, *longitude*, *year*, *gv*, *weather_path*)

cea.technologies.solar_collector.**calc_anglemodifierSC**(*Az_vector*, *g_vector*, *ha_vector*, *teta_z*, *tilt_angle*, *type_SCpanel*, *latitude*, *Sz_vector*)

cea.technologies.solar_collector.**calc_groups**(*Clean_hourly*, *observers_fin*)

cea.technologies.solar_collector.**calc_properties_SC**(*type_SCpanel*)
properties of module :param type_SCpanel: :return:

cea.technologies.solar_collector.**calc_qgain**(*Tfl*, *Tabs*, *qrad*, *DT*, *TinSub*, *Tout*, *Aseg*, *c1*, *c2*, *Mfl*, *delts*, *Cpwg*, *C_eff*, *Te*)

cea.technologies.solar_collector.**calc_qrad**(*n0*, *IAM_b*, *I_direct*, *IAM_d*, *I_diffuse*, *tilt*)

cea.technologies.solar_collector.**optimal_angle_and_tilt**(*observers_all*, *latitude*, *worst_sh*, *worst_Az*, *transmittivity*, *grid_side*, *module_lenght*, *angle_north*, *Min_Isol*, *Max_Isol*)

cea.technologies.solar_collector.**test_solar_collector**()

cea.technologies.storagetank module

Sensible Heat Storage - Fully Mixed tank

cea.technologies.storagetank.**calc_Qww_ls_st**(*ta*, *te*, *Tww_st*, *V*, *Qww*, *Qww_ls_r*, *Qww_ls_nr*, *gv*)
This algorithm calculates the heat flows within a fully mixed water storage tank. Heat flows include sensible heat loss to the environment (ql), heat charged into the tank (qc), and heat discharged from the tank (qd).

Parameters

- **Tww_st** (*float*) – tank temperature in [C]
- **Tww_setpoint** (*float*) – DHW temperature set point in [C]
- **ta** (*float*) – room temperature in [C]
- **te** (*float*) – ambient temperature in [C]
- **V** (*float*) – DHW tank size in [m3]
- **Qww** (*float*) – DHW demand in [W]
- **Qww_ls_r** – recoverable loss in distribution in [W]

- **Qww_ls_nr** (*float*) – non-recoverable loss in distribution in [W]
- **gv** – globalvar.py

Return ql storage sensible heat loss in [W].

Return qd heat discharged from the tank in [W], including dhw heating demand and distribution heat loss.

Return qc heat charged into the tank in [W].

Rtype ql float

Rtype qd float

Rtype qc float

cea.technologies.storagetank.**ode**(*y*, *t*, *ql*, *qd*, *qc*, *Pwater*, *Cpw*, *Vtank*)
This algorithm describe the energy balance of the dhw tank with a differential equation.

Parameters

- **y** (*float*) – storage sensible temperature in K.
- **t** (*float*) – time steps.
- **ql** (*float*) – storage tank sensible heat loss in W.
- **qd** (*float*) – heat discharged from the tank in W.
- **qc** (*float*) – heat charged into the tank in W.

Return dydt change in temperature at each time step.

cea.technologies.storagetank.**solve_ode_storage**(*Tww_st_0*, *ql*, *qd*, *qc*, *Vtank*, *gv*)
This algorithm solves the differential equation, ode.

Parameters

- **Tww_st_0** (*float*) – initial tank temperature in [C]
- **ql** (*float*) – storage tank sensible heat loss in W.
- **qd** (*float*) – heat discharged from the tank in W.
- **qc** (*float*) – heat charged into the tank in W.
- **Vtank** (*float*) – DHW tank size in [m3]
- **gv** – globalvar.py

Returns y[1] solution of the ode

Rtype y[1] float

cea.technologies.substation module

Substation Model

cea.technologies.substation.**calc_DC_supply**(*t_0*, *t_1*)
This function calculates the temperature of the district cooling network according to the minimum observed (different to zero) in all buildings connected to the grid.

Parameters

- **t_0** – last minimum temperature
- **t_1** – current minimum temperature to evaluate

Returns tmin, new minimum temperature

cea.technologies.substation.**calc_DH_supply**(*t_0*, *t_1*)
This function calculates the temperature of the district heating network according to the maximum observed in all buildings connected to the grid. :param t_0: last maximum temperature :param t_1: current maximum temperature :return: tmax, new maximum temperature

cea.technologies.substation.**calc_HEX_cooling**(*Q*, *UA*, *thi*, *tho*, *tci*, *ch*)
This function calculates the mass flow rate, temperature of return (secondary side) and heat exchanger area for a plate heat exchanger. Method of Number of Transfer Units (NTU)

Parameters

- **Q** – cooling load
- **UA** – coefficient representing the area of heat exchanger times the coefficient of transmittance of the heat exchanger
- **thi** – in temperature of primary side
- **tho** – out temperature of primary side
- **tci** – in temperature of secondary side
- **ch** – capacity mass flow rate primary side

Returns

- tco, out temperature of secondary side (district cooling network)
- cc, capacity mass flow rate secondary side

cea.technologies.substation.**calc_HEX_heating**(*Q*, *UA*, *thi*, *tco*, *tci*, *cc*)
This function calculates the mass flow rate, temperature of return (secondary side) and heat exchanger area for a shell-tube pleat exchanger in the heating case.

Method of Number of Transfer Units (NTU)

Parameters

- **Q** – load
- **UA** – coefficient representing the area of heat exchanger times the coefficient of transmittance of the heat exchanger
- **thi** – in temperature of secondary side
- **tco** – out temperature of primary side
- **tci** – in temperature of primary side
- **cc** – capacity mass flow rate primary side

Returns

- tho, out temperature of secondary side (district cooling network)
- ch, capacity mass flow rate secondary side

cea.technologies.substation.**calc_HEX_mix**(*Q1*, *Q2*, *t1*, *m1*, *t2*, *m2*)
This function computes the average temperature between two vectors of heating demand. In this case, domestic hotwater and space heating.

Parameters

- **Q1** – load heating
- **Q2** – load domestic hot water

- **t1** – out temperature of heat exchanger for space heating
- **m1** – mas flow rate secondary side of heat exchanger for space heating
- **t2** – out temperature of heat exchanger for domestic hot water
- **m2** – mas flow rate secondary side of heat exchanger for domestic hot water

Returns tavg: average out temperature.

cea.technologies.substation.**calc_area_HEX**(*Qnom*, *dTm_0*, *U*)
Thi function calculates the area of a het exchanger at nominal conditions.

Parameters

- **Qnom** – nominal load
- **dTm_0** – nominal logarithmic temperature difference
- **U** – coeffiicent of transmissivity

Returns

- area, area of heat exchange
- UA, coefficient representing the area of heat exchanger times the coefficient of transmittance of the heat exchanger

cea.technologies.substation.**calc_dTm_HEX**(*thi*, *tho*, *tci*, *tco*, *flag*)
This function estimates the logarithmic temperature difference between two streams

Parameters

- **thi** – in temperature hot stream
- **tho** – out temperature hot stream
- **tci** – in temperature cold stream
- **tco** – out temperature cold stream
- **flag** – heat: when using for the heating case, 'cool' otherwise

Returns dtm = logaritimic temperature difference

cea.technologies.substation.**calc_plate_HEX**(*NTU*, *cr*)
This function calculates the efficiency of exchange for a plate heat exchanger according to the NTU method of AShRAE 90.1

Parameters

- **NTU** – number of transfer units
- **cr** – ratio between min and max capacity mass flow rates

Returns eff: efficiency of heat exchange

cea.technologies.substation.**calc_shell_HEX**(*NTU*, *cr*)
This function calculates the efficiency of exchange for a tube-shell heat exchanger according to the NTU method of AShRAE 90.1

Parameters

- **NTU** – number of transfer units
- **cr** – ratio between min and max capacity mass flow rates

Returns eff: efficiency of heat exchange

cea.technologies.substation.**calc_substation_cooling**(*Q*, *thi*, *tho*, *tci*, *ch*, *ch_0*, *Qnom*, *thi_0*, *tci_0*, *tho_0*, *gv*)

this function calculates the state of the heat exchanger at the substation of every customer with cooling needs

Parameters

- **Q** – cooling laad
- **thi** – in temperature of primary side
- **tho** – out temperature of primary side
- **tci** – in temperature of secondary side
- **ch** – capacity mass flow rate primary side
- **ch_0** – nominal capacity mass flow rate primary side
- **Qnom** – nominal cooling load
- **thi_0** – nominal in temperature of primary side
- **tci_0** – nominal in temperature of secondary side
- **tho_0** – nominal out temperature of primary side
- **gv** – path to global variables class

Returns tco = out temperature of secondary side (district cooling network) cc = capacity mass flow rate secondary side Area_HEX_cooling = are of heat excahnger.

cea.technologies.substation.**calc_substation_heating**(*Q*, *thi*, *tco*, *tci*, *cc*, *cc_0*, *Qnom*, *thi_0*, *tci_0*, *tco_0*, *gv*)

This function calculates the mass flow rate, temperature of return (secondary side) and heat exchanger area of every substation.

Parameters

- **Q** – heating load
- **thi** – in temperature of secondary side
- **tco** – out temperature of primary side
- **tci** – in temperature of primary side
- **cc** – capacity mass flow rate primary side
- **cc_0** – nominal capacity mass flow rate primary side
- **Qnom** – nominal cooling load
- **thi_0** – nominal in temperature of secondary side
- **tci_0** – nominal in temperature of primary side
- **tco_0** – nominal out temperature of primary side
- **gv** – path to global variables class

Returns tho = out temperature of secondary side (district cooling network) ch = capacity mass flow rate secondary side Area_HEX_heating = are of heat excahnger.

cea.technologies.substation.**substation_main**(*locator*, *total_demand*, *building_names*, *gv*, *Flag*)

this function calculates the temperatures and mass flow rates of the district heating network at every costumer. Based on this, the script calculates the hourly temperature of the network at the plant. This temperature needs to be equal to that of the customer with the highest temperature requirement plus thermal losses in the network.

Parameters

- **locator** – path to locator function
- **total_demand** – dataframe with total demand and names of all building in the area
- **building_names** – dataframe with names of all buildings in the area
- **gv** – path to global variables class
- **Flag** – boolean, True if the function is called by the master optimizaiton. False if the fucntion is called during preprocessing

cea.technologies.substation.**substation_model**(*locator*, *gv*, *building*, *t_DH*, *t_DH_supply*, *t_DC_supply*, *t_HS*, *t_WW*)

Parameters

- **locator** – path to locator function
- **gv** – path to global variables class
- **building** – dataframe with consumption data per building
- **t_DH** – vector with hourly temperature of the district heating network without losses
- **t_DH_supply** – vector with hourly temperature of the district heating netowork with losses
- **t_DC_supply** – vector with hourly temperature of the district coolig network with losses
- **t_HS** – maximum hourly temperature for all buildings connected due to space heating
- **t_WW** – maximum hourly temperature for all buildings connected due to domestic hot water

Returns

Dataframe stored for every building with the mass flow rates and temperatures district heating and cooling side in:

where fName_result: ID of the building accounting for the individual at which it belongs to.

cea.technologies.tabs module

Termoactivated building surfaces (TABS)

cea.technologies.tabs.**calc_floorheating**(*Qh*, *tm*, *Qh0*, *tsh0*, *trh0*, *Af*)

Calculates the operating conditions of the TABS system based on existing radiator model, replacing the radiator equation with the simple calculation for TABS from SIA 2044, which in turn is based on Koschenz & Lehmann "Thermoaktive Bauteilsysteme (TABS)".

Parameters

- **Qh** – heating demand
- **tm** – Temperature of the thermal mass
- **Qh0** – nominal heating power of the heating system
- **tsh0** – nominal supply temperature to the TABS system
- **trh0** – nominal return temperature from the TABS system
- **Af** – heated area

Returns

- tsh, supply temperature to the TABS system
- trh, return temperature from the TABS system
- mCw, flow rate in the TABS system

cea.technologies.thermal_network module

hydraulic network

cea.technologies.thermal_network.**calc_Cinv_network_linear**(*LengthNetwork*, *gV*)
calculate annualised network investment cost with a linearized function.

Parameters **LengthNetwork** – total length of the network in [m]

Pram gV globalvar.py

Returns InvCa annualised investment cost of the thermal network

Rtype InvCa float

cea.technologies.thermal_storage module

thermal storage

cea.technologies.thermal_storage.**calc_Cinv_storage**(*vol*, *gV*)
calculate the annualized investment cost of a thermal storage tank

Parameters

- **vol** (*float*) – storage tank volume
- **gV** – global.var

Returns InvCa

Module contents

cea.utilities package

Submodules

cea.utilities.building_info module

cea.utilities.compile_pyd_files module

cea.utilities.epwreader module

Energyplus file reader

cea.utilities.epwreader.**calc_skytemp**(*Tdrybulb*, *Tdewpoint*, *N*)

cea.utilities.epwreader.**epw_reader**(*weather_path*)

cea.utilities.epwreader.**test_reader**()

cea.utilities.helpers module

Some helper functions that could be useful

cea.utilities.helpers.**check_doy**(*doy*)
check for day of year within bounds doy: day of year bool

cea.utilities.helpers.**check_hoy**(*hoy*)
check for hour of year within bounds hoy: hour of year bool

cea.utilities.helpers.**doy_2_hoy**(*doy*)
Day of year to hour of year

doy: day of year hoy: hour of year

cea.utilities.helpers.**hoy_2_dom**(*hoy*)
hour of year to day of month hoy: hour of year dom: day of month

cea.utilities.helpers.**hoy_2_doy**(*hoy*)
Hour of year to day of year hoy: hour of year doy: day of year

cea.utilities.helpers.**hoy_2_hod**(*hoy*)
hour of year to hour of day hoy: hour of year hod: hour of day

cea.utilities.helpers.**hoy_2_moy**(*hoy*)
hour of year to month of year hoy: hour of year moy: month of year

cea.utilities.helpers.**hoy_2_seasonhour**(*hoy*, *gv*)
hour of year to hour relative to start of heating season hoy: hour of year seasonhour: hour relative to start of heating season

cea.utilities.helpers.**hoy_2_woy**(*hoy*)
Hour of year to week of year hoy: hour of year woy: weak of year

cea.utilities.helpers.**is_coolingseason_hoy**(*hoy*)
checks if a certain hour of the year is part of the cooling season or not hoy : hour of year bool

cea.utilities.helpers.**is_daytime_hoy**(*hoy*)
Check if a certain hour of the year is during the daytime or not hoy : hour of year bool

cea.utilities.helpers.**is_heatingseason_hoy**(*hoy*)
checks if a certain hour of the year is part of the heating season or not hoy : hour of year bool

cea.utilities.helpers.**is_nighttime_hoy**(*hoy*)
Check if a certain hour of year is during night or not hoy: hour of year bool

cea.utilities.helpers.**seasonhour_2_hoy**(*seasonhour*, *gv*)
hour relative to start of heating season to hour of year seasonhour: hour relative to start of heating season hoy: hour of year

cea.utilities.helpers.**test_helpers**()
test helpers

cea.utilities.physics module

Physical functions

cea.utilities.physics.**calc_RH**(*w*, *t*)

cea.utilities.physics.**calc_h**(*t*, *w*)
calculates enthalpy of moist air in kJ/kg

t : air temperature in (°C) w : moisture content of air in (kg/kg dry air)

h : enthalpy of moist air in (kJ/kg)

cea.utilities.physics.**calc_rho_air**(*temp_air*)
Calculation of density of air according to 6.4.2.1 in [1]

temp_air : air temperature in (°C)

rho_air : air density in (kg/m3)

cea.utilities.physics.**calc_t**(*w*, *RH*)

cea.utilities.physics.**calc_t_from_h**(*h*, *w*)
calculates temperature in (°C) from enthalpy (kJ/kg) of moist air with know moisture content inverse equation of calc_h(t,w)

Parameters

- **h** – enthalpy of moist air in (kJ/kg)
- **w** – moisture content of air in (kg/kg dry air)

Returns temperature in (°C)

cea.utilities.physics.**calc_w**(*t*, *RH*)
Moisture content in kg/kg of dry air

t : temperature of air in (°C) RH : relative humidity of air in (%)

w : moisture content of air in (kg/kg dry air)

cea.utilities.reporting module

Functions for Report generation

cea.utilities.reporting.**full_report_to_xls**(*tsd*, *output_folder*, *basename*, *gv*)
this function is to write a full report to an `*.xls` file containing all intermediate and final results of a single building thermal loads calculation

cea.utilities.solar_equations module

solar equations

cea.utilities.solar_equations.**calc_sun_properties**(*latitude*, *longitude*, *weather_data*, *gv*)

cea.utilities.solar_equations.**calc_sunrise**(*sunrise*, *Yearsimul*, *longitude*, *latitude*)

cea.utilities.solar_equations.**declination_degree**(*when*, *TY*)
The declination of the sun is the angle between Earth's equatorial plane and a line between the Earth and the sun. It varies between 23.45 degrees and -23.45 degrees, hitting zero on the equinoxes and peaking on the solstices. [1]

Parameters

- **when** – datetime.datetime, date/time for which to do the calculation
- **TY** – float, Total number of days in a year. eg. 365 days per year,(no leap days)
- **DEC** – float, The declination of the Sun

cea.utilities.solar_equations.**get_hour_angle**(*when*, *longitude_deg*)

[1] http://pysolar.org/

cea.utilities.solar_equations.**get_solar_time**(*longitude_deg*, *when*)
returns solar time in hours for the specified longitude and time, accurate only to the nearest minute.

cea.utilities.solar_equations.**pyephem**(*time*, *latitude*, *longitude*, *altitude=0*, *pressure=101325*, *temperature=12*)

Module contents

Submodules

cea.globalvar module

Global variables - this object contains context information and is expected to be refactored away in future.

class cea.globalvar.**GlobalVariables**
Bases: object

is_heating_season(*timestep*)

log(*msg*, ***kwargs*)

report(*tsd*, *output_folder*, *basename*)
Use vars to fill worksheets in an excel file $destination_template based on the template. The template references self.report_variables. The destination_template may contain date format codes that will be updated with the current datetime.

cea.inputlocator module

inputlocator.py - locate input files by name based on the reference folder structure.

class cea.inputlocator.**InputLocator**(*scenario_path*)
Bases: object

The InputLocator locates files and folders for input to the scripts. This works, because we have a convention for the folder structure of a scenario. It also provides locations of other files, such as those in the databases folder (e.g. archetypes).

get_archetypes_properties()
databases/CH/Archetypes/Archetypes_properties.xlsx path to database of archetypes file Archetypes_properties.xlsx

get_archetypes_schedules()
databases/CH/Archetypes/Archetypes_schedules.xlsx path to database of archetypes file Archetypes_HVAC_properties.xlsx

get_benchmark_plots_file()
scenario/outputs/plots/graphs/Benchmark_scenarios.pdf

get_building_age()
scenario/inputs/building-properties/age.dbf

get_building_architecture()
scenario/inputs/building-properties/architecture.dbf This file is generated by the properties script. This file is used in the embodied energy script (cea/embodied.py) and the demand script (cea/demand_main.py)

get_building_comfort()
scenario/inputs/building-properties/indoor_comfort.dbf

get_building_geometry()
scenario/inputs/building-geometry/zone.shp

get_building_hvac()
scenario/inputs/building-properties/technical_systems.dbf

get_building_internal()
scenario/inputs/building-properties/internal_loads.dbf

get_building_occupancy()
scenario/inputs/building-properties/building_occupancy.dbf

get_building_overrides()
scenario/inputs/building-properties/overrides.csv This file contains overrides to the building properties input files. They are applied after reading those files and are matched by column name.

get_building_supply()
scenario/inputs/building-properties/building_supply.dbf

get_calibration_cluster(*sax_name*)
scenario/outputs/data/demand/{sax_name}.csv

get_calibration_cluster_opt_checkpoint(*generation*)
scenario/outputs/data/demand/{sax_name}.csv

get_calibration_clusters_names()
scenario/outputs/data/demand/{sax_name}.csv

get_calibration_folder()
scenario/outputs/data/calibration

get_data_benchmark()
databases/CH/Benchmarks/benchmark_targets.xls

get_data_mobility()
databases/CH/Benchmarks/mobility.xls

get_default_weather()
weather/Zug-2010.epw path to database of archetypes file Archetypes_properties.xlsx

get_demand_measured_file(*building_name*)
scenario/outputs/data/demand/{building_name}.csv

get_demand_measured_folder()
scenario/outputs/data/demand

get_demand_plots_file(*building_name*)
scenario/outputs/plots/timeseries/{building_name}.pdf

get_demand_plots_folder()
scenario/outputs/plots/timeseries

get_demand_results_file(*building_name*)
scenario/outputs/data/demand/{building_name}.csv

get_demand_results_folder()
scenario/outputs/data/demand

get_district()
scenario/inputs/building-geometry/district.shp

get_envelope_systems()
databases/CH/Systems/emission_systems.csv

get_geothermal_potential()
scenario/outputs/data/potentials/geothermal.csv

get_heatmaps_demand_folder()
scenario/outputs/plots/heatmaps

get_heatmaps_emission_folder()
scenario/outputs/plots/heatmaps

get_lca_embodied()
scenario/outputs/data/emissions/Total_LCA_embodied.csv

get_lca_emissions_results_folder()
scenario/outputs/data/emissions

get_lca_mobility()
scenario/outputs/data/emissions/Total_LCA_mobility.csv

get_lca_operation()
scenario/outputs/data/emissions/Total_LCA_operation.csv

get_life_cycle_inventory_supply_systems()
databases/CH/Systems/supply_systems.csv

get_measurements()
scenario/inputs/ Operation pattern for disconnected buildings

get_optimization_clustering_folder()
scenario/outputs/data/optimization/clustering Clustering results for disconnected buildings

get_optimization_disconnected_folder()
scenario/outputs/data/optimization/disconnected Operation pattern for disconnected buildings

get_optimization_disconnected_result_file(*building_name*)
scenario/outputs/data/optimization/disconnected/DiscOp_${building_name}_result.csv

get_optimization_master_results_folder()
scenario/outputs/data/optimization/master Master checkpoints

get_optimization_network_layout_folder()
scenario/outputs/data/optimization/network/layout Network layout files

get_optimization_network_layout_pipes_file()
scenario/outputs/data/optimization/network/layout/PipesData_DH.csv Network layout files for pipes of district heat networks

get_optimization_network_results_folder()
scenario/outputs/data/optimization/network Network summary results

get_optimization_network_totals_folder()
scenario/outputs/data/optimization/network/totals Total files (inputs to substation + network in master)

get_optimization_plots_folder()
scenario/outputs/plots/graphs/Benchmark_scenarios.pdf

get_optimization_results_folder()
scenario/outputs/data/optimization

get_optimization_slave_results_folder()
scenario/outputs/data/optimization/slave Slave results folder (storage + operation pattern)

get_optimization_substations_folder()
scenario/outputs/data/optimization/substations Substation results for disconnected buildings

get_optimization_substations_results_file(*building_name*)
scenario/outputs/data/optimization/substations/${building_name}_result.csv

get_optimization_substations_total_file(*genome*)
scenario/outputs/data/optimization/substations/Total_${genome}.csv

get_potentials_results_folder()
scenario/outputs/data/potentials

get_potentials_solar_folder()
scenario/outputs/data/potentials/solar Contains raw solar files

get_radiation()
scenario/outputs/data/solar-radiation/radiation.csv

get_sensitivity_output(*method*, *samples*)
scenario/outputs/data/sensitivity-analysis/sensitivity_${METHOD}_${SAMPLES}.xls

get_sensitivity_plots_file(*parameter*)
scenario/outputs/plots/sensitivity/${PARAMETER}.pdf

get_sewage_heat_potential()

get_solar_radiation_folder()
scenario/outputs/data/solar-radiation

get_surface_properties()
scenario/outputs/data/solar-radiation/properties_surfaces.csv

get_technical_emission_systems()
databases/CH/Systems/emission_systems.csv

get_temporary_file(*filename*)
Returns the path to a file in the temporary folder with the name *filename*

get_temporary_folder()
Temporary folder as returned by *tempfile*.

get_terrain()
scenario/inputs/topography/terrain.tif

get_timeseries_plots_file(*building_name*)
scenario/outputs/plots/timeseries/{building_name}.html

get_total_demand()
scenario/outputs/data/demand/Total_demand.csv

get_uncertainty_db()
databases/CH/Uncertainty/uncertainty_distributions.xls

get_weather(*name*)
weather/{name}.epw

get_weather_names()
Return a list of all installed epw files in the system

Module contents

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

[Kämpf2009] Kämpf, Jérôme Henri On the modelling and optimisation of urban energy fluxes http://dx.doi.org/10.5075/epfl-thesis-4548

[DIN-16798-7] Energieeffizienz von Gebäuden - Teil 7: Modul M5-1, M 5-5, M 5-6, M 5-8 – Berechnungsmethoden zur Bestimmung der Luftvolumenströme in Gebäuden inklusive Infiltration; Deutsche Fassung prEN 16798-7:2014

[ISO-9972] Wärmetechnisches Verhalten von Gebäuden – Bestimmung der Luftdurchlässigkeit von Gebäuden – Differenzdruckverfahren (ISO 9972:2015); Deutsche Fassung EN ISO 9972:2015

U

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W