# Get Started with CEA

1.1 Install and set up

1.2 Download a test case study

1.3 Begin with a tutorial

1.4 Cite us

# Tutorials

2.1 Essential

2.2 Advanced

2.3 Developers

2.4 Metadata and Documentation

# Being Agile with CEA

3.1 Roles and Responsibilities

3.2 User Personas

3.3 User Stories

3.4 Activities

3.5 Communication channels

# Contributing

4.1 Step 1. Create a User Story or report a Bug

4.2 Step 2. Create a Github branch for your User story or Bug

4.3 Step 3. Get acquainted with variable names

4.4 Step 4. Create your contribution

4.5 Step 5. Test

4.6 Step 6. Create a Pull request

4.7 Step 7. Claim your CEA T-shirt!

4.8 Additional steps

# Known issues

5.1 Report a new issue

# Glossary

6.1 Input

6.2 Output

# Legal
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>License</td>
<td>65</td>
</tr>
<tr>
<td>7.2</td>
<td>Disclaimer</td>
<td>66</td>
</tr>
<tr>
<td>8</td>
<td>Other stuff</td>
<td>67</td>
</tr>
<tr>
<td>8.1</td>
<td>The Configuration File</td>
<td>67</td>
</tr>
<tr>
<td>8.2</td>
<td>Configuration File Details</td>
<td>68</td>
</tr>
<tr>
<td>8.3</td>
<td>How to set up the Jenkins server on a new PC</td>
<td>70</td>
</tr>
<tr>
<td>8.4</td>
<td>How to add a heating/cooling system in CEA</td>
<td>75</td>
</tr>
<tr>
<td>8.5</td>
<td>Data description for thermal_network_matrix.py</td>
<td>77</td>
</tr>
<tr>
<td>8.6</td>
<td>Description of DataFrames and Lists written to csv by the</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>thermal_network_matrix.py file</td>
<td></td>
</tr>
<tr>
<td>8.7</td>
<td>User Interfaces</td>
<td>88</td>
</tr>
<tr>
<td>8.8</td>
<td>Architecture</td>
<td>90</td>
</tr>
<tr>
<td>9</td>
<td>API reference</td>
<td>93</td>
</tr>
<tr>
<td>9.1</td>
<td>cea package</td>
<td>93</td>
</tr>
<tr>
<td>10</td>
<td>Indices and tables</td>
<td>345</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>347</td>
</tr>
<tr>
<td></td>
<td>Python Module Index</td>
<td>349</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>353</td>
</tr>
</tbody>
</table>
City Energy Analyst (CEA) is an open-source software for the analysis of energy systems in cities. CEA helps you to analyse the effects of building retrofits, land-use planning, district heating and cooling and renewable energy on the future costs, emissions and energy consumption of neighbourhoods and districts. In addition CEA helps you to find the optimal location, size and operation of energy generation and distribution technologies for a neighbourhood or a district of your choice.

For the moment, CEA supports neighbourhoods and districts in Temperate (e.g., Switzerland) and Tropical climates (e.g., Singapore).

Learn more..
Get Started with CEA

This guide should get you started with the City Energy Analyst (CEA) 2.29.0.

1.1 Install and set up

CEA can be installed in Windows, Ubuntu and on the computer cluster of the ETH Zurich Euler. The latter is only available for students and faculty of the ETH Zurich.

• installation-on-windows.
• installation-guide-mac.
• installation-guide-ubuntu.
• installation-on-euler.

1.2 Download a test case study

Just go to this link and unzip the case study where necessary.

You can also use the command `cea extract-reference-case --destination C:\Your\Favorite\Folder`. After completion, that folder will contain a subfolder `cea-reference-case` which contains the baseline scenario of the test case study.

1.3 Begin with a tutorial

Check the tutorial `cea-workflow-guide` showing a step-by-step guide on how to use the full stack of tools in CEA. For a complete list of tutorial check out the section Tutorials.
1.4 Cite us

2.1 Essential

These are a collection of tutorials we consider essential for new users. We suggest to take them in order for a better learning experience.

1. How to \textit{Install and set up} CEA?
2. \texttt{script-data-flow}
3. How are databases classified in CEA?
4. What are the input databases of CEA?
5. What are the default databases of CEA?
6. How to edit the input databases of CEA?
7. \texttt{config-file-guide}
8. \texttt{new-project-guide}
9. \texttt{cea-workflow-guide}
10. How to create your own input geometry?
11. \texttt{how-are-schedules-defined}

2.2 Advanced

These are a collection of tutorials related to the specific use of one or more of the tools of CEA.

1. How does the Urban Solar Radiation tool work?
2. How does the Dynamic Demand Forecast feature work?
3. How does the Renewable Energy Assessment tool work?
4. How does the Life Cycle Assessment tool work?
5. How does the 2000-Watt Benchmarking tool work?
6. How to study building retrofits in the CEA?
7. How does the Sensitivity Analysis tool work?
8. how-to-prepare-inputs-for-network-simulation
9. how-to-run-thermal-network-optimization
10. how-to-run-thermal-electrical-grid-planning
11. how-to-run-MPC-building
12. how-to-run-MPC-district
13. how-to-run-CEA-optimization

2.3 Developers

These are a collection of tutorials we consider essential for any new developer of CEA. We suggest to take them in order for a better learning experience.

1. Contributing
2. User Stories
3. how-to-report-bugs
4. how-to-use-github
5. how-to-name-variables
6. how-to-add-a-new-script-to-the-cea
7. how-to-test-the-cea
8. how-to-set-up-nsis
9. how-to-create-a-new-release
10. how-to-publish-cea

2.4 Metadata and Documentation

These are a collection of tutorials detailing the acquisition of scenario specific metadata and generalized CEA schema as well as how to use the automated documentation tools.

1. how-trace-inputlocator-works
2. how-to-document-cea
3.1 Roles and Responsibilities

This is a guide of roles and responsibilities in CEA. We use this to more formally coordinate our work. To check who is currently holding each position, we invite you to check www.cityenergyanalyst.com/people for more information.

3.1.1 Product Sponsor

The Project Sponsor is the person that has overall responsibility and accountability. His/her main responsibilities are:

1. To champion the project based on whether the work fits our research needs and overall strategy.
2. To find and secure the budget and ensure high-level organizational risks are mitigated.
3. The Sponsor empowers the product owner to act for him/her on a more tactical basis.
4. Participate in Review events at least.

3.1.2 Product Owner

The Product Owner is the voice of the customer and performs all tactical operations. His/her main responsibilities are:

1. Acting as a conduit/facilitator for communication between the team and the outside world.
2. Defining and elaborating user stories, prioritizing the backlog and accepting work when the team completes a story.
3. The Product Owner breaks down the overall vision into feasible and tactical pieces of work for the development team to understand and create.
4. Participate in Planning and Review events at least.
3.1.3 Scrum Master

The Scrum Master is the servant leader and maintainer of agile practices. His/her main responsibilities are:

1. Serve as a servant leader, keeping the team out of distractions and empowering them to do their work with ease.
2. Maintain all agile practices (including Activities) and communication protocols between the Development team, the Product Owner and the Product Sponsor.
3. Support the Product Owner on setting priorities and grooming the backlog.
4. Nurturing new and existing members of the development team.

3.1.4 Development team

The Development team is the group of individuals who build CEA. His/her main responsibilities are:

1. Design, analyze, develop, test and document new features in CEA.
2. Be Self-organized and self-directed.
3. Participate in all agile practices led by the Scrum Master.

3.2 User Personas

This is a guide of User Personas in CEA. These are potential end-users of CEA. We describe the goals and priorities of these user personas hereafter. We use this information to build User Stories / Report bugs / Develop new features. It is of utmost importance for every developer to know this by heart.

3.2.1 Researcher

The priorities of this User Persona are:

1. To find an answer to a unique research question/hypothesis
2. To build upon the state-of-the-art, saving time in coding newly established algorithms.
3. To conduct experiments quickly.
4. To generate replicable and transparent results.

3.2.2 Student

The priorities of this User Persona are:

1. To understand the trade-offs of planning cities for energy efficiency.
2. To build intuition about the mechanics of different analysis and modeling techniques.
3. To generate replicable and transparent results.
3.2.3 Practitioner

The priorities of this User Persona are:

1. To analyze real case studies quickly.
2. To generate great visualizations which aid communication to stakeholders.
3. To generate standard, replicable and transparent results.

3.3 User Stories

User stories define WHAT (need), WHY (reason) and for WHOM (User persona) we aim to solve a new bug or implement a new feature. This helps our Project Owner to better prioritize issues around CEA.

We have a standard yet simple way to define these user stories. This guide walks you through defining your first user story.

3.3.1 Step 1. define a user persona

At CEA we differentiate four (4) potential USERS for creating an user story. These are key groups of users we want CEA to serve and are defined as user personas. The first step in creating a new story is to select a User Persona from the following list:

1. Researcher: A member of the CEA research team or network of contributors.
2. Student or Practitioner: An active user of CEA.

For more information about the goals and priorities of the different user personas check: User Personas

3.3.2 Step 2. define its needs

Think about the NEED this user persona has. Is it about a problem/bug or rather about a new feature you want to be implemented? It is important that you think about the NEED from the point of view of the user persona.

3.3.3 Step 3. define a reason

Now think about why the user persona needs that.

3.3.4 Step 4. put the story together

Now put it all together using the following template:

As a **USER PERSONA** I want to **NEED** so I can **REASON**

Here is an example:

**USER PERSONA** Researcher

**NEED** know how to define user stories

**REASON** add new bugs and features to the pipeline of CEA
The result will be the title of your user story:

**As a** Researcher **I want to** know how to define user stories **so I** can add new bugs and features to the pipeline of CEA.

One more example:

**USER PERSONA**  Student

**NEED**  understand how the dynamic tool works

**REASON**  use CEA more effectively

The result will be the title of your user story:

**As a** Student **I want to** understand how the dynamic tool works **so I** can use CEA more effectively.

### 3.3.5 Step 5. submit a new user story

Now it is time to submit a new user story in CEA. For this:

1. Go to the CEA repository in Github.
2. Click **New Issue**
3. in Title, add the name of the user story
4. Finally, in description, give a more detailed description of the problem.
5. In the description you can directly connect to other user stories using # or connect to other people using @ in the text.

### 3.4 Activities

This is a guide of Activities carried out during the development of CEA. This activities are part of our concept of operation under responsibility of our the Scrum Master

#### 3.4.1 Planning event

Duration: 1 - 4 hours. Frequency: After every block of work (sprint) (max of 1 month) Scope: Define what user stories should go in the next block of work (sprint). These stories are attached to a new time-bounded milestone Attendees: Scrum Master, Product Owner, Product Sponsor, and Development Team.

#### 3.4.2 Daily stand-up

Duration: 15 min. Frequency: daily Scope: Provide early support and mentoring Attendees: Scrum Master and Development Team.

#### 3.4.3 Review event

3.4.4 Retrospective

Duration: 1 hour. Frequency: After every review event Scope: Discuss what was wrong and set next steps to follow. Attendees: Scrum Master and Development Team.

3.5 Communication channels

This is a guide of Communication channels in CEA. These are maintained by our Scrum Master.

3.5.1 CEA website

Host: squarespace.com Credentials: Ask the Product Owner Jimeno A. Fonseca Administrator: Product Owner Access granted to: Scrum Master, Product Owner, Development team

3.5.2 CEA e-mail

Host: arch.ethz.ch Credentials: Ask the Product Owner Jimeno A. Fonseca Administrator: Product Owner Access granted to: Scrum Master, Product Owner

3.5.3 CEA messenger

Host: ceadev.slack.com Credential: Ask the Scrum Master Daren Thomas Administrator: Scrum Master Access granted to: Scrum Master, Product Owner, Development team

3.5.4 CEA newsletter

Host: mailchimp.com Credentials: Ask the Product Owner Jimeno A. Fonseca Administrator: Product Owner Access granted to: Scrum Master, Product Owner

3.5.5 CEA documents

Host: gmail.com Credentials: Ask the Scrum Master Daren Thomas Administrator: Scrum Master Access granted to: Scrum Master, Product Owner, Development team
This is a guide on how to expand CEA and become an official contributor!

The main steps you need to take are:

1. Create an User Story or a Bug in Github.
2. Create branch and learn how to use Github.
3. get acquainted with variable naming.
4. create your contribution.
5. test your creation.
6. Create a pull request on Github.
7. Claim your CEA T-shirt!

### 4.1 Step 1. Create a User Story or report a Bug

If you decide to contribute to CEA is because you might have already an problem/feature in mind to solve/implement.

New features are called **User Stories**.

Problems are called **Bugs**

Check this guide for more details on how to create **User Stories**.

Check this guide for more details on how-to-report-bugs.

### 4.2 Step 2. Create a Github branch for your User story or Bug

At CEA we create something called Branches for each User Story or Bug we would like to complete.

Check this guide for more details on how to do it: how-to-use-github.
4.3 Step 3. Get acquainted with variable names

If you have not done it yet, take some time to get acquainted with variable names in CEA. This would make easier for you to understand and develop consistent code.

Check this guide for more details: how-to-name-variables.

4.4 Step 4. Create your contribution

Go ahead and start coding! For this make sure to use one of our template scripts and follow the documentation guide. This could help to maintain an homogenous structure, and help us to give you credits for your work.

Check this guide for more details: how-to-add-a-new-script-to-the-cea.

Check this guide for adding documentation how-to-document-cea.

4.5 Step 5. Test

Now just test if your creation is not buggy.

Check this guide for more details on how to do it: how-to-test-the-cea.

And that is all! Many thanks for this!

4.6 Step 6. Create a Pull request

Now it is time to ask other developer of CEA to review your code and put it inside the core of CEA. For this we will be creating a Pull Request in Github.

Check this guide for more details on how to do it: how-to-use-github.

4.7 Step 7. Claim your CEA T-shirt!

What happens after that? We will check the code, and if all is correct we will proceed to merge your User Story/ Github Branch into the CEA main core.

AND THAT IS ALL! if your work has been merged, give yourself an applause. You have just made part of the growing network of developers of CEA.

Your are entitled to claim a CEA T-shirt after this to cea@arch.ethz.ch

4.8 Additional steps

There are two more additional steps that are undertaken only when the CEA team decides to make a new release and publish!

1. how-to-create-a-new-release.

2. how-to-publish-cea.
CEA uses Github Issues to document new ideas as well as issues and bugs.

The table below contains a number of the most common issues:

<table>
<thead>
<tr>
<th>Issue #</th>
<th>Regarding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1577</td>
<td>ArcGIS &lt; 10.6</td>
<td>Internet Explorer Script Error: An error has occurred in the script of this page. Do you want to continue running scripts on this page?</td>
</tr>
<tr>
<td>1704</td>
<td>Installation</td>
<td>Error during installation via command line: Import Error: no module named arcpy</td>
</tr>
<tr>
<td>1708</td>
<td>Fiona/GDAL</td>
<td>Error after installation via command line ImportError: DLL load failed: The specified module could not be found.</td>
</tr>
<tr>
<td>1709</td>
<td>daysim_main.py</td>
<td>Error when running daysim_main.py: .wea weather file cannot be found.</td>
</tr>
</tbody>
</table>

If you cannot find your issue here, please check the complete list of known issues.

5.1 Report a new issue

For any problems please post a new issue here.

Please read the how-to-report-bugs guide and review the open issues before posting.

We have a turn-over time of a couple of days.

We appreciate your contribution!
This glossary contains all the written input and output variables used by CEA. These variables are stored in databases, themed by the type of information they contain. There are two main types of databases in CEA: input and output. This glossary is organised through the cea’s inputlocator method, which is used to retrieve the information within each file.

6.1 Input

Inputlocator methods:

6.1.1 get_archetypes_properties

The following file is used by scripts: ['data-helper', 'demand']
Table 1: databases/ch/archetypes/construction_properties.xlsx:ARCHITECTURE

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Es</td>
<td>Fraction of gross floor area with electrical demands. - Unit: [m2/m2]</td>
</tr>
<tr>
<td>Ns</td>
<td>Fraction of net gross floor area. - Unit: [m2/m2]</td>
</tr>
<tr>
<td>building_use</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>standard</td>
<td>Letter representing whereas the field represent construction properties of a building as newly constructed (C) or renovated (R) - Unit: [-]</td>
</tr>
<tr>
<td>type_cons</td>
<td>Type of construction. It relates to the contents of the default database of Envelope Properties: construction - Unit: [code]</td>
</tr>
<tr>
<td>type_leak</td>
<td>Leakage level. It relates to the contents of the default database of Envelope Properties: leakage - Unit: [code]</td>
</tr>
<tr>
<td>type_roof</td>
<td>Roof construction type (relates to values in Default Database Construction Properties) - Unit: [-]</td>
</tr>
<tr>
<td>type_wall</td>
<td>Wall construction type (relates to values in Default Database Construction Properties) - Unit: [m2/m2]</td>
</tr>
<tr>
<td>type_win</td>
<td>Window type (relates to values in Default Database Construction Properties) - Unit: [m2/m2]</td>
</tr>
<tr>
<td>void_deck</td>
<td>Number of floors (from the ground up) with an open envelope (default = 0) - Unit: [-]</td>
</tr>
<tr>
<td>wwr_east</td>
<td>Window to wall ratio in in facades facing east - Unit: [m2/m2]</td>
</tr>
<tr>
<td>wwr_north</td>
<td>Window to wall ratio in in facades facing north - Unit: [m2/m2]</td>
</tr>
<tr>
<td>wwr_south</td>
<td>Window to wall ratio in in facades facing south - Unit: [m2/m2]</td>
</tr>
<tr>
<td>wwr_west</td>
<td>Window to wall ratio in in facades facing west - Unit: [m2/m2]</td>
</tr>
<tr>
<td>year_end</td>
<td>Upper limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
<tr>
<td>year_start</td>
<td>Lower limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
</tbody>
</table>

Table 2: databases/ch/archetypes/construction_properties.xlsx:HVAC

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>building_use</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>standard</td>
<td>Letter representing whereas the field represent construction properties of a building as newly constructed (C) or renovated (R) - Unit: [-]</td>
</tr>
<tr>
<td>type_cs</td>
<td>Type of cooling supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_ctrl</td>
<td>Type of heating and cooling control systems (relates to values in Default Database HVAC Properties) - Unit: [code]</td>
</tr>
<tr>
<td>type_dhw</td>
<td>Type of hot water supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_hs</td>
<td>Type of heating supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_vent</td>
<td>Type of ventilation strategy (relates to values in Default Database HVAC Properties) - Unit: [code]</td>
</tr>
<tr>
<td>year_end</td>
<td>Upper limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
<tr>
<td>year_start</td>
<td>Lower limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
</tbody>
</table>

Table 3: databases/ch/archetypes/construction_properties.xlsx:INDOOR_COMFORT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Unique code for the material of the pipe. - Unit: [-]</td>
</tr>
<tr>
<td>RH_max_pc</td>
<td>Upper bound of relative humidity - Unit: [%]</td>
</tr>
<tr>
<td>RH_min_pc</td>
<td>Lower bound of relative humidity - Unit: [%]</td>
</tr>
<tr>
<td>Tcs_set_C</td>
<td>Setpoint temperature for cooling system - Unit: [C]</td>
</tr>
<tr>
<td>Tcs_setb_C</td>
<td>Setback point of temperature for cooling system - Unit: [C]</td>
</tr>
<tr>
<td>Ths_set_C</td>
<td>Setpoint temperature for heating system - Unit: [C]</td>
</tr>
<tr>
<td>Ths_setb_C</td>
<td>Setback point of temperature for heating system - Unit: [C]</td>
</tr>
<tr>
<td>Ve_lpspax</td>
<td>Indoor quality requirements of indoor ventilation per person - Unit: [l/s]</td>
</tr>
</tbody>
</table>
Table 4: `databases/ch/archetypes/construction_properties.xlsx:INTERNAL_LOADS`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Unique code for the material of the pipe. - Unit: [-]</td>
</tr>
<tr>
<td>Ea_Wm2</td>
<td>Peak specific electrical load due to computers and devices - Unit: [W/m²]</td>
</tr>
<tr>
<td>Ed_Wm2</td>
<td>Peak specific electrical load due to servers/data centres - Unit: [W/m²]</td>
</tr>
<tr>
<td>El_Wm2</td>
<td>Peak specific electrical load due to artificial lighting - Unit: [W/m²]</td>
</tr>
<tr>
<td>Epro_Wm2</td>
<td>Peak specific electrical load due to industrial processes - Unit: [W/m²]</td>
</tr>
<tr>
<td>Qcpro_Wm2</td>
<td>Peak specific process cooling load - Unit: [W/m²]</td>
</tr>
<tr>
<td>Qcre_Wm2</td>
<td>Peak specific cooling load due to refrigeration (cooling rooms) - Unit: [W/m²]</td>
</tr>
<tr>
<td>Qhpro_Wm2</td>
<td>Peak specific process heating load - Unit: [W/m²]</td>
</tr>
<tr>
<td>Qs_Wpax</td>
<td>Peak sensible heat load of people - Unit: [W/pax]</td>
</tr>
<tr>
<td>Vw_lpdax</td>
<td>Peak specific fresh water consumption (includes cold and hot water) - Unit: [lpd]</td>
</tr>
<tr>
<td>Vww_lpdax</td>
<td>Peak specific daily hot water consumption - Unit: [lpd]</td>
</tr>
<tr>
<td>X_ghpax</td>
<td>Moisture released by occupancy at peak conditions - Unit: [gh/kg/p]</td>
</tr>
</tbody>
</table>

Table 5: `databases/ch/archetypes/construction_properties.xlsx:SUPPLY`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>building_use</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>standard</td>
<td>Letter representing whereas the field represent construction properties of a building as newly constructed (C) or renovated (R) - Unit: [-]</td>
</tr>
<tr>
<td>type_cs</td>
<td>Type of cooling supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_dhw</td>
<td>Type of hot water supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_el</td>
<td>Type of electrical supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_hs</td>
<td>Type of heating supply system - Unit: [code]</td>
</tr>
<tr>
<td>year_end</td>
<td>Upper limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
<tr>
<td>year_start</td>
<td>Lower limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
</tbody>
</table>

6.1.2 get_archetypes_schedules

The following file is used by scripts: ['data-helper', 'demand']

Table 6: `databases/ch/archetypes/occupancy_schedules.xlsx:COOLROOM`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m² per person - Unit: [m²/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>
Table 7: databases/ch/archetypes/occupancy_schedules.xlsx:FOODSTORE

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

Table 8: databases/ch/archetypes/occupancy_schedules.xlsx:GYM

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

Table 9: databases/ch/archetypes/occupancy_schedules.xlsx:HOSPITAL

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>
### Table 10: databases/ch/archetypes/occupancy_schedules.xlsx:HOTEL

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 11: databases/ch/archetypes/occupancy_schedules.xlsx:INDUSTRIAL

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 12: databases/ch/archetypes/occupancy_schedules.xlsx:LAB

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>
### Table 13: databases/ch/archetypes/occupancy_schedules.xlsx:LIBRARY

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 14: databases/ch/archetypes/occupancy_schedules.xlsx:MULTI_RES

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 15: databases/ch/archetypes/occupancy_schedules.xlsx:MUSEUM

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>
### Table 16: databases/ch/archetypes/occupancy_schedules.xlsx:OFFICE

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m² per person - Unit: [m²/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 17: databases/ch/archetypes/occupancy_schedules.xlsx:PARKING

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m² per person - Unit: [m²/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 18: databases/ch/archetypes/occupancy_schedules.xlsx:RESTAURANT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m² per person - Unit: [m²/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>
### Table 19: databases/ch/archetypes/occupancy_schedules.xlsx:RETAIL

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 20: databases/ch/archetypes/occupancy_schedules.xlsx:SCHOOL

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 21: databases/ch/archetypes/occupancy_schedules.xlsx:SERVERROOM

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m2 per person - Unit: [m2/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>
### Table 22: `databases/ch/archetypes/occupancy_schedules.xlsx:SINGLE_RES`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m² per person - Unit: [m²/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 23: `databases/ch/archetypes/occupancy_schedules.xlsx:SWIMMING`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturday_1</td>
<td>Probability of maximum occupancy per hour on Saturday - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Saturday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_1</td>
<td>Probability of maximum occupancy per hour on Sunday - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Sunday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_1</td>
<td>Probability of maximum occupancy per hour in a weekday - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_2</td>
<td>Probability of use of lighting and appliances (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>Weekday_3</td>
<td>Probability of domestic hot water consumption (daily) for each hour - Unit: [p/p]</td>
</tr>
<tr>
<td>density</td>
<td>m² per person - Unit: [m²/p]</td>
</tr>
<tr>
<td>month</td>
<td>Probability of use for the month - Unit: [p/p]</td>
</tr>
</tbody>
</table>

### Table 24: `databases/ch/archetypes/system_controls.xlsx:heating_cooling`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cooling-season-end</td>
<td>Last day of the cooling season - Unit: [-]</td>
</tr>
<tr>
<td>cooling-season-start</td>
<td>Day on which the cooling season starts - Unit: [-]</td>
</tr>
<tr>
<td>has-cooling-season</td>
<td>Defines whether the scenario has a cooling season. - Unit: [-]</td>
</tr>
<tr>
<td>has-heating-season</td>
<td>Defines whether the scenario has a heating season. - Unit: [-]</td>
</tr>
<tr>
<td>heating-season-end</td>
<td>Last day of the heating season - Unit: [-]</td>
</tr>
<tr>
<td>heating-season-start</td>
<td>Day on which the heating season starts - Unit: [-]</td>
</tr>
</tbody>
</table>

### 6.1.3 `get_archetypes_system_controls`

The following file is used by scripts: ['demand']

### Table 24: `databases/ch/archetypes/system_controls.xlsx:heating_cooling`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cooling-season-end</td>
<td>Last day of the cooling season - Unit: [-]</td>
</tr>
<tr>
<td>cooling-season-start</td>
<td>Day on which the cooling season starts - Unit: [-]</td>
</tr>
<tr>
<td>has-cooling-season</td>
<td>Defines whether the scenario has a cooling season. - Unit: [-]</td>
</tr>
<tr>
<td>has-heating-season</td>
<td>Defines whether the scenario has a heating season. - Unit: [-]</td>
</tr>
<tr>
<td>heating-season-end</td>
<td>Last day of the heating season - Unit: [-]</td>
</tr>
<tr>
<td>heating-season-start</td>
<td>Day on which the heating season starts - Unit: [-]</td>
</tr>
</tbody>
</table>

### 6.1.4 `get_building_age`

The following file is used by scripts: ['data-helper', 'emissions', 'demand']
Table 25: `inputs/building-properties/age.dbf`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVAC</td>
<td>Year of last retrofit of HVAC systems (0 if none) - Unit: [-]</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>basement</td>
<td>Year of last retrofit of basement (0 if none) - Unit: [-]</td>
</tr>
<tr>
<td>built</td>
<td>Construction year - Unit: [-]</td>
</tr>
<tr>
<td>envelope</td>
<td>Year of last retrofit of building facades (0 if none) - Unit: [-]</td>
</tr>
<tr>
<td>partitions</td>
<td>Year of last retrofit of internal wall partitions(0 if none) - Unit: [-]</td>
</tr>
<tr>
<td>roof</td>
<td>Year of last retrofit of roof (0 if none) - Unit: [-]</td>
</tr>
<tr>
<td>windows</td>
<td>Year of last retrofit of windows (0 if none) - Unit: [-]</td>
</tr>
</tbody>
</table>

6.1.5 get_building_occupancy

The following file is used by scripts: ['data-helper', 'emissions', 'demand']

Table 26: `inputs/building-properties/occupancy.dbf`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COOLROOM</td>
<td>Refrigeration rooms - Unit: [m2]</td>
</tr>
<tr>
<td>FOODSTORE</td>
<td>Food stores - Unit: [m2]</td>
</tr>
<tr>
<td>GYM</td>
<td>Gymnasiums - Unit: [m2]</td>
</tr>
<tr>
<td>HOSPITAL</td>
<td>Hospitals - Unit: [m2]</td>
</tr>
<tr>
<td>HOTEL</td>
<td>Hotels - Unit: [m2]</td>
</tr>
<tr>
<td>INDUSTRIAL</td>
<td>Light industry - Unit: [m2]</td>
</tr>
<tr>
<td>LIBRARY</td>
<td>Libraries - Unit: [m2]</td>
</tr>
<tr>
<td>MULTI_RES</td>
<td>Residential (multiple dwellings) - Unit: [m2]</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>OFFICE</td>
<td>Offices - Unit: [m2]</td>
</tr>
<tr>
<td>PARKING</td>
<td>Parking - Unit: [m2]</td>
</tr>
<tr>
<td>RESTAURANT</td>
<td>Restaurants - Unit: [m2]</td>
</tr>
<tr>
<td>RETAIL</td>
<td>Retail - Unit: [m2]</td>
</tr>
<tr>
<td>SCHOOL</td>
<td>Schools - Unit: [m2]</td>
</tr>
<tr>
<td>SERVERROOM</td>
<td>Data center - Unit: [m2]</td>
</tr>
<tr>
<td>SINGLE_RES</td>
<td>Residential (single dwellings) - Unit: [m2]</td>
</tr>
<tr>
<td>SWIMMING</td>
<td>Swimming halls - Unit: [m2]</td>
</tr>
</tbody>
</table>

6.1.6 get_database_air_conditioning_systems

The following file is used by scripts: ['demand']

6.1.7 get_envelope_systems

The following file is used by scripts: ['radiation', 'demand']
### Table 27: databases/ch/systems/envelope_systems.xls:CONSTRUCTION

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cm_Af</td>
<td>Internal heat capacity per unit of air conditioned area. Defined according to ISO 13790. - Unit: [J/Km2]</td>
</tr>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
</tbody>
</table>

### Table 28: databases/ch/systems/envelope_systems.xls:LEAKAGE

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>n50</td>
<td>Air exchanges per hour at a pressure of 50 Pa. - Unit: [1/h]</td>
</tr>
</tbody>
</table>

### Table 29: databases/ch/systems/envelope_systems.xls:ROOF

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>U_roof</td>
<td>Thermal transmittance of windows including linear losses (+10%). Defined according to ISO 13790. - Unit: [-]</td>
</tr>
<tr>
<td>a_roof</td>
<td>Solar absorption coefficient. Defined according to ISO 13790. - Unit: [-]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>e_roof</td>
<td>Emissivity of external surface. Defined according to ISO 13790. - Unit: [-]</td>
</tr>
<tr>
<td>r_roof</td>
<td>Reflectance in the Red spectrum. Defined according Radiance. (long-wave) - Unit: [-]</td>
</tr>
</tbody>
</table>

### Table 30: databases/ch/systems/envelope_systems.xls:SHADING

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>rf_sh</td>
<td>Shading coefficient when shading device is active. Defined according to ISO 13790. - Unit: [-]</td>
</tr>
</tbody>
</table>

### Table 31: databases/ch/systems/envelope_systems.xls:WALL

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>U_base</td>
<td>Thermal transmittance of basement including linear losses (+10%). Defined according to ISO 13790. - Unit: [-]</td>
</tr>
<tr>
<td>U_wall</td>
<td>Thermal transmittance of windows including linear losses (+10%). Defined according to ISO 13790. - Unit: [-]</td>
</tr>
<tr>
<td>a_wall</td>
<td>Solar absorption coefficient. Defined according to ISO 13790. - Unit: [-]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>e_wall</td>
<td>Emissivity of external surface. Defined according to ISO 13790. - Unit: [-]</td>
</tr>
<tr>
<td>r_wall</td>
<td>Reflectance in the Red spectrum. Defined according Radiance. (long-wave) - Unit: [-]</td>
</tr>
</tbody>
</table>
Table 32: databases/ch/systems/envelope_systems.xls:WINDOW

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>G_win</td>
<td>Solar heat gain coefficient. Defined according to ISO 13790. - Unit: [-]</td>
</tr>
<tr>
<td>U_win</td>
<td>Thermal transmittance of windows including linear losses (+10%). Defined according to ISO 13790. - Unit: [-]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>e_win</td>
<td>Emissivity of external surface. Defined according to ISO 13790. - Unit: [-]</td>
</tr>
</tbody>
</table>

6.1.8 get_life_cycle_inventory_building_systems

The following file is used by scripts: ['emissions']

Table 33: databases/sg/lifecycle/lca_buildings.xlsx:EMBODIED_EMISSIONS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excavation</td>
<td>Typical embodied emissions for site excavation. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>Floor_g</td>
<td>Typical embodied emissions of the ground floor. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>Floor_int</td>
<td>Typical embodied emissions of the interior floor. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>Roof</td>
<td>Typical embodied emissions of the roof. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>Services</td>
<td>Typical embodied emissions of the building services. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>Wall_ext_ag</td>
<td>Typical embodied emissions of the exterior above ground walls. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>Wall_ext_bg</td>
<td>Typical embodied emissions of the exterior below ground walls. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>Wall_int_nosup</td>
<td>Typical embodied emissions of the interior above ground walls. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>Wall_int_sup</td>
<td>Typical embodied emissions of the interior below ground walls. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>Win_ext</td>
<td>Typical embodied emissions of the external glazing. - Unit: [kg CO2-eq/m2]</td>
</tr>
<tr>
<td>building_use</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>standard</td>
<td>Letter representing whereas the field represent construction properties of a building as newly constructed (C) or renovated (R) - Unit: [-]</td>
</tr>
<tr>
<td>year_end</td>
<td>Upper limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
<tr>
<td>year_start</td>
<td>Lower limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
</tbody>
</table>
Table 34: databases/sg/lifecycle/lca_buildings.xlsx:EMBODIED_ENERGY

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excavation</td>
<td>Typical embodied energy for site excavation. - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>Floor_g</td>
<td>Typical embodied energy of the ground floor. - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>Floor_int</td>
<td>Typical embodied energy of the interior floor. - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>Roof</td>
<td>Typical embodied energy of the roof. - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>Services</td>
<td>Typical embodied energy of the building services. - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>Wall_ext_ag</td>
<td>Typical embodied energy of the exterior above ground walls. - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>Wall_ext_bg</td>
<td>Typical embodied energy of the exterior below ground walls. - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>Wall_int_no_sup</td>
<td>nan - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>Wall_int_sup</td>
<td>nan - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>Win_ext</td>
<td>Typical embodied energy of the external glazing. - Unit: [MJ oil-eq/m2]</td>
</tr>
<tr>
<td>building_use</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>standard</td>
<td>Letter representing whereas the field represent construction properties of a building as newly constructed (C) or renovated (R) - Unit: [-]</td>
</tr>
<tr>
<td>year_end</td>
<td>Upper limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
<tr>
<td>year_start</td>
<td>Lower limit of year interval where the building properties apply - Unit: [yr]</td>
</tr>
</tbody>
</table>

6.1.9 get_life_cycle_inventory_supply_systems

The following file is used by scripts: ['demand', 'operation-costs', 'emissions']

Table 35: databases/sg/lifecycle/lca_infrastructure.xlsx:COOLING

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>eff_cs</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>reference</td>
<td>nan - Unit: [-]</td>
</tr>
<tr>
<td>scale_cs</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>source_cs</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

Table 36: databases/sg/lifecycle/lca_infrastructure.xlsx:DHW

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>eff_dhw</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>reference</td>
<td>nan - Unit: [-]</td>
</tr>
<tr>
<td>scale_dhw</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>source_dhw</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.1. Input
Table 37: databases/sg/lifecycle/lca_infrastructure.xlsx:ELECTRICITY

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy. - Unit: [-]</td>
</tr>
<tr>
<td>eff_el</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>reference</td>
<td>nan - Unit: [-]</td>
</tr>
<tr>
<td>scale_el</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>source_el</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

Table 38: databases/sg/lifecycle/lca_infrastructure.xlsx:HEATING

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy. - Unit: [-]</td>
</tr>
<tr>
<td>eff_hs</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>reference</td>
<td>nan - Unit: [-]</td>
</tr>
<tr>
<td>scale_hs</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>source_hs</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

Table 39: databases/sg/lifecycle/lca_infrastructure.xlsx:RESOURCES

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CO2</td>
<td>Refers to the equivalent CO2 required to run the heating or cooling system. - Unit: [kg/kWh]</td>
</tr>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>PEN</td>
<td>Refers to the amount of primary energy needed (PEN) to run the heating or cooling system. - Unit: [kWh/kWh]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy. - Unit: [-]</td>
</tr>
<tr>
<td>costs_kWh</td>
<td>Refers to the financial costs required to run the heating or cooling system. - Unit: [$/kWh]</td>
</tr>
<tr>
<td>reference</td>
<td>nan - Unit: [-]</td>
</tr>
</tbody>
</table>

6.1.10 get_street_network

The following file is used by scripts: ['network-layout']

Table 40: inputs/networks/streets.shp

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>geometry</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.1.11 get_supply_systems

The following file is used by scripts: ['thermal-network', 'photovoltaic', 'photovoltaic-thermal', 'solar-collector']
### Table 41: databases/ch/systems/supply_systems.xls:Absorption_chiller

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a_e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a_g</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e_e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e_g</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>el_W</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>m_cw</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>m_hw</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>r_e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>r_g</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>s_e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>s_g</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>type</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 42: databases/ch/systems/supply_systems.xls:BH

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.1. Input
### Table 43: databases/ch/systems/supply_systems.xls:Boiler

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 44: databases/ch/systems/supply_systems.xls:CCGT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>
### Table 45: databases/ch/systems/supply_systems.xls:CT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 46: databases/ch/systems/supply_systems.xls:Chiller

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>
### Table 47: `databases/ch/systems/supply_systems.xls:FC`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumptions</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 48: `databases/ch/systems/supply_systems.xls:Furnace`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>
### Table 49: databases/ch/systems/supply_systems.xls:HEX

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency</td>
<td>Defines the unit of currency used to create the cost estimations (year specific). E.g. USD-2015. - Unit: [-]</td>
</tr>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a_p</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b_p</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c_p</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d_p</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e_p</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 50: databases/ch/systems/supply_systems.xls:HP

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>
### Table 51: databases/ch/systems/supply_systems.xls:PV

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
<td></td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>PV_Bref</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>PV_a0</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>PV_a1</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>PV_a2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>PV_a3</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>PV_a4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>PV_n</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>PV_noct</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>PV_th</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>misc_losses</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>module_length_m</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>type</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 52: databases/ch/systems/supply_systems.xls:PVT

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
<td></td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>
### Table 53: databases/ch/systems/supply_systems.xls:Piping

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>Diameter_max</td>
<td>Defines the maximum pipe diameter tolerance for the nominal diameter (DN) bin. - Unit: [-]</td>
</tr>
<tr>
<td>Diameter_min</td>
<td>Defines the minimum pipe diameter tolerance for the nominal diameter (DN) bin. - Unit: [-]</td>
</tr>
<tr>
<td>Investment</td>
<td>Typical cost of investment for a given pipe diameter. - Unit: [$/m]</td>
</tr>
<tr>
<td>Unit</td>
<td>Defines the unit of measurement for the diameter values. - Unit: [mm]</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 54: databases/ch/systems/supply_systems.xls:Pricing

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>value</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 55: databases/ch/systems/supply_systems.xls:Pump

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 56: databases/ch/systems/supply_systems.xls:SC

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_eff</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Cp_fluid</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>IAM_d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>aperture_area_ratio</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 56 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c1</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>dP1</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>dP2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>dP3</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>dP4</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>mB0_r</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>mB_max_r</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>mB_min_r</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>module_area_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>module_length_m</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>n0</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>t_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>type</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### Table 57: databases/ch/systems/supply_systems.xls:TES

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Describes the source of the benchmark standards. - Unit: [-]</td>
</tr>
<tr>
<td>IR_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>LT_yr</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>O&amp;M_%</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>a</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>assumption</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>b</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>c</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_max</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>cap_min</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>currency</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>d</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>e</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>unit</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### 6.1.12 get_surroundings_geometry

The following file is used by scripts: ['radiation']
Table 58: inputs/building-geometry/surroundings.shp

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>floors_ag</td>
<td>Number of floors above ground (incl. ground floor) - Unit: [-]</td>
</tr>
<tr>
<td>geometry</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>height_ag</td>
<td>Height above ground (incl. ground floor) - Unit: [m]</td>
</tr>
</tbody>
</table>

6.1.13 get_terrain

The following file is used by scripts: ['radiation']

Table 59: inputs/topography/terrain.tif

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mock_variable</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.1.14 get_thermal_networks

The following file is used by scripts: ['thermal-network']

Table 60: databases/ch/systems/thermal_networks.xls:MATERIAL PROPERTIES

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cp_JkgK</td>
<td>Heat capacity of transmission fluid. - Unit: [J/kgK]</td>
</tr>
<tr>
<td>code</td>
<td>Building use. It relates to the uses stored in the input database of Zone_occupancy - Unit: [-]</td>
</tr>
<tr>
<td>lambda_WmK</td>
<td>Thermal conductivity - Unit: [W/mK]</td>
</tr>
<tr>
<td>material</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>rho_kgm3</td>
<td>Density of transmission fluid. - Unit: [kg/m3]</td>
</tr>
</tbody>
</table>

Table 61: databases/ch/systems/thermal_networks.xls:PIPING CAT-ALOG

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_ext_m</td>
<td>Defines the maximum pipe diameter tolerance for the nominal diameter (DN) bin. - Unit: [m]</td>
</tr>
<tr>
<td>D_ins_m</td>
<td>Defines the pipe insulation diameter for the nominal diameter (DN) bin. - Unit: [m]</td>
</tr>
<tr>
<td>D_int_m</td>
<td>Defines the minimum pipe diameter tolerance for the nominal diameter (DN) bin. - Unit: [m]</td>
</tr>
<tr>
<td>Pipe_DN</td>
<td>Classifies nominal pipe diameters (DN) into typical bins. E.g. DN100 refers to pipes of approx. 100mm in diameter. - Unit: [DN#]</td>
</tr>
<tr>
<td>Vdot_max_m3s</td>
<td>Maximum volume flow rate for the nominal diameter (DN) bin. - Unit: [m3/s]</td>
</tr>
<tr>
<td>Vdot_min_m3s</td>
<td>Minimum volume flow rate for the nominal diameter (DN) bin. - Unit: [m3/s]</td>
</tr>
</tbody>
</table>

6.1.15 get_zone_geometry

The following file is used by scripts: ['photovoltaic', 'photovoltaic-thermal', 'emissions', 'network-layout', 'radiation', 'demand', 'solar-collector']
6.2 Output

Input locator methods:

6.2.1 PVT_metadata_results

The following file is used by scripts: []

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA_m2</td>
<td>Surface area. - Unit: [m2]</td>
</tr>
<tr>
<td>BUILDING</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>B_deg</td>
<td>Tilt angle of the installed solar panels - Unit: [deg]</td>
</tr>
<tr>
<td>CATB</td>
<td>Category according to the tilt angle of the panel - Unit: [-]</td>
</tr>
<tr>
<td>CATGB</td>
<td>Category according to the annual radiation on the panel surface - Unit: [-]</td>
</tr>
<tr>
<td>CAT_teta_z</td>
<td>Category according to the surface azimuth of the panel - Unit: [-]</td>
</tr>
<tr>
<td>SURFACE</td>
<td>Unique surface ID for each building exterior surface. - Unit: [-]</td>
</tr>
<tr>
<td>TYPE</td>
<td>Surface typology. - Unit: [-]</td>
</tr>
<tr>
<td>Xcoor</td>
<td>Describes the position of the x vector. - Unit: [-]</td>
</tr>
<tr>
<td>Xdir</td>
<td>Directional scalar of the x vector. - Unit: [-]</td>
</tr>
<tr>
<td>Ycoor</td>
<td>Describes the position of the y vector. - Unit: [-]</td>
</tr>
<tr>
<td>Ydir</td>
<td>Directional scalar of the y vector. - Unit: [-]</td>
</tr>
<tr>
<td>Zcoor</td>
<td>Describes the position of the z vector. - Unit: [-]</td>
</tr>
<tr>
<td>Zdir</td>
<td>Directional scalar of the z vector. - Unit: [-]</td>
</tr>
<tr>
<td>area_installed_module_m2</td>
<td>The area of the building suface covered by one solar panel - Unit: [m2]</td>
</tr>
<tr>
<td>array_spacing_m</td>
<td>Spacing between solar arrays. - Unit: [m]</td>
</tr>
<tr>
<td>orientation</td>
<td>Orientation of the surface (north/east/south/west/top) - Unit: [-]</td>
</tr>
<tr>
<td>surface</td>
<td>Unique surface ID for each building exterior surface. - Unit: [-]</td>
</tr>
<tr>
<td>surface_azimuth_deg</td>
<td>Azimuth angle of the panel surface e.g. south facing = 180 deg - Unit: [deg]</td>
</tr>
<tr>
<td>tilt_deg</td>
<td>Tilt angle of roof or walls - Unit: [deg]</td>
</tr>
<tr>
<td>total_rad_Whm2</td>
<td>Total radiative potential of a given surfaces area. - Unit: [Wh/m2]</td>
</tr>
<tr>
<td>type_orientation</td>
<td>Concatenated surface type and orientation. - Unit: [-]</td>
</tr>
</tbody>
</table>

6.2.2 PVT_results

The following file is used by scripts: []
### Table 64: `outputs/data/potentials/solar/b001_pvt.csv`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area_PVT_m2</td>
<td>Total area of investigated collector. - Unit: [m²]</td>
</tr>
<tr>
<td>Date</td>
<td>Date and time in hourly steps. - Unit: [datetime]</td>
</tr>
<tr>
<td>E_PVT_gen_kWh</td>
<td>Total electricity generated by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>Eaux_PVT_kWh</td>
<td>Auxiliary electricity consumed by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_roofs_top_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on roof tops - Unit: kWh</td>
</tr>
<tr>
<td>PVT_roofs_top_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on roof tops - Unit: kWh</td>
</tr>
<tr>
<td>PVT_roofs_top_m2</td>
<td>Collector surface area on roof tops. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_east_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on east facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_east_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on east facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_east_m2</td>
<td>Collector surface area on east facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_north_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on north facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_north_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on north facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_north_m2</td>
<td>Collector surface area on north facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_south_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on south facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_south_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on south facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_south_m2</td>
<td>Collector surface area on south facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_west_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on west facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_west_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on west facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_west_m2</td>
<td>West facing wall collector surface area. - Unit: [m²]</td>
</tr>
<tr>
<td>Q_PVT_gen_kWh</td>
<td>Total heat generated by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>Q_PVT_L_kWh</td>
<td>Collector heat loss. - Unit: [kWh]</td>
</tr>
<tr>
<td>T_PVT_re_C</td>
<td>Collector heating supply temperature. - Unit: [°C]</td>
</tr>
<tr>
<td>T_PVT_sup_C</td>
<td>Collector heating supply temperature. - Unit: [°C]</td>
</tr>
<tr>
<td>mcp_PVT_kWperC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the hot water delivered by the collector. - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>radiation_kWh</td>
<td>Total radiatiative potential. - Unit: [kWh]</td>
</tr>
</tbody>
</table>

### 6.2.3 PVT_total_buildings

The following file is used by scripts: []
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area_PVT_m2</td>
<td>Total area of investigated collector. - Unit: [m²]</td>
</tr>
<tr>
<td>E_PVT_gen_kWh</td>
<td>Total electricity generated by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>Eaux_PVT_kWh</td>
<td>Auxiliary electricity consumed by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_roofs_top_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on roof tops - Unit: kWh</td>
</tr>
<tr>
<td>PVT_roofs_top_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on roof tops - Unit: kWh</td>
</tr>
<tr>
<td>PVT_roofs_top_m2</td>
<td>Collector surface area on roof tops. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_east_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on east facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_east_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on east facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_east_m2</td>
<td>Collector surface area on east facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_north_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on north facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_north_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on north facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_north_m2</td>
<td>Collector surface area on north facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_south_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on south facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_south_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on south facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_south_m2</td>
<td>Collector surface area on south facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_west_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on west facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_west_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on west facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_west_m2</td>
<td>West facing wall collector surface area. - Unit: [m²]</td>
</tr>
<tr>
<td>Q_PVT_gen_kWh</td>
<td>Total heat generated by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>Q_PVT_l_kWh</td>
<td>Collector heat loss. - Unit: [kWh]</td>
</tr>
<tr>
<td>Unnamed: 0</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>radiation_kWh</td>
<td>Total radiative potential. - Unit: [kWh]</td>
</tr>
</tbody>
</table>

### 6.2.4 PVT_totals

The following file is used by scripts: []
Table 66: outputs/data/potentials/solar/pvt_total.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area_PVT_m2</td>
<td>Total area of investigated collector. - Unit: [m²]</td>
</tr>
<tr>
<td>Date</td>
<td>Date and time in hourly steps. - Unit: [datetime]</td>
</tr>
<tr>
<td>E_PVT_gen_kWh</td>
<td>Total electricity generated by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>Eaux_PVT_kWh</td>
<td>Auxiliary electricity consumed by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_roofs_top_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on roof tops - Unit: kWh</td>
</tr>
<tr>
<td>PVT_roofs_top_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on roof tops - Unit: kWh</td>
</tr>
<tr>
<td>PVT_roofs_top_m2</td>
<td>Collector surface area on roof tops. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_east_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on east facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_east_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on east facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_east_m2</td>
<td>Collector surface area on east facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_north_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on north facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_north_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on north facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_north_m2</td>
<td>Collector surface area on north facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_south_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on south facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_south_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on south facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_south_m2</td>
<td>Collector surface area on south facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PVT_walls_west_E_kWh</td>
<td>Electricity production from photovoltaic-thermal panels on west facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_west_Q_kWh</td>
<td>Heat production from photovoltaic-thermal panels on west facades - Unit: kWh</td>
</tr>
<tr>
<td>PVT_walls_west_m2</td>
<td>West facing wall collector surface area. - Unit: [m²]</td>
</tr>
<tr>
<td>Q_PVT_gen_kWh</td>
<td>Total heat generated by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>Q_PVT_l_kWh</td>
<td>Collector heat loss. - Unit: [kWh]</td>
</tr>
<tr>
<td>T_PVT_re_C</td>
<td>Collector heating supply temperature. - Unit: [C]</td>
</tr>
<tr>
<td>T_PVT_sup_C</td>
<td>Collector heating supply temperature. - Unit: [C]</td>
</tr>
<tr>
<td>mcp_PVT_kWperC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the hot water delivered by the collector. - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>radiation_kWh</td>
<td>Total radiative potential. - Unit: [kWh]</td>
</tr>
</tbody>
</table>

6.2.5 PV metadata results

The following file is used by scripts: []
Table 67: outputs/data/potentials/solar/b001_pv_sensors.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA_m2</td>
<td>Surface area. - Unit: [m²]</td>
</tr>
<tr>
<td>BUILDING</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>B_deg</td>
<td>Tilt angle of the installed solar panels - Unit: [deg]</td>
</tr>
<tr>
<td>CATB</td>
<td>Category according to the tilt angle of the panel - Unit: [-]</td>
</tr>
<tr>
<td>CATGB</td>
<td>Category according to the annual radiation on the panel surface - Unit: [-]</td>
</tr>
<tr>
<td>CATteta_z</td>
<td>Category according to the surface azimuth of the panel - Unit: [-]</td>
</tr>
<tr>
<td>SURFACE</td>
<td>Unique surface ID for each building exterior surface. - Unit: [-]</td>
</tr>
<tr>
<td>TYPE</td>
<td>Surface typology. - Unit: [-]</td>
</tr>
<tr>
<td>Xcoor</td>
<td>Describes the position of the x vector. - Unit: [-]</td>
</tr>
<tr>
<td>Xdir</td>
<td>Directional scalar of the x vector. - Unit: [-]</td>
</tr>
<tr>
<td>Ycoor</td>
<td>Describes the position of the y vector. - Unit: [-]</td>
</tr>
<tr>
<td>Ydir</td>
<td>Directional scalar of the y vector. - Unit: [-]</td>
</tr>
<tr>
<td>Zcoor</td>
<td>Describes the position of the z vector. - Unit: [-]</td>
</tr>
<tr>
<td>Zdir</td>
<td>Directional scalar of the z vector. - Unit: [-]</td>
</tr>
<tr>
<td>area_installed_module_m2</td>
<td>The area of the building surface covered by one solar panel - Unit: [m²]</td>
</tr>
<tr>
<td>array_spacing_m</td>
<td>Spacing between solar arrays. - Unit: [m]</td>
</tr>
<tr>
<td>orientation</td>
<td>Orientation of the surface (north/east/south/west/top) - Unit: [-]</td>
</tr>
<tr>
<td>surface</td>
<td>Unique surface ID for each building exterior surface. - Unit: [-]</td>
</tr>
<tr>
<td>surface_azimuth_deg</td>
<td>Azimuth angle of the panel surface e.g. south facing = 180 deg - Unit: [deg]</td>
</tr>
<tr>
<td>tilt_deg</td>
<td>Tilt angle of roof or walls - Unit: [deg]</td>
</tr>
<tr>
<td>total_rad_Whm2</td>
<td>Total radiative potential of a given surfaces area. - Unit: [Wh/m²]</td>
</tr>
<tr>
<td>type_orientation</td>
<td>Concatenated surface type and orientation. - Unit: [-]</td>
</tr>
</tbody>
</table>

6.2.6 PV_results

The following file is used by scripts: []

Table 68: outputs/data/potentials/solar/b001_pv.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area_PV_m2</td>
<td>Total area of investigated collector. - Unit: [m²]</td>
</tr>
<tr>
<td>Date</td>
<td>Date and time in hourly steps. - Unit: [datetime]</td>
</tr>
<tr>
<td>E_PV_gen_kWh</td>
<td>Total electricity generated by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_roofs_top_E_kWh</td>
<td>Electricity production from photovoltaic panels on roof tops - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_roofs_top_m2</td>
<td>Collector surface area on roof tops. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_east_E_kWh</td>
<td>Electricity production from photovoltaic panels on east facades - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_east_m2</td>
<td>Collector surface area on east facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_north_E_kWh</td>
<td>Electricity production from photovoltaic panels on north facades - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_north_m2</td>
<td>Collector surface area on north facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_south_E_kWh</td>
<td>Electricity production from photovoltaic panels on south facades - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_south_m2</td>
<td>Collector surface area on south facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_west_E_kWh</td>
<td>Electricity production from photovoltaic panels on west facades - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_west_m2</td>
<td>West facing wall collector surface area. - Unit: [kWh]</td>
</tr>
<tr>
<td>radiation_kWh</td>
<td>Total radiative potential. - Unit: [kWh]</td>
</tr>
</tbody>
</table>

6.2.7 PV_total_buildings

The following file is used by scripts: []
Table 69: `outputs/data/potentials/solar/pv_total_buildings.csv`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area_PV_m2</td>
<td>Total area of investigated collector. - Unit: [m²]</td>
</tr>
<tr>
<td>E_PV_gen_kWh</td>
<td>Total electricity generated by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_roofs_top_E_kWh</td>
<td>Electricity production from photovoltaic panels on roof tops - Unit: kWh</td>
</tr>
<tr>
<td>PV_roofs_top_m2</td>
<td>Collector surface area on roof tops. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_east_E_kWh</td>
<td>Electricity production from photovoltaic panels on east facades - Unit: kWh</td>
</tr>
<tr>
<td>PV_walls_east_m2</td>
<td>Collector surface area on east facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_north_E_kWh</td>
<td>Electricity production from photovoltaic panels on north facades - Unit: kWh</td>
</tr>
<tr>
<td>PV_walls_north_m2</td>
<td>Collector surface area on north facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_south_E_kWh</td>
<td>Electricity production from photovoltaic panels on south facades - Unit: kWh</td>
</tr>
<tr>
<td>PV_walls_south_m2</td>
<td>Collector surface area on south facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_south_m2</td>
<td>Collector surface area on north facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_west_E_kWh</td>
<td>Electricity production from photovoltaic panels on west facades - Unit: kWh</td>
</tr>
<tr>
<td>PV_walls_west_m2</td>
<td>West facing wall collector surface area. - Unit: [kWh]</td>
</tr>
<tr>
<td>Unnamed: 0</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>radiation_kWh</td>
<td>Total radiative potential. - Unit: [kWh]</td>
</tr>
</tbody>
</table>

### 6.2.8 PV_totals

The following file is used by scripts: []

Table 70: `outputs/data/potentials/solar/pv_total.csv`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area_PV_m2</td>
<td>Total area of investigated collector. - Unit: [m²]</td>
</tr>
<tr>
<td>Date</td>
<td>Date and time in hourly steps. - Unit: [datetime]</td>
</tr>
<tr>
<td>E_PV_gen_kWh</td>
<td>Total electricity generated by the collector. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_roofs_top_E_kWh</td>
<td>Electricity production from photovoltaic panels on roof tops - Unit: kWh</td>
</tr>
<tr>
<td>PV_roofs_top_m2</td>
<td>Collector surface area on roof tops. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_east_E_kWh</td>
<td>Electricity production from photovoltaic panels on east facades - Unit: kWh</td>
</tr>
<tr>
<td>PV_walls_east_m2</td>
<td>Collector surface area on east facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_north_E_kWh</td>
<td>Electricity production from photovoltaic panels on north facades - Unit: kWh</td>
</tr>
<tr>
<td>PV_walls_north_m2</td>
<td>Collector surface area on north facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_south_E_kWh</td>
<td>Electricity production from photovoltaic panels on south facades - Unit: kWh</td>
</tr>
<tr>
<td>PV_walls_south_m2</td>
<td>Collector surface area on south facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_south_m2</td>
<td>Collector surface area on south facades. - Unit: [kWh]</td>
</tr>
<tr>
<td>PV_walls_west_E_kWh</td>
<td>Electricity production from photovoltaic panels on west facades - Unit: kWh</td>
</tr>
<tr>
<td>PV_walls_west_m2</td>
<td>West facing wall collector surface area. - Unit: [kWh]</td>
</tr>
<tr>
<td>radiation_kWh</td>
<td>Total radiative potential. - Unit: [kWh]</td>
</tr>
</tbody>
</table>

### 6.2.9 SC_metadata_results

The following file is used by scripts: []
Table 71: `outputs/data/potentials/solar/b001_sc_et_sensors.csv`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA_m2</td>
<td>Surface area. - Unit: [m²]</td>
</tr>
<tr>
<td>BUILDING</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>B_deg</td>
<td>Tilt angle of the installed solar panels - Unit: [deg]</td>
</tr>
<tr>
<td>CATB</td>
<td>Category according to the tilt angle of the panel - Unit: [-]</td>
</tr>
<tr>
<td>CATGB</td>
<td>Category according to the annual radiation on the panel surface - Unit: [-]</td>
</tr>
<tr>
<td>CATeta_z</td>
<td>Category according to the surface azimuth of the panel - Unit: [-]</td>
</tr>
<tr>
<td>SURFACE</td>
<td>Unique surface ID for each building exterior surface. - Unit: [-]</td>
</tr>
<tr>
<td>TYPE</td>
<td>Surface typology. - Unit: [-]</td>
</tr>
<tr>
<td>Xcoor</td>
<td>Describes the position of the x vector. - Unit: [-]</td>
</tr>
<tr>
<td>Xdir</td>
<td>Directional scalar of the x vector. - Unit: [-]</td>
</tr>
<tr>
<td>Ycoor</td>
<td>Describes the position of the y vector. - Unit: [-]</td>
</tr>
<tr>
<td>Ydir</td>
<td>Directional scalar of the y vector. - Unit: [-]</td>
</tr>
<tr>
<td>Zcoor</td>
<td>Describes the position of the z vector. - Unit: [-]</td>
</tr>
<tr>
<td>Zdir</td>
<td>Directional scalar of the z vector. - Unit: [-]</td>
</tr>
<tr>
<td>area_installed_module_m2</td>
<td>The area of the building surface covered by one solar panel - Unit: [m²]</td>
</tr>
<tr>
<td>array_spacing_m</td>
<td>Spacing between solar arrays. - Unit: [m]</td>
</tr>
<tr>
<td>orientation</td>
<td>Orientation of the surface (north/east/south/west/top) - Unit: [-]</td>
</tr>
<tr>
<td>surface</td>
<td>Unique surface ID for each building exterior surface. - Unit: [-]</td>
</tr>
<tr>
<td>surface_azimuth_deg</td>
<td>Azimuth angle of the panel surface e.g. south facing = 180 deg - Unit: [deg]</td>
</tr>
<tr>
<td>tilt_deg</td>
<td>Tilt angle of roof or walls - Unit: [deg]</td>
</tr>
<tr>
<td>total_rad_Whm2</td>
<td>Total radiative potential of a given surfaces area. - Unit: [Wh/m²]</td>
</tr>
<tr>
<td>type_orientation</td>
<td>Concatenated surface type and orientation. - Unit: [-]</td>
</tr>
</tbody>
</table>

6.2.10 SC_results

The following file is used by scripts: []

Table 72: `outputs/data/potentials/solar/b001_sc_et.csv`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area_SC_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Date</td>
<td>Date and time in hourly steps. - Unit: [datetime]</td>
</tr>
<tr>
<td>Eaux_SC_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Q_SC_gen_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Q_SC_l_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_roofs_top_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_roofs_top_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_east_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_east_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_north_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_north_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_south_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_south_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_west_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_west_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>T_SC_re_C</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>T_SC_sup_C</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>mcp_SC_kWperC</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>radiation_kWh</td>
<td>Total radiative potential. - Unit: [kWh]</td>
</tr>
</tbody>
</table>
6.2.11 SC_total_buildings

The following file is used by scripts: []

Table 73: outputs/data/potentials/solar/sc_et_total_buildings.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area_SC_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Eaux_SC_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Q_SC_gen_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Q_SC_l_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_roofs_top_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_roofs_top_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_east_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_east_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_north_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_north_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_south_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_south_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_west_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_west_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Unnamed: 0</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>radiation_kWh</td>
<td>Total radiatiative potential. - Unit: [kWh]</td>
</tr>
</tbody>
</table>

6.2.12 SC_totals

The following file is used by scripts: []

Table 74: outputs/data/potentials/solar/sc_et_total.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area_SC_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Date</td>
<td>Date and time in hourly steps. - Unit: [datetime]</td>
</tr>
<tr>
<td>Eaux_SC_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Q_SC_gen_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Q_SC_l_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_roofs_top_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_roofs_top_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_east_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_east_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_north_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_north_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_south_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_south_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_west_Q_kWh</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>SC_walls_west_m2</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>T_SC_re_C</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>T_SC_sup_C</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>mcp_SC_kWperC</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>radiation_kWh</td>
<td>Total radiatiative potential. - Unit: [kWh]</td>
</tr>
</tbody>
</table>
6.2.13 get_building_air_conditioning

The following file is used by scripts: `['demand']`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>type_cs</td>
<td>Type of cooling supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_ctrl</td>
<td>Type of heating and cooling control systems (relates to values in Default Database HVAC Properties) - Unit: [code]</td>
</tr>
<tr>
<td>type_dhw</td>
<td>Type of hot water supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_hs</td>
<td>Type of heating supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_vent</td>
<td>Type of ventilation strategy (relates to values in Default Database HVAC Properties) - Unit: [code]</td>
</tr>
</tbody>
</table>

6.2.14 get_building_architecture

The following file is used by scripts: `['radiation', 'emissions', 'demand']`

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Es</td>
<td>Fraction of gross floor area with electrical demands. - Unit: [m2/m2]</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>Ns</td>
<td>Fraction of net gross floor area. - Unit: [m2/m2]</td>
</tr>
<tr>
<td>type_cons</td>
<td>Type of construction. It relates to the contents of the default database of Envelope Properties: construction - Unit: [code]</td>
</tr>
<tr>
<td>type_leak</td>
<td>Leakage level. It relates to the contents of the default database of Envelope Properties: leakage - Unit: [code]</td>
</tr>
<tr>
<td>type_roof</td>
<td>Roof construction type (relates to values in Default Database Construction Properties) - Unit: [-]</td>
</tr>
<tr>
<td>type_shade</td>
<td>Shading system type (relates to values in Default Database Construction Properties) - Unit: [m2/m2]</td>
</tr>
<tr>
<td>type_wall</td>
<td>Wall construction type (relates to values in Default Database Construction Properties) - Unit: [m2/m2]</td>
</tr>
<tr>
<td>type_win</td>
<td>Window type (relates to values in Default Database Construction Properties) - Unit: [m2/m2]</td>
</tr>
<tr>
<td>void_deck</td>
<td>Number of floors (from the ground up) with an open envelope (default = 0) - Unit: [-]</td>
</tr>
<tr>
<td>wwr_east</td>
<td>Window to wall ratio in in facades facing east - Unit: [m2/m2]</td>
</tr>
<tr>
<td>wwr_north</td>
<td>Window to wall ratio in in facades facing north - Unit: [m2/m2]</td>
</tr>
<tr>
<td>wwr_south</td>
<td>Window to wall ratio in in facades facing south - Unit: [m2/m2]</td>
</tr>
<tr>
<td>wwr_west</td>
<td>Window to wall ratio in in facades facing west - Unit: [m2/m2]</td>
</tr>
</tbody>
</table>

6.2.15 get_building_comfort

The following file is used by scripts: `['demand']`
Table 77: inputs/building-properties/indoor_comfort.dbf

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>RH_max_pc</td>
<td>Upper bound of relative humidity - Unit: [%]</td>
</tr>
<tr>
<td>RH_min_pc</td>
<td>Lower bound of relative humidity - Unit: [%]</td>
</tr>
<tr>
<td>Tcs_set_C</td>
<td>Setpoint temperature for cooling system - Unit: [C]</td>
</tr>
<tr>
<td>Tcs_setb_C</td>
<td>Setback point of temperature for cooling system - Unit: [C]</td>
</tr>
<tr>
<td>Ths_set_C</td>
<td>Setpoint temperature for heating system - Unit: [C]</td>
</tr>
<tr>
<td>Ths_setb_C</td>
<td>Setback point of temperature for heating system - Unit: [C]</td>
</tr>
<tr>
<td>Ve_lpspax</td>
<td>Indoor quality requirements of indoor ventilation per person - Unit: [l/s]</td>
</tr>
</tbody>
</table>

6.2.16 get_building_internal

The following file is used by scripts: ['demand']

Table 78: inputs/building-properties/internal_loads.dbf

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ea_Wm2</td>
<td>Peak specific electrical load due to computers and devices - Unit: [W/m2]</td>
</tr>
<tr>
<td>Ed_Wm2</td>
<td>Peak specific electrical load due to servers/data centres - Unit: [W/m2]</td>
</tr>
<tr>
<td>El_Wm2</td>
<td>Peak specific electrical load due to artificial lighting - Unit: [W/m2]</td>
</tr>
<tr>
<td>Epro_Wm2</td>
<td>Peak specific electrical load due to industrial processes - Unit: [W/m2]</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>Qcpro_Wm2</td>
<td>Peak specific process cooling load - Unit: [W/m2]</td>
</tr>
<tr>
<td>Qcre_Wm2</td>
<td>Peak specific cooling load due to refrigeration (cooling rooms) - Unit: [W/m2]</td>
</tr>
<tr>
<td>Qhpro_Wm2</td>
<td>Peak specific process heating load - Unit: [W/m2]</td>
</tr>
<tr>
<td>Vw_lpdpax</td>
<td>Peak specific fresh water consumption (includes cold and hot water) - Unit: [lpd]</td>
</tr>
<tr>
<td>Vww_lpdpax</td>
<td>Peak specific daily hot water consumption - Unit: [lpd]</td>
</tr>
<tr>
<td>X_ghpax</td>
<td>Moisture released by occupancy at peak conditions - Unit: [gh/kg/p]</td>
</tr>
</tbody>
</table>

6.2.17 get_building_supply

The following file is used by scripts: ['demand', 'operation-costs', 'emissions']

Table 79: inputs/building-properties/supply_systems.dbf

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>type_cs</td>
<td>Type of cooling supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_dhw</td>
<td>Type of hot water supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_el</td>
<td>Type of electrical supply system - Unit: [code]</td>
</tr>
<tr>
<td>type_hs</td>
<td>Type of heating supply system - Unit: [code]</td>
</tr>
</tbody>
</table>

6.2.18 get_costs_operation_file

The following file is used by scripts: []

6.2. Output
6.2.19 get_demand_results_file

The following file is used by scripts: [‘sewage-potential’, ‘thermal-network’]

Table 81: outputs/data/demand/b001.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COAL_hs_kWh</td>
<td>Coal requirement for space heating supply - Unit: kWh</td>
</tr>
<tr>
<td>COAL_ww_kWh</td>
<td>Coal requirement for hotwater supply - Unit: kWh</td>
</tr>
<tr>
<td>DATE</td>
<td>Time stamp for each day of the year ascending in hour intervals. - Unit: [smalldatetime]</td>
</tr>
<tr>
<td>DC_cdata_kWh</td>
<td>District cooling for data center cooling demand - Unit: kWh</td>
</tr>
<tr>
<td>DC_cre_kWh</td>
<td>District cooling for refrigeration demand - Unit: kWh</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DC_cs_kWh</td>
<td>District cooling for space cooling demand - Unit: kWh</td>
</tr>
<tr>
<td>DH_hs_kWh</td>
<td>Energy requirement by district heating (space heating supply) - Unit: kWh</td>
</tr>
<tr>
<td>DH_ww_kWh</td>
<td>Energy requirement by district heating (hotwater supply) - Unit: kWh</td>
</tr>
<tr>
<td>E_cdata_kWh</td>
<td>Data centre cooling specific electricity consumption. - Unit: [kWh]</td>
</tr>
<tr>
<td>E_cre_kWh</td>
<td>Refrigeration system electricity consumption. - Unit: [kWh]</td>
</tr>
<tr>
<td>E_cs_kWh</td>
<td>Cooling system electricity consumption. - Unit: [kWh]</td>
</tr>
<tr>
<td>E_hs_kWh</td>
<td>Heating system electricity consumption. - Unit: [kWh]</td>
</tr>
<tr>
<td>E_sys_kWh</td>
<td>End-use total electricity system consumption = Ea + El + Edata + Epro + Eaux - Unit: kWh</td>
</tr>
<tr>
<td>E_ww_kWh</td>
<td>Hot water system electricity consumption. - Unit: [kWh]</td>
</tr>
<tr>
<td>Eal_kWh</td>
<td>End-use electricity consumption of appliances and lights - Unit: [kWh]</td>
</tr>
<tr>
<td>Eaux_kWh</td>
<td>End-use auxiliary electricity consumption. - Unit: [kWh]</td>
</tr>
<tr>
<td>Edata_kWh</td>
<td>End-use data centre electricity consumption. - Unit: [kWh]</td>
</tr>
<tr>
<td>Epro_kWh</td>
<td>End-use electricity consumption for industrial processes. - Unit: [kWh]</td>
</tr>
<tr>
<td>GRID_kWh</td>
<td>Grid total requirements of electricity = GRID_a + GRID_l + GRID_data + GRID_pro + GRID_aux + GRID_cdata + GRID_cre + GRID_hs + GRID_ww + GRID_cs - Unit: kWh</td>
</tr>
<tr>
<td>I_rad_kWh</td>
<td>Radiative heat loss - Unit: kWh</td>
</tr>
<tr>
<td>I_sol_and_I_rad_kWh</td>
<td>Net radiative heat gain - Unit: [kWh]</td>
</tr>
<tr>
<td>I_sol_kWh</td>
<td>Solar heat gain - Unit: kWh</td>
</tr>
<tr>
<td>NG_hs_kWh</td>
<td>NG requirement for space heating supply - Unit: kWh</td>
</tr>
<tr>
<td>NG_ww_kWh</td>
<td>NG requirement for hotwater supply - Unit: kWh</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>OIL_hs_kWh</td>
<td>OIL requirement for space heating supply - Unit: kWh</td>
</tr>
<tr>
<td>OIL_ww_kWh</td>
<td>OIL requirement for hotwater supply - Unit: kWh</td>
</tr>
<tr>
<td>PV_kWh</td>
<td>PV electricity consumption - Unit: kWh</td>
</tr>
<tr>
<td>QC_sys_kWh</td>
<td>Total cool consumption - Unit: [kWh]</td>
</tr>
<tr>
<td>QH_sys_kWh</td>
<td>Total heat consumption - Unit: [kWh]</td>
</tr>
<tr>
<td>Q_gain_lat_peop_kWh</td>
<td>Latent heat gain from people - Unit: kWh</td>
</tr>
<tr>
<td>Q_gain_sen_app_kWh</td>
<td>Sensible heat gain from appliances - Unit: kWh</td>
</tr>
<tr>
<td>Q_gain_sen_base_kWh</td>
<td>Sensible heat gain from transmission through the base - Unit: kWh</td>
</tr>
<tr>
<td>Q_gain_sen_data_kWh</td>
<td>Sensible heat gain from data centres - Unit: kWh</td>
</tr>
<tr>
<td>Q_gain_sen_light_kWh</td>
<td>Sensible heat gain from lighting - Unit: kWh</td>
</tr>
<tr>
<td>Q_gain_sen_peop_kWh</td>
<td>Sensible heat gain from people - Unit: kWh</td>
</tr>
<tr>
<td>Q_gain_sen_pro_kWh</td>
<td>Sensible heat gain from industrial processes. - Unit: [kWh]</td>
</tr>
<tr>
<td>Q_gain_sen_roof_kWh</td>
<td>Sensible heat gain from transmission through the roof - Unit: kWh</td>
</tr>
<tr>
<td>Q_gain_sen_vent_kWh</td>
<td>Sensible heat gain from ventilation and infiltration - Unit: kWh</td>
</tr>
<tr>
<td>Q_gain_sen_wall_kWh</td>
<td>Sensible heat gain from transmission through the walls - Unit: kWh</td>
</tr>
<tr>
<td>Q_gain_sen_wind_kWh</td>
<td>Sensible heat gain from transmission through the windows - Unit: kWh</td>
</tr>
<tr>
<td>Q_loss_sen_ref_kWh</td>
<td>Sensible heat loss from refrigeration systems - Unit: kWh</td>
</tr>
<tr>
<td>Qcdata_kWh</td>
<td>Data centre space cooling demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcdata_sys_kWh</td>
<td>End-use data center cooling demand - Unit: kWh</td>
</tr>
<tr>
<td>Qcre_kWh</td>
<td>Refrigeration space cooling demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcre_sys_kWh</td>
<td>End-use refrigeration demand - Unit: kWh</td>
</tr>
<tr>
<td>Qcs_dis_ls_kWh</td>
<td>Cooling system distribution losses - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_em_ls_kWh</td>
<td>Cooling system emission losses - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_kWh</td>
<td>Specific cool demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_lat_ahu_kWh</td>
<td>AHU latent cool demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_lat_aru_kWh</td>
<td>ARU latent cool demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_lat_sys_kWh</td>
<td>Total latent cool demand for all systems - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_sen_ahu_kWh</td>
<td>AHU sensible cool demand - Unit: [kWh]</td>
</tr>
</tbody>
</table>

6.2. Output
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qcs_sen_aru_kWh</td>
<td>ARU sensible cool demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_sen_scu_kWh</td>
<td>SHU sensible cool demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_sen_sys_kWh</td>
<td>Total sensible cool demand for all systems - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_sys_ahu_kWh</td>
<td>AHU system cool demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_sys_aru_kWh</td>
<td>ARU system cool demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qcs_sys_kWh</td>
<td>End-use space cooling demand - Unit: kWh</td>
</tr>
<tr>
<td>Qcs_sys_scu_kWh</td>
<td>SCU system cool demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhpro_sys_kWh</td>
<td>Process heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_dis_ls_kWh</td>
<td>Heating system distribution losses - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_em_ls_kWh</td>
<td>Heating system emission losses - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_kWh</td>
<td>Sensible heating system demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_lat_ahu_kWh</td>
<td>AHU latent heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_lat_aru_kWh</td>
<td>ARU latent heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_lat_sys_kWh</td>
<td>Total latent heat demand for all systems - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_sen_ahu_kWh</td>
<td>AHU sensible heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_sen_aru_kWh</td>
<td>ARU sensible heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_sen_scu_kWh</td>
<td>SHU sensible heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_sen_sys_kWh</td>
<td>Total sensible heat demand for all systems - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_sys_ahu_kWh</td>
<td>AHU system heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_sys_aru_kWh</td>
<td>ARU system heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qhs_sys_kWh</td>
<td>End-use space heating demand - Unit: kWh</td>
</tr>
<tr>
<td>Qhs_sys_shu_kWh</td>
<td>SHU system heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qww_kWh</td>
<td>DHW specific heat demand - Unit: [kWh]</td>
</tr>
<tr>
<td>Qww_sys_kWh</td>
<td>End-use hotwater demand - Unit: kWh</td>
</tr>
<tr>
<td>SOLAR_hs_kWh</td>
<td>Solar thermal energy requirement for space heating supply - Unit: kWh</td>
</tr>
<tr>
<td>SOLAR_ww_kWh</td>
<td>Solar thermal energy requirement for hotwater supply - Unit: kWh</td>
</tr>
<tr>
<td>T_ext_C</td>
<td>Outdoor temperature - Unit: C</td>
</tr>
<tr>
<td>T_int_C</td>
<td>Indoor temperature - Unit: C</td>
</tr>
<tr>
<td>Tcsdata_sys_re_C</td>
<td>Cooling supply temperature of the data centre - Unit: [C]</td>
</tr>
<tr>
<td>Tcsdata_sys_sup_C</td>
<td>Cooling return temperature of the data centre - Unit: [C]</td>
</tr>
<tr>
<td>Tcre_sys_re_C</td>
<td>Cooling return temperature of the refrigeration system. - Unit: [C]</td>
</tr>
<tr>
<td>Tcre_sys_sup_C</td>
<td>Cooling supply temperature of the refrigeration system. - Unit: [C]</td>
</tr>
<tr>
<td>Tcs_sys_re_C</td>
<td>System cooling return temperature. - Unit: [C]</td>
</tr>
<tr>
<td>Tcs_sys_re_ahu_C</td>
<td>Return temperature cooling system - Unit: C</td>
</tr>
<tr>
<td>Tcs_sys_re_aru_C</td>
<td>Return temperature cooling system - Unit: C</td>
</tr>
<tr>
<td>Tcs_sys_re_scu_C</td>
<td>Return temperature cooling system - Unit: C</td>
</tr>
<tr>
<td>Tcs_sys_sup_C</td>
<td>Supply temperature cooling system. - Unit: [C]</td>
</tr>
<tr>
<td>Tcs_sys_sup_ahu_C</td>
<td>Supply temperature cooling system - Unit: C</td>
</tr>
<tr>
<td>Tcs_sys_sup_aru_C</td>
<td>Supply temperature cooling system - Unit: C</td>
</tr>
<tr>
<td>Tcs_sys_sup_scu_C</td>
<td>Supply temperature cooling system - Unit: C</td>
</tr>
<tr>
<td>Ths_sys_re_C</td>
<td>Heating system return temperature. - Unit: [C]</td>
</tr>
<tr>
<td>Ths_sys_re_ahu_C</td>
<td>Return temperature heating system - Unit: C</td>
</tr>
<tr>
<td>Ths_sys_re_aru_C</td>
<td>Return temperature heating system - Unit: C</td>
</tr>
<tr>
<td>Ths_sys_re_shu_C</td>
<td>Return temperature heating system - Unit: C</td>
</tr>
<tr>
<td>Ths_sys_sup_C</td>
<td>Heating system supply temperature. - Unit: [C]</td>
</tr>
<tr>
<td>Ths_sys_sup_ahu_C</td>
<td>Supply temperature heating system - Unit: C</td>
</tr>
<tr>
<td>Ths_sys_sup_aru_C</td>
<td>Supply temperature heating system - Unit: C</td>
</tr>
<tr>
<td>Ths_sys_sup_shu_C</td>
<td>Supply temperature heating system - Unit: C</td>
</tr>
<tr>
<td>Tww_sys_re_C</td>
<td>Return temperature hotwater system - Unit: C</td>
</tr>
</tbody>
</table>
### Table 81 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tww_sys_sup_C</td>
<td>Supply temperature hotwater system - Unit: °C</td>
</tr>
<tr>
<td>WOOD_hs_kWh</td>
<td>WOOD requirement for space heating supply - Unit: kWh</td>
</tr>
<tr>
<td>WOOD_ww_kWh</td>
<td>WOOD requirement for hotwater supply - Unit: kWh</td>
</tr>
<tr>
<td>mcpcdata_sys_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the chilled water delivered to data centre. - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcpcrc_sys_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the chilled water delivered to refrigeration. - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcpcs_sys_ahu_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the chilled water delivered to air handling units (space cooling). - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcpcs_sys_aru_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the chilled water delivered to air recirculation units (space cooling). - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcpcs_sys_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the chilled water delivered to space cooling. - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcpcs_sys_scu_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the chilled water delivered to sensible cooling units (space cooling). - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcphs_sys_ahu_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the warm water delivered to air handling units (space heating). - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcphs_sys_aru_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the warm water delivered to air recirculation units (space heating). - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcphs_sys_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the warm water delivered to space heating. - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcphs_sys_shu_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the warm water delivered to sensible heating units (space heating). - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcptw_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of the fresh water - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>mcpww_sys_kW_perC</td>
<td>Capacity flow rate (mass flow* specific heat capacity) of domestic hot water - Unit: [kW/Cap]</td>
</tr>
<tr>
<td>people</td>
<td>Predicted occupancy: number of people in building - Unit: [people]</td>
</tr>
<tr>
<td>theta_o_C</td>
<td>Operative temperature in building (RC-model) used for comfort plotting - Unit: °C</td>
</tr>
<tr>
<td>x_int</td>
<td>Internal mass fraction of humidity (water/dry air) - Unit: [kg/kg]</td>
</tr>
</tbody>
</table>

#### 6.2.20 get_edge_mass_flow_csv_file

The following file is used by scripts: []

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPE0</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Unnamed: 0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

#### 6.2.21 get_geothermal_potential

The following file is used by scripts: ['optimization']

#### 6.2.22 get_lake_potential

The following file is used by scripts: []

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hour</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>lake_potential</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

#### 6.2.23 get_lca_embodied

The following file is used by scripts: []
Table 84: outputs/data/emissions/total_lca_embodied.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_ghg_kgm2</td>
<td>Building construction and decommissioning - Unit: kg CO2-eq/m2.yr</td>
</tr>
<tr>
<td>E_ghg_ton</td>
<td>Building construction and decommissioning - Unit: ton CO2-eq/yr</td>
</tr>
<tr>
<td>E_nre_pen_GJ</td>
<td>Building construction and decommissioning - Unit: GJ/yr</td>
</tr>
<tr>
<td>E_nre_pen_MJm2</td>
<td>Building construction and decommissioning - Unit: MJoil-eq/m2.yr</td>
</tr>
<tr>
<td>GFA_m2</td>
<td>Gross floor area - Unit: [m2]</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
</tbody>
</table>

6.2.24 get_lca_mobility

The following file is used by scripts: []

Table 85: outputs/data/emissions/total_lca_mobility.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GFA_m2</td>
<td>Gross floor area - Unit: [m2]</td>
</tr>
<tr>
<td>M_ghg_kgm2</td>
<td>Commuting - Unit: kg CO2-eq/m2.yr</td>
</tr>
<tr>
<td>M_ghg_ton</td>
<td>Commuting - Unit: ton CO2-eq/yr</td>
</tr>
<tr>
<td>M_nre_pen_GJ</td>
<td>Commuting - Unit: GJ/yr</td>
</tr>
<tr>
<td>M_nre_pen_MJm2</td>
<td>Commuting - Unit: MJoil-eq/m2.yr</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
</tbody>
</table>

6.2.25 get_lca_operation

The following file is used by scripts: []

Table 86: outputs/data/emissions/total_lca_operation.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COAL_hs_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the coal powered heating system - Unit: [kg/m2-yr]</td>
</tr>
<tr>
<td>COAL_hs_ghg_ton</td>
<td>Emissions due to operational energy of the coal powered heating system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>COAL_hs_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for coal powered heating system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>COAL_hs_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the coal powered heating system - Unit: [MJ/m2-yr]</td>
</tr>
<tr>
<td>COAL_ww_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the coal powered domestic hot water system - Unit: [kg/m2-yr]</td>
</tr>
<tr>
<td>COAL_ww_ghg_ton</td>
<td>Emissions due to operational energy of the coal powered domestic hot water system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>COAL_ww_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for coal powered domestic hot water system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>COAL_ww_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the coal powered domestic hot water system - Unit: [MJ/m2-yr]</td>
</tr>
<tr>
<td>DC_cdata_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the district cooling for the data center - Unit: [kg/m2-yr]</td>
</tr>
<tr>
<td>DC_cdata_ghg_ton</td>
<td>Emissions due to operational energy of the district cooling for the data center - Unit: [ton/yr]</td>
</tr>
<tr>
<td>DC_cdata_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for district cooling system of the data center - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>DC_cdata_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the district cooling system of the data center - Unit: [MJ/m2-yr]</td>
</tr>
<tr>
<td>DC_cre_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the district cooling for cooling and refrigeration - Unit: [kg/m2-yr]</td>
</tr>
<tr>
<td>DC_cre_ghg_ton</td>
<td>Emissions due to operational energy of the district cooling for cooling and refrigeration - Unit: [ton/yr]</td>
</tr>
<tr>
<td>DC_cre_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for district cooling system for cooling and refrigeration - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>DC_cre_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the district cooling system for cooling and refrigeration - Unit: [MJ/m2-yr]</td>
</tr>
<tr>
<td>DC_cs_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the district cooling - Unit: [kg/m2-yr]</td>
</tr>
<tr>
<td>DC_cs_ghg_ton</td>
<td>Emissions due to operational energy of the district cooling - Unit: [ton/yr]</td>
</tr>
<tr>
<td>DC_cs_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for district cooling system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>DC_cs_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the district cooling system - Unit: [MJ/m2-yr]</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>DH_hs_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the district heating system - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>DH_hs_ghg_ton</td>
<td>Emissions due to operational energy of the district heating system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>DH_hs_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for district heating system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>DH_hs_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the district heating system - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>DH_ww_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the district heating domestic hot water system - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>DH_ww_ghg_ton</td>
<td>Emissions due to operational energy of the district heating domestic hot water system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>DH_ww_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for district heating powered domestic hot water system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>DH_ww_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the district heating domestic hot water system - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>GFA_m2</td>
<td>Gross floor area - Unit: [m^2]</td>
</tr>
<tr>
<td>GRID_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area from grid electricity - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>GRID_ggh_ton</td>
<td>Emissions due to operational energy of the electricity from the grid - Unit: [ton/yr]</td>
</tr>
<tr>
<td>GRID_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) from the grid - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>GRID_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) from grid electricity - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>NG_hs_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the natural gas powered heating systems - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>NG_hs_ghg_ton</td>
<td>Emissions due to operational energy of the natural gas powered heating system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>NG_hs_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for natural gas powered heating system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>NG_hs_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the natural gas powered heating systems - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>NG_ww_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the grid powered domestic hot water system - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>NG_ww_ghg_ton</td>
<td>Emissions due to operational energy of the solar powered domestic hot water system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>NG_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for grid powered domestic hot water system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>NG_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the grid powered domestic hot water system - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>Name.1</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>OIL_hs_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the oil powered heating systems - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>OIL_hs_ghg_ton</td>
<td>Emissions due to operational energy of the oil powered heating system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>OIL_hs_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for oil powered heating system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>OIL_hs_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the oil powered heating systems - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>OIL_ww_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the oil powered domestic hot water system - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>OIL_ww_ghg_ton</td>
<td>Emissions due to operational energy of the oil powered domestic hot water system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>OIL_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for oil powered domestic hot water system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>OIL_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the oil powered domestic hot water system - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>OIL_ww_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the oil powered domestic hot water system - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>O_ghg_kgm2</td>
<td>Energy system operation - Unit: [kg CO2-eq/m^2-yr]</td>
</tr>
<tr>
<td>O_ghg_ton</td>
<td>Energy system operation - Unit: [ton CO2-eq/yr]</td>
</tr>
<tr>
<td>O_nre_pen_GJ</td>
<td>Energy system operation - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>O_nre_pen_MJm2</td>
<td>Energy system operation - Unit: [MJCO2-eq/m^2-yr]</td>
</tr>
<tr>
<td>PV_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area for PV-System - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>PV_ghg_ton</td>
<td>Emissions due to operational energy of the PV-System - Unit: [ton/yr]</td>
</tr>
<tr>
<td>PV_nre_pen_GJ.1</td>
<td>Operational primary energy demand (non-renewable) for PV-System - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>PV_nre_pen_MJm2.1</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) for PV System - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>SOLAR_hs_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the solar powered heating system - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>SOLAR_hs_ghg_ton</td>
<td>Emissions due to operational energy of the solar powered heating system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>SOLAR_hs_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) of the solar powered heating system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>SOLAR_hs_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the solar powered heating systems - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>SOLAR_ww_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the solar powered domestic hot water system - Unit: [kg/m^2-yr]</td>
</tr>
<tr>
<td>SOLAR_ww_ghg_ton</td>
<td>Emissions due to operational energy of the solar powered domestic hot water system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>SOLAR_ww_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) of the solar powered domestic hot water system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>SOLAR_ww_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the solar powered domestic hot water system - Unit: [MJ/m^2-yr]</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SOLAR_ww_ghg_ton</td>
<td>Emissions due to operational energy of the solar powered domestic hot water system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>SOLAR_ww_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for solar powered domestic hot water system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>SOLAR_ww_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the solar powered domestic hot water system - Unit: [MJ/m2-yr]</td>
</tr>
<tr>
<td>WOOD_hs_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the wood powered heating system - Unit: [kg/m2-yr]</td>
</tr>
<tr>
<td>WOOD_hs_ghg_ton</td>
<td>Emissions due to operational energy of the wood powered heating system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>WOOD_hs_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for wood powered heating system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>WOOD_hs_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the wood powered heating system - Unit: [MJ/m2-yr]</td>
</tr>
<tr>
<td>WOOD_ww_ghg_kgm2</td>
<td>Emissions due to operational energy per unit of conditioned floor area of the wood powered domestic hot water system - Unit: [kg/m2-yr]</td>
</tr>
<tr>
<td>WOOD_ww_ghg_ton</td>
<td>Emissions due to operational energy of the wood powered domestic hot water system - Unit: [ton/yr]</td>
</tr>
<tr>
<td>WOOD_ww_nre_pen_GJ</td>
<td>Operational primary energy demand (non-renewable) for wood powered domestic hot water system - Unit: [GJ/yr]</td>
</tr>
<tr>
<td>WOOD_ww_nre_pen_MJm2</td>
<td>Operational primary energy demand per unit of conditioned floor area (non-renewable) of the wood powered domestic hot water system - Unit: [MJ/m2-yr]</td>
</tr>
</tbody>
</table>

### 6.2.26 get_network_layout_edges_shapefile

The following file is used by scripts: []

Table 87: inputs/networks/dc/edges.shp

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>Pipe_DN</td>
<td>Classifies nominal pipe diameters (DN) into typical bins. E.g. DN100 refers to pipes of approx. 100mm in diameter. - Unit: [DN#]</td>
</tr>
<tr>
<td>Type_mat</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>geometry</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>weight</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### 6.2.27 get_network_layout_nodes_shapefile

The following file is used by scripts: ['thermal-network']

Table 88: inputs/networks/dc/nodes.shp

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>Type</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>geometry</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

### 6.2.28 get_node_mass_flow_csv_file

The following file is used by scripts: []

Table 89: outputs/data/optimization/network/layout/nominal_nodemassflow_at_design_dh__kgpers.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE0</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Unnamed: 0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>
6.2.29 get_optimization_network_edge_list_file

The following file is used by scripts: []

Table 90: outputs/data/optimization/network/layout/dh__edges.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_ext_m</td>
<td>Defines the maximum pipe diameter tolerance for the nominal diameter (DN) bin. - Unit: [m]</td>
</tr>
<tr>
<td>D_ins_m</td>
<td>Defines the pipe insulation diameter for the nominal diameter (DN) bin. - Unit: [m]</td>
</tr>
<tr>
<td>D_int_m</td>
<td>Defines the minimum pipe diameter tolerance for the nominal diameter (DN) bin. - Unit: [m]</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>Pipe_DN_x</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Pipe_DN_y</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Type_mat</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Vdot_max_m3s</td>
<td>Maximum volume flow rate for the nominal diameter (DN) bin. - Unit: [m3/s]</td>
</tr>
<tr>
<td>Vdot_min_m3s</td>
<td>Minimum volume flow rate for the nominal diameter (DN) bin. - Unit: [m3/s]</td>
</tr>
<tr>
<td>coordinates</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>end node</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>geometry</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>mdot_max_kgs</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>mdot_min_kgs</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>pipe length</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>start node</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.30 get_optimization_network_edge_node_matrix_file

The following file is used by scripts: []

Table 91: outputs/data/optimization/network/layout/dh__edgenode.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPE0</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Unnamed: 0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.31 get_optimization_network_layout_massflow_file

The following file is used by scripts: []

Table 92: outputs/data/optimization/network/layout/dh__massflow_kgs.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPE0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.32 get_optimization_network_layout_plant_heat_requirement_file

The following file is used by scripts: []

Table 93: outputs/data/optimization/network/layout/dh__plant_heat_requirement_kw.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>
6.2.33 get_optimization_network_layout_ploss_system_edges_file

The following file is used by scripts: []

Table 94: outputs/data/optimization/network/layout/dh__ploss_system_edges_kw.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPE0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.34 get_optimization_network_layout_pressure_drop_file

The following file is used by scripts: []

Table 95: outputs/data/optimization/network/layout/dh__p_deltap_pa.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pressure_loss_return_Pa</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>pressure_loss_substations_Pa</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>pressure_loss_supply_Pa</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>pressure_loss_total_Pa</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.35 get_optimization_network_layout_qloss_system_file

The following file is used by scripts: []

Table 96: outputs/data/optimization/network/layout/dh__qloss_system_kw.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPE0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.36 get_optimization_network_layout_return_temperature_file

The following file is used by scripts: []

Table 97: outputs/data/optimization/network/layout/dh__t_return_k.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.37 get_optimization_network_layout_supply_temperature_file

The following file is used by scripts: []

Table 98: outputs/data/optimization/network/layout/dh__t_supply_k.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>
6.2.38 get_optimization_network_substation_ploss_file

The following file is used by scripts: []

Table 99: outputs/data/optimization/network/layout/dh__ploss_substations_kw.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B001</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.39 get_radiation_building

The following file is used by scripts: ['photovoltaic', 'photovoltaic-thermal', 'solar-collector', 'demand']

Table 100: outputs/data/solar-radiation/b001_insolation_whm2.json

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srf0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.40 get_radiation_metadata

The following file is used by scripts: ['photovoltaic', 'photovoltaic-thermal', 'solar-collector', 'demand']

Table 101: outputs/data/solar-radiation/b001_geometry.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA_m2</td>
<td>Surface area. - Unit: [m^2]</td>
</tr>
<tr>
<td>BUILDING</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>SURFACE</td>
<td>Unique surface ID for each building exterior surface. - Unit: [-]</td>
</tr>
<tr>
<td>TYPE</td>
<td>Surface typology. - Unit: [-]</td>
</tr>
<tr>
<td>Xcoor</td>
<td>Describes the position of the x vector. - Unit: [-]</td>
</tr>
<tr>
<td>Xdir</td>
<td>Directional scalar of the x vector. - Unit: [-]</td>
</tr>
<tr>
<td>Ycoor</td>
<td>Describes the position of the y vector. - Unit: [-]</td>
</tr>
<tr>
<td>Ydir</td>
<td>Directional scalar of the y vector. - Unit: [-]</td>
</tr>
<tr>
<td>Zcoor</td>
<td>Describes the position of the z vector. - Unit: [-]</td>
</tr>
<tr>
<td>Zdir</td>
<td>Directional scalar of the z vector. - Unit: [-]</td>
</tr>
<tr>
<td>orientation</td>
<td>Orientation of the surface (north/east/south/west/top) - Unit: [-]</td>
</tr>
</tbody>
</table>

6.2.41 get_sewage_heat_potential

The following file is used by scripts: []

Table 102: outputs/data/potentials/swp.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qsw_kW</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>tin_HP_C</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>tin_sw_C</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>tout_HP_C</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>tout_sw_C</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>ts_C</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>
6.2.42 get_thermal_demand_csv_file

The following file is used by scripts: []

Table 103: outputs/data/optimization/network/layout/aggregated_demand_dh__wh.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B001</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>Unnamed: 0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.43 get_thermal_network_layout_pressure_drop_kw_file

The following file is used by scripts: []

Table 104: outputs/data/thermal-network/dc__p_deltap_kw.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pressure_loss_return_kW</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>pressure_loss_substations_kW</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>pressure_loss_supply_kW</td>
<td>TODO - Unit: TODO</td>
</tr>
<tr>
<td>pressure_loss_total_kW</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.44 get_thermal_network_node_types_csv_file

The following file is used by scripts: []

6.2.45 get_thermal_network_qloss_system_file

The following file is used by scripts: []

Table 105: outputs/data/thermal-network/dc__p_deltap_kw.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPE0</td>
<td>TODO - Unit: TODO</td>
</tr>
</tbody>
</table>

6.2.46 get_total_demand

The following file is used by scripts: ['sewage-potential’, ‘emissions’, ‘operation-costs’, ‘network-layout’]

Table 106: outputs/data/demand/total_demand.csv

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Af_m2</td>
<td>Conditioned floor area (heated/cooled) - Unit: [m2]</td>
</tr>
<tr>
<td>Aocc_m2</td>
<td>Occupied floor area (heated/cooled) - Unit: [m2]</td>
</tr>
<tr>
<td>Aroof_m2</td>
<td>Roof area - Unit: [m2]</td>
</tr>
<tr>
<td>COAL_hs0_kW</td>
<td>Nominal Coal requirement for space heating supply - Unit: kW</td>
</tr>
<tr>
<td>COAL_hs_MWhyr</td>
<td>Coal requirement for space heating supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>COAL_ww0_kW</td>
<td>Nominal Coal requirement for hotwater supply - Unit: kW</td>
</tr>
<tr>
<td>COAL_ww_MWhyr</td>
<td>Coal requirement for hotwater supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>DC_cdata0_kW</td>
<td>Nominal district cooling for final data center cooling demand - Unit: kW</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC_cdata_MWhyr</td>
<td>District cooling for data center cooling demand - Unit: MWh/yr</td>
</tr>
<tr>
<td>DC_cre0_kW</td>
<td>Nominal district cooling for refrigeration demand - Unit: kW</td>
</tr>
<tr>
<td>DC_cre_MWhyr</td>
<td>District cooling for refrigeration demand - Unit: MWh/yr</td>
</tr>
<tr>
<td>DC_cs0_kW</td>
<td>Nominal district cooling for space cooling demand - Unit: kW</td>
</tr>
<tr>
<td>DC_cs_MWhyr</td>
<td>District cooling for space cooling demand - Unit: MWh/yr</td>
</tr>
<tr>
<td>DH_hs0_kW</td>
<td>Nominal energy requirement by district heating (space heating supply) - Unit: kW</td>
</tr>
<tr>
<td>DH_hs_MWhyr</td>
<td>Energy requirement by district heating (space heating supply) - Unit: MWh/yr</td>
</tr>
<tr>
<td>DH_ww0_kW</td>
<td>Nominal Energy requirement by district heating (hotwater supply) - Unit: kW</td>
</tr>
<tr>
<td>DH_ww_MWhyr</td>
<td>Energy requirement by district heating (hotwater supply) - Unit: MWh/yr</td>
</tr>
<tr>
<td>E_cdata0_kW</td>
<td>Nominal Data centre cooling specific electricity consumption. - Unit: [kW]</td>
</tr>
<tr>
<td>E_cdata_MWhyr</td>
<td>Electricity consumption due to data center cooling - Unit: MWh/yr</td>
</tr>
<tr>
<td>E_cre0_kW</td>
<td>Nominal Refrigeration system electricity consumption. - Unit: [kW]</td>
</tr>
<tr>
<td>E_cre_MWhyr</td>
<td>Electricity consumption due to refrigeration - Unit: MWh/yr</td>
</tr>
<tr>
<td>E_cs0_kW</td>
<td>Nominal Cooling system electricity consumption. - Unit: [kW]</td>
</tr>
<tr>
<td>E_cs_MWhyr</td>
<td>Electricity consumption due to space cooling - Unit: MWh/yr</td>
</tr>
<tr>
<td>E_hs0_kW</td>
<td>Nominal Heating system electricity consumption. - Unit: [kW]</td>
</tr>
<tr>
<td>E_hs_MWhyr</td>
<td>Electricity consumption due to space heating - Unit: MWh/yr</td>
</tr>
<tr>
<td>E_sys0_kW</td>
<td>Nominal end-use electricity demand - Unit: kW</td>
</tr>
<tr>
<td>E_sys_MWhyr</td>
<td>End-use electricity demand - Unit: MWh/yr</td>
</tr>
<tr>
<td>E_ww0_kW</td>
<td>Nominal Domestic hot water electricity consumption. - Unit: [kW]</td>
</tr>
<tr>
<td>E_ww_MWhyr</td>
<td>Electricity consumption due to hot water - Unit: MWh/yr</td>
</tr>
<tr>
<td>Eal0_kW</td>
<td>Nominal Total net electricity for all sources and sinks, - Unit: [kW]</td>
</tr>
<tr>
<td>Eal_MWhyr</td>
<td>Electricity consumption due to appliances and lighting - Unit: MWh/yr</td>
</tr>
<tr>
<td>Eaux0_kW</td>
<td>Nominal Auxiliary electricity consumption. - Unit: [kW]</td>
</tr>
<tr>
<td>Eaux_MWhyr</td>
<td>Electricity consumption due to auxiliary equipment - Unit: MWh/yr</td>
</tr>
<tr>
<td>Edata0_kW</td>
<td>Nominal Data centre electricity consumption. - Unit: [kW]</td>
</tr>
<tr>
<td>Edata_MWhyr</td>
<td>Electricity consumption for data centers - Unit: MWh/yr</td>
</tr>
<tr>
<td>Epro0_kW</td>
<td>Nominal Industrial processes electricity consumption. - Unit: [kW]</td>
</tr>
<tr>
<td>Epro_MWhyr</td>
<td>Electricity supplied to industrial processes - Unit: MWh/yr</td>
</tr>
<tr>
<td>GFA_m2</td>
<td>Gross floor area - Unit: [m2]</td>
</tr>
<tr>
<td>GRIND0_kW</td>
<td>Nominal Grid electricity consumption - Unit: kW</td>
</tr>
<tr>
<td>GRIND_MWhyr</td>
<td>Grid electricity consumption - Unit: MWh/yr</td>
</tr>
<tr>
<td>NG_hs0_kW</td>
<td>Nominal NG requirement for space heating supply - Unit: kW</td>
</tr>
<tr>
<td>NG_hs_MWhyr</td>
<td>NG requirement for space heating supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>NG_ww0_kW</td>
<td>Nominal NG requirement for hotwater supply - Unit: kW</td>
</tr>
<tr>
<td>NG_ww_MWhyr</td>
<td>NG requirement for hotwater supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>Name</td>
<td>Unique building ID. It must start with a letter. - Unit: [-]</td>
</tr>
<tr>
<td>OIL_hs0_kW</td>
<td>Nominal OIL requirement for space heating supply - Unit: kW</td>
</tr>
<tr>
<td>OIL_hs_MWhyr</td>
<td>OIL requirement for space heating supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>OIL_ww0_kW</td>
<td>Nominal OIL requirement for hotwater supply - Unit: kW</td>
</tr>
<tr>
<td>OIL_ww_MWhyr</td>
<td>OIL requirement for hotwater supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>PV0_kW</td>
<td>Nominal PV electricity consumption - Unit: kW</td>
</tr>
<tr>
<td>PV_MWhyr</td>
<td>PV electricity consumption - Unit: MWh/yr</td>
</tr>
<tr>
<td>QC_sys0_kW</td>
<td>Nominal Total system cooling demand. - Unit: [kW]</td>
</tr>
<tr>
<td>QC_sys_MWhyr</td>
<td>Total system cooling demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>QH_sys0_kW</td>
<td>Nominal total building heating demand. - Unit: [kW]</td>
</tr>
<tr>
<td>QH_sys_MWhyr</td>
<td>Total building heating demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcdata0_kW</td>
<td>Nominal Data centre cooling demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcdata_MWhyr</td>
<td>Data centre cooling demand - Unit: [MWh/year]</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qcdata_sys0_kW</td>
<td>Nominal end-use data center cooling demand - Unit: kW</td>
</tr>
<tr>
<td>Qcdata_sys_MWhyr</td>
<td>End-use data center cooling demand - Unit: MWh/yr</td>
</tr>
<tr>
<td>Qcpro_sys0_kW</td>
<td>Nominal process cooling demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcpro_sys_MWhyr</td>
<td>Yearly processes cooling demand. - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcre0_kW</td>
<td>Nominal Refrigeration cooling demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcre_MWhyr</td>
<td>Refrigeration cooling demand for the system - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcre_sys0_kW</td>
<td>Nominal refrigeration cooling demand - Unit: kW</td>
</tr>
<tr>
<td>Qcre_sys_MWhyr</td>
<td>End-use refrigeration demand - Unit: MWh/yr</td>
</tr>
<tr>
<td>Qcs0_kW</td>
<td>Nominal Total cooling demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_MWhyr</td>
<td>Total cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_dis_ls0_kW</td>
<td>Nominal Cool distribution losses. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_dis_ls_MWhyr</td>
<td>Cool distribution losses - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_em_ls0_kW</td>
<td>Nominal Cool emission losses. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_em_ls_MWhyr</td>
<td>Cool emission losses - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_lat_ahu0_kW</td>
<td>Nominal AHU latent cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_lat_ahu_MWhyr</td>
<td>AHU latent cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_lat_aru0_kW</td>
<td>Nominal ARU latent cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_lat_aru_MWhyr</td>
<td>ARU latent cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_lat_sys0_kW</td>
<td>Nominal System latent cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_lat_sys_MWhyr</td>
<td>System latent cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_sen_ahu0_kW</td>
<td>Nominal AHU system cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_sen_ahu_MWhyr</td>
<td>AHU system cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_sen_aru0_kW</td>
<td>Nominal ARU system cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_sen_aru_MWhyr</td>
<td>ARU system cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_sen_scu0_kW</td>
<td>Nominal SCU system cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_sen_scu_MWhyr</td>
<td>SCU system cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_sen_sys0_kW</td>
<td>Nominal Sensible system cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_sen_sys_MWhyr</td>
<td>Sensible system cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_sys0_kW</td>
<td>Nominal end-use space cooling demand - Unit: kW</td>
</tr>
<tr>
<td>Qcs_sys_MWhyr</td>
<td>End-use space cooling demand - Unit: MWh/yr</td>
</tr>
<tr>
<td>Qcs_sys_ahu0_kW</td>
<td>Nominal AHU system cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_sys_ahu_MWhyr</td>
<td>AHU system cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_sys_aru0_kW</td>
<td>Nominal ARU system cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_sys_aru_MWhyr</td>
<td>ARU system cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qcs_sys_scu0_kW</td>
<td>Nominal SCU system cool demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qcs_sys_scu_MWhyr</td>
<td>SCU system cool demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhpro_sys0_kW</td>
<td>Nominal process heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhpro_sys_MWhyr</td>
<td>Yearly processes heat demand. - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs0_kW</td>
<td>Nominal Total heating demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_MWhyr</td>
<td>Total heating demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_dis_ls0_kW</td>
<td>Nominal Heating system distribution losses. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_dis_ls_MWhyr</td>
<td>Heating system distribution losses - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_em_ls0_kW</td>
<td>Nominal Heating emission losses. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_em_ls_MWhyr</td>
<td>Heating system emission losses - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_lat_ahu0_kW</td>
<td>Nominal AHU latent heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_lat_ahu_MWhyr</td>
<td>AHU latent heat demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_lat_aru0_kW</td>
<td>Nominal ARU latent heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_lat_aru_MWhyr</td>
<td>ARU latent heat demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_lat_sys0_kW</td>
<td>Nominal System latent heat demand. - Unit: [kW]</td>
</tr>
</tbody>
</table>

Continued on next page
Table 106 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qhs_lat_sys_MWhyr</td>
<td>System latent heat demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_sen_ahu0_kW</td>
<td>Nominal AHU sensible heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_sen_ahu_MWhyr</td>
<td>AHU sensible heat demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_sen_aru0_kW</td>
<td>ARU sensible heat demand - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_sen_aru_MWhyr</td>
<td>ARU sensible heat demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_sen_shu0_kW</td>
<td>Nominal SHU sensible heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_sen_shu_MWhyr</td>
<td>SHU sensible heat demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_sen_sys0_kW</td>
<td>Nominal HVAC systems sensible heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_sen_sys_MWhyr</td>
<td>SHU sensible heat demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qhs_sys0_kW</td>
<td>Nominal end-use space heating demand - Unit: kW</td>
</tr>
<tr>
<td>Qhs_sys_MWhyr</td>
<td>End-use space heating demand - Unit: MWh/yr</td>
</tr>
<tr>
<td>Qhs_sys_ahu0_kW</td>
<td>Nominal AHU sensible heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_sys_aru0_kW</td>
<td>Nominal ARU sensible heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_sys_shu0_kW</td>
<td>Nominal SHU sensible heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qhs_sys_shu_MWhyr</td>
<td>SHU sensible heat demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qww0_kW</td>
<td>Nominal DHW heat demand. - Unit: [kW]</td>
</tr>
<tr>
<td>Qww_MWhyr</td>
<td>DHW heat demand - Unit: [MWh/year]</td>
</tr>
<tr>
<td>Qww_sys0_kW</td>
<td>Nominal end-use hotwater demand - Unit: kW</td>
</tr>
<tr>
<td>Qww_sys_MWhyr</td>
<td>End-use hotwater demand - Unit: MWh/yr</td>
</tr>
<tr>
<td>SOLAR_hs0_kW</td>
<td>Nominal solar thermal energy requirement for space heating supply - Unit: kW</td>
</tr>
<tr>
<td>SOLAR_hs_MWhyr</td>
<td>Solar thermal energy requirement for space heating supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>SOLAR_ww0_kW</td>
<td>Nominal solar thermal energy requirement for hotwater supply - Unit: kW</td>
</tr>
<tr>
<td>SOLAR_ww_MWhyr</td>
<td>Solar thermal energy requirement for hotwater supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>WOOD_hs0_kW</td>
<td>Nominal WOOD requirement for space heating supply - Unit: kW</td>
</tr>
<tr>
<td>WOOD_hs_MWhyr</td>
<td>WOOD requirement for space heating supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>WOOD_ww0_kW</td>
<td>Nominal WOOD requirement for hotwater supply - Unit: kW</td>
</tr>
<tr>
<td>WOOD_ww_MWhyr</td>
<td>WOOD requirement for hotwater supply - Unit: MWh/yr</td>
</tr>
<tr>
<td>people0</td>
<td>Nominal occupancy - Unit: [people]</td>
</tr>
</tbody>
</table>

6.2.47 get_weather_file

The following file is used by scripts: ['radiation', 'photovoltaic', 'photovoltaic-thermal', 'solar-collector', 'demand', 'thermal-network']
CHAPTER 7

Legal

7.1 License

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

7.1.1 for V0.3c


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.

7.1.2 for V0.1


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.

7.2 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.
8.1 The Configuration File

The City Energy Analyst uses a configuration file for storing user preferences. User preferences are inputs to simulation runs like what weather file to use, what scenario to use and script-specific inputs.

When you first run the `cea` tool (e.g., with `cea install-toolbox` during the installation process), the default configuration file is copied to your home folder. On Windows systems, the home folder is usually something like `C:\Users\YourUserName`, so the configuration file would be stored in `C:\Users\michelle\cea.config`, assuming that your username is `michelle`.

8.1.1 Setting values in the configuration file

The most important values to set when working with the CEA are probably those under the `[general]` section, specifically `scenario`, and `weather`.

Open the `cea.config` file with a text editor (`notebook.exe` will do just fine) and update the values.

**Note:** We expect to implement an editor for the configuration file soon.

8.1.2 The configuration file and the command line interface

When you run the CEA from the command line (with the `cea` command), then the values to use as inputs to the scripts are taken from the configuration file. You can override each value by adding it as a parameter to the `cea` command, using the syntax `-- + parameter-name + "" + value`. Example:

```
$ cea demand --scenario C:\scenario\baseline --weather Brussels
```
8.1.3 The configuration file and the ArcGIS interface

The values in the configuration file are used as the default values when you open up a cea tool in the ArcGIS interface.

8.2 Configuration File Details

Let’s explore the details of how the configuration file works!

The configuration file edited by the user (~/cea.config) is only the tip of the iceberg, resting on a foundation of the default configuration file default.config file and the class cea.config.Configuration, which reads in the default configuration file as well as the user configuration file and makes those parameters available to the scripts. Each script is provided with an instance of cea.config.Configuration called config.

Each parameter is defined in a section. Each parameter has a type, which specifies the range of values allowed for that parameter and also how to read and write them to the configuration file.

Access to parameters through the config variable happens by section. Since all section names and parameter names in the configuration file follow the kebab-case naming convention, and these are not valid python identifiers, a translation is made to the snake_case naming convention: All hyphens (−) are replaced by underscores (_).

The syntax is simple:

```
"config." + [section] + "." + parameter
```

The section name is optional for the section general, so config.general.scenario refers to the same parameter as config.scenario. Note that these parameters can also be set:

```
config.scenario = r'C:\hoenggerberg\baseline'
```

If you want to persist these changes to disk, you need to explicitly save them with `cea.config.Configuration.save()`.

**Note:** It is a bad idea to have multiple instances of `cea.config.Configuration`, as if one part of a script changes a parameter, this will not be reflected in the other instances. Each CEA script accepts a config argument to it’s main function and should only use that.
8.2.1 Overview

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Section</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>default_config</td>
<td>name</td>
<td>name</td>
</tr>
<tr>
<td>user_config</td>
<td>help</td>
<td>help</td>
</tr>
<tr>
<td>sections</td>
<td>category</td>
<td>category</td>
</tr>
<tr>
<td></td>
<td>section</td>
<td>section</td>
</tr>
<tr>
<td></td>
<td>config</td>
<td>config</td>
</tr>
<tr>
<td>apply_command_line_args()</td>
<td></td>
<td><strong>getattr</strong>()</td>
</tr>
<tr>
<td>save()</td>
<td><strong>setattr</strong>()</td>
<td><strong>setattr</strong>()</td>
</tr>
</tbody>
</table>

8.2.2 Initialization of the config object

An instance of `cea.config.Configuration` is created with an optional `config_file` parameter that specifies the configuration file to load as the user configuration file. This defaults to `~/cea.config`. This file is parsed as a `ConfigParser.SafeConfigParser`, using the default configuration as a backup for the values and stored in the attribute `user_config`. Another `ConfigParser.SafeConfigParser` is created for the default configuration and stored in the attribute `default_config`.

Next, the `default_config` is used to create a dictionary of `cea.config.Section` objects and each section is populated with a dictionary of `cea.config.Parameter` instances. The default configuration file lists not only each parameter, but additional keys for each parameter as well. Example:

```
[general]
scenario = C:\reference-case-open\baseline
scenario.type = PathParameter
scenario.help = Path to the scenario to run
```

Using this information, the parameter `general:scenario` is assigned a default value of `C:\reference-case-open\baseline`, is represented by a subtype of `cea.config.Parameter` called `cea.config.PathParameter` and has a help text “Path to the scenario to run” - which is stored in the help attribute of the parameter object.

Some subclasses of `cea.config.Parameter` have additional configuration, like the `cea.config.ChoiceParameter`:

```
[data-helper]
region = CH
region.type = ChoiceParameter
region.choices = CH SIN custom
region.help = The region to use for the databases (either CH or SIN) - set to "custom" if you want to edit them
```

When the `config` instance is creating the parameters, each parameter object is given a chance to initialize itself with a call to `cea.config.Parameter.initialize(parser)` with `parser` set to the `default_config`. Subclasses of `Parameter` can override this method to read this additional configuration.

8.2. Configuration File Details
8.2.3 How a value is read from the config file

When a script does something like `config.general.weather`, the `config.sections` dictionary is checked for the section named `general` and the `parameters` dictionary in that section is checked for a parameter named `weather`. The `cea.config.Parameter.get()` method is called on that parameter and the result of this call is returned.

Based on the default configuration file, this is defined as:

```
[general]
weather = Zug-inducity_1990_2010_TMY
weather.type = WeatherPathParameter
weather.help = either a full path to a weather file or the name of one of the weather files shipped with the CEA
```

So the parameter is of type `cea.config.WeatherPathParameter`.

Inside the `cea.config.Parameter.get()` method, a call is made to `cea.config.Parameter.decode()`, passing in the value read from the user configuration file. Subclasses of `Parameter` specify how to encode and decode values to the configuration file. The semantics are:

- `decode` takes a string from a configuration file (or from the command line) and returns a typed value (e.g. a `bool` if the parameter type is `cea.config.BooleanParameter`).
- `encode` takes a typed value (e.g. a boolean value) and encodes it to a string that can be stored in the configuration file.

In the case of `cea.config.WeatherPathParameter`, `decode` will ensure that the path to the weather file exists and, if just the name of a weather file in the CEA weather file database is returned, resolves that to the full path to that file. Hence, on my system, the value of `config.weather` is `C:\Users\darthoma\Documents\GitHub\CityEnergyAnalyst\cea\databases\weather\Zurich.epw`.

8.2.4 How a value is saved to the config file

The mechanism for saving a value to the config file works similarly: `cea.config.Parameter.set()` is called, which in turn calls `cea.config.Parameter.encode()` - subclasses can override this to provide type specific behaviour.

8.2.5 How to create new parameter types

Steps:

1. subclass `cea.config.Parameter`
2. optional: override `initialize` to settings
3. optional: override `encode` to format the parameter value as a string
4. optional: override `decode` to read the parameter value from a string

Check the existing parameter types for ideas!

8.3 How to set up the Jenkins server on a new PC
There are a few steps to take to setting up a Jenkins server:

- installation of some prerequisites
- installation of Jenkins
- installation of a tunnel to the Jenkins server
- global configuration of Jenkins
- configuration of the Jenkins items - cea test for new pull requests - cea test for merges to master

8.3.1 Installation of some prerequisites

You will need to install these softwares:

- Miniconda2 (Python 2.7, 64-bit, download here: https://conda.io/miniconda.html)
  - this setup expects you installed Miniconda with the “Just for me” option. You might need to change some paths along the way if you install for all users.
  - to the default folder (%USERPROFILE%Miniconda2),
  - you don’t need to add it to the PATH environment variable
  - I registered it as the default Python 2.7 (but I don’t think that is necessary)
- git (I think any version will do, make sure git.exe is in your PATH by opening a command prompt and typing git --version)

8.3.2 Installation of Jenkins

- Download & install jenkins from https://jenkins.io - LTS version Jenkins 2.60.3 for Windows - just double click the installer, next, next, next (all default values) - set jenkins service to use local user
  - Open up the Services Manager (search for “Services” in the Windows menu)
  - locate and open the “Jenkins” service
  - make sure the Startup type is set to “Automatic” so the Jenkins starts up again after reboots
  - on the tab “Log On”, select “This account” instead of “Local System account” and enter in your credentials
  - this will allow the Jenkins to have access to your user profile. You can create an account just for this service and use that for the rest of this guide.
- open browser to http://localhost:8080
  - follow instructions to enter initial admin password
  - click “install suggested plugins”
  - create first admin user
    - Username: cea
8.3.3 Installation of a tunnel to the Jenkins server

This guide assumes you’re running the Jenkins on a Windows PC inside a corporate network. We use the ngrok service to tunnel webhooks triggered by GitHub back to the Jenkins server.

- download ngrok for Windows (https://ngrok.com/download)
- extract ngrok.exe to %PROGRAMDATA%\ceajenkins\ngrok.exe
  - (you might need to create the folder ceajenkins first)
- create a file ngrok.yml in the folder %PROGRAMDATA%\ceajenkins with the following contents:

```yaml
authtoken: Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 tunnels:
  ceajenkins:
    proto: http
    addr: 8080
    subdomain: ceajenkins
```
  - (replace the authtoken variable with the authtoken obtained from ngrok)
- test it with this command: %PROGRAMDATA%\ceajenkins\ngrok.exe start --config %PROGRAMDATA%\ceajenkins\ngrok.yml ceajenkins
  - you should now be able to access your Jenkins installation by going to https://ceajenkins.ngrok.io from any computer with access to the internet
  - press CTRL+C to shutdown the tunnel
- copy the CityEnergyAnalyst\bin\ceajenkins.py file to %PROGRAMDATA%\ceajenkins
- open the Anaconda Prompt and do conda create --name ceajenkins python=2.7 pywin32, then do activate ceajenkins
- open a new Anaconda Prompt with administrator rights (right click, then “Run as Administrator”)
- run python %PROGRAMDATA%\ceajenkins\ceajenkins\ceajenkins.py install
- in order for the service to find required DLL’s, ensure the PATH includes the following folders (use the windows search function to find the control panel item “Edit System Environment Variables”):
  - %USERPROFILE%\Miniconda2\envs\ceajenkins\%
  - %USERPROFILE%\Miniconda2\envs\ceajenkins\lib\site-packages\win32
- open the windows services panel (just search for “Services” in the windows menu)
  - locate “CEA Jenkins keepalive”, right click, “Properties”
  - set Startup type to “Automatic”
– set the account in the “Log On” tab to your user account (the one that you used to install all of the above stuff)
– start the service!
– you should now be able to access your Jenkins installation by going to https://ceajenkins.ngrok.io from any computer with access to the internet (test this)

8.3.4 Global configuration of Jenkins

Now that we have a tunnel set up, we can start configuring the Jenkins server, mainly following this guide:

• open browser to http://localhost:8080 and log in
• click “Manage Jenkins” and then “Configure System” - set “# of executors” to 1 (let’s just make it dead simple, no concurrency, less headache) - scroll to “GitHub” section - click “Advanced” - dropdown “Manage additional GitHub actions”, click “Convert login and password to token” - choose “From login and password”, enter GitHub user and password, click “Create token credentials” - Click “Add GitHub Server”
  – Name: (leave blank)
  – Credentials: (choose the GitHub credentials auto-generated for your username)
  – click “Test connection” - expect this message: “Credentials verified for user <username>”
  – check “Override Hook URL”
  – enter hook url https://ceajenkins.ngrok.io
  – click “Save”

Next, we make sure all the required Jenkins plugins are installed

• open browser to http://localhost:8080 and log in
• click “Manage Jenkins” and then “Manage Plugins”
  – install the following plugins / make sure they’re installed:
    * github-api plugin (https://wiki.jenkins-ci.org/display/JENKINS/GitHub+API+Plugin)
    * github plugin (https://wiki.jenkins-ci.org/display/JENKINS/GitHub+Plugin)
    * git plugin (https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin)
    * credentials plugin (https://wiki.jenkins-ci.org/display/JENKINS/Credentials+Plugin)
    * plain credentials plugin (https://wiki.jenkins-ci.org/display/JENKINS/Plain+Credentials+Plugin)
    * github pull request builder plugin (https://github.com/jenkinsci/ghprb-plugin)

Next, we configure the GitHub Pull Request Builder plugin, following the instructions here: https://github.com/jenkinsci/ghprb-plugin

• open browser to http://localhost:8080 and log in
• click “Manage Jenkins” and then “Configure System”
• scroll down to the “GitHub Pull Request Builder” section
  – leave the GitHub Server API URL: https://api.github.com
  – set the Jenkins URL override: https://ceajenkins.ngrok.io
  – leave the Shared secret: (bunch of *'s... idk...)
  – select the credentials (This should be the GitHub auto generated token credentials you created above)
– select Auto-manage webhooks
– set the Admin list to the two lines daren-thomas and JIMENOFONSECA

• click Save

8.3.5 Configuration of the Jenkins items

First, we configure a Jenkins item for pull requests:
• open browser to http://localhost:8080 and log in
• click “New Item”
• Enter an item name: run cea test for pull requests
  – Choose “Freestyle project”
  – Project name: “run cea test for pull requests”
  – Description: “Check out the CityEnergyAnalyst, create a conda environment for it and run cea test”
  – check “Discard old builds”
    • Strategy: “Log Rotation”
    • Max # of builds to keep: 10
  – check “GitHub project”
  – section “Source Code Management”:
    • select “Git”
    • Repository URL: https://github.com/architecture-building-systems/CityEnergyAnalyst.git
    • Credentials: (use the ones created above)
    • Branches to build: ${ghprbActualCommit}
  – section “Build Triggers”:
    • check “GitHub Pull Request Builder”
    • GitHub API credentials: choose your credentials from the list
    • check “Use github hooks for build triggering”
    • click “Advanced”
    • List of organizations. Their members will be whitelisted: architecture-building-systems
  – section “Build”
    • Execute Windows batch command: bin\ceatest.bat

Next, we configure a Jenkins item for merging to master:
• open browser to http://localhost:8080 and log in
• click “New Item”
• Enter an item name: run cea test on merge to master
  – Choose “Freestyle project”
– Project name: “run cea test on merge to master”
– Description: “Check out the CityEnergyAnalyst, create a conda environment for it and run `cea test --reference-case all`”
– check “Discard old builds”
  • Strategy: “Log Rotation”
  • Max # of builds to keep: 10
– check “GitHub project”
– section “Source Code Management”:  
  • select “Git”
  • Repository URL: https://github.com/architecture-building-systems/CityEnergyAnalyst.git
  • Credentials: (use the ones created above)
  • Refspec: +refs/heads/master:refs/remotes/origin/master
  • Branches to build: refs/heads/master
– section “Build Triggers”:
  • check “GitHub hook trigger for GITScm polling”
– section “Build”
  • Execute Windows batch command: `bin\ceatestall.bat`
  • open GitHub Integrations & services (https://github.com/architecture-building-systems/CityEnergyAnalyst/settings/installations)
    • dropdown “Add service”
      • select “Jenkins (GitHub plugin)”
      • enter Jenkins hook url: https://ceajenkins.ngrok.io
      • click “Add service” to save

### 8.4 How to add a heating/cooling system in CEA

#### 8.4.1 Step 0: Make an issue and create a branch
As this procedure requires adding scripts in CEA master, please make a branch before performing the changes.

#### 8.4.2 Step 1: Add the new system to the database

1. Open `cea/databases/systems/emission_systems.xls`
2. In the tab `heating` or `cooling`, add a row for the new system.
3. Specify the operating conditions of the new system, for cooling systems:
   • `code`: add a new code `Tx` that has not been used.
   • `Qcsmax_Wm2`: maximum cooling capacity of the system.
- dTcs_C:

For Air Handling Units (ahu), if applicable:
  - Tcs0_ahu_C: coolant (water) supply temperature at the primary side
  - dTcs0_ahu_C: temperature change of the coolant at the primary side
  - Tc_sup_air_ahu_C: air supply temperature from ahu to the room

For Air Recirculation Units (aru), if applicable:
  - Tcs0_aru_C: coolant (water) supply temperature at the primary side
  - dTcs0_aru_C: temperature change of the coolant at the primary side
  - Tc_sup_air_aru_C: air supply temperature from ahu to the room

For Sensible Cooling Units (scu), if applicable:
  - Tcs0_scu_C: coolant (water) supply temperature at the primary side
  - dTcs0_scu_C: temperature change of the coolant at the primary side

### 8.4.3 Step 2: Add the new system to the options

1. Go to script `cea/demand/control_heating_cooling_systems.py`
2. Add the code of the new systems (Tx) to function `has_cooling_systems` or `has_heating_systems`
3. Add a new function that check the type of the system, similar to `has_3for2_cooling_systems`

### 8.4.4 Step 3: Add a new function to model new technologies

Currently, there are models for AHU, ARU, SCU running with heating/cooling coil. If the new systems is utilizing different technologies, the models should be added to `airconditioning_model.py`.

### 8.4.5 Step 3: Add a new function to calculate cooling/heating loads

1. Go to script `cea/demand/hourly_procedure_heating_cooling_system_load.py`
2. Add a new function to set up the calculation procedure for cooling/heating loads, similar to `calc_cool_loads_3for2`

### 8.4.6 Step 4: Add distribution losses

1. Go to `cea/demand/sensible_loads.py`
2. Update `calc_Qhs_Qcs_loss`

### 8.4.7 Step 5: Calculate temperature and mass flow primary supply systems

1. Go to `cea/demand/sensible_loads.py`
2. Update `calc_temperatures_emission_systems`
8.4.8 Step 6: Calculate auxiliary electricity

calc_Eauxf/cs/dis calc_Eauxf_hs/dis

8.5 Data description for thermal_network_matrix.py

Contents

- Data description for thermal_network_matrix.py
  - ThermalNetwork.buildings_demands
  - ThermalNetwork.substations_HEX Specs
  - ThermalNetwork.t_target_supply_C
  - T_substation_supply_K
  - ThermalNetwork.t_target_supply_df
  - ThermalNetwork.all_nodes_df
  - ThermalNetwork.edge_df
  - ThermalNetwork.edge_node_df
  - ThermalNetwork.edge_mass_flow_df
  - ThermalNetwork.node_mass_flow_df
  - T_return_all
  - mdot_all
  - required_flow_rate_df
  - max_edge_mass_flow_df
  - ThermalNetwork.pipe_properties

- Description of DataFrames and Lists written to csv by the thermal_network_matrix.py file
  - ThermalNetwork.all_nodes_df
  - ThermalNetwork.edge_df
  - csv_outputs['T_supply_nodes']
  - csv_outputs['T_return_nodes']
  - csv_outputs['q_loss_supply_edges']
  - csv_outputs['plant_heat_requirement']
  - csv_outputs['pressure_nodes_supply']
  - csv_outputs['pressure_nodes_return']
  - csv_outputs['pressure_loss_system_Pa']
  - csv_outputs['pressure_loss_system_kW']
  - csv_outputs['pressure_loss_supply_kW']
This document describes the main variables used in the `cea.technologies.thermal_network.thermal_network_matrix` module.

The order of presentation follows the order of creating when running the script.

### 8.5.1 ThermalNetwork.buildings_demands

- **Type**: dictionary containing a DataFrame for each building

Description of each DataFrame:

- **shape**: $(8760, 25)$
- **Columns**:
  - Name
  - Ef_kWh
  - Qhsf_kWh
  - Qwwf_kWh
  - Qcsf_kWh
  - Qcsf_lat_kWh
  - Qcdataf_kWh
  - Qcref_kWh
  - mcphsf_kWperC
  - mcpcsf_kWperC
  - mcpwwf_kWperC
  - Twwf_sup_C
  - Twwf_re_C
  - Thsf_sup_C
  - Thsf_re_C
  - Tcsf_sup_C
  - Tcsf_re_C
  - Tcdataf_re_C
  - Tcdataf_sup_C
  - Tcref_re_C
  - Tcref_sup_C
  - Q_substation_heating
  - Q_substation_cooling
  - T_sup_target_DH


- T_sup_target_DC

Index Time steps 0-8759

created by
- ThermalNetwork_init_ (empty)
- substation_matrix.determine_building_supply_temperatures

passed to
- substation_matrix.substation_HEX_design_main (creating substations_HEX_specs)
- read_properties_from_buildings (creating t_target_supply_C)
- initial_diameter_guess
- substation_return_model_main
- hourly_thermal_calculation

8.5.2 ThermalNetwork.substations_HEX_specs

type DataFrame

shape (len(building_names), 6)

Columns
- HEX_area_SH
- HEX_area_DHW
- HEX_area_SC
- HEX_UA_SH
- HEX_UA_DHW
- HEX_UA_SC

Index building_names

created by
- ThermalNetwork_init_ (empty)
- substation_matrix.substation_HEX_design_main

passed to
- network_parameters (dictionary)
- initial_diameter_guess
- hourly_mass_flow_calculation
- substation_return_model_main
- hourly_thermal_calculation

8.5. Data description for thermal_network_matrix.py 79
8.5.3 ThermalNetwork.t_target_supply_C

type DataFrame
shape (8760, len(building_names))
Columns building_names
Index Timesteps 0-8759

created by
• ThermalNetwork_init_ (empty)
• read_properties_from_buildings

passed to
• write_substation_temperatures_to_nodes_df (creating t_target_supply_df),
• calc_max_edge_flowrate,
• initial_diameter_guess,
• hourly_mass_flow_calculation

8.5.4 T_substation_supply_K

type DataFrame
shape (1, len(building_names))
Columns building_names
Index ['T_supply']

created by
• hourly_mass_flow_calculation
• write_nodes_values_to_substations

passed to
• substation_return_model_main

8.5.5 ThermalNetwork.t_target_supply_df

type DataFrame
shape (8760, number_of_nodes)
Columns All Nodes ([NODE0, ...])
Index Timesteps 0-8759

created by
• ThermalNetwork_init_ (empty)
• write_substation_temperatures_to_nodes_df

passed to
8.5.6 **ThermalNetwork.all_nodes_df**

**type** DataFrame  
**shape** (number_of_nodes, 2)  

**Columns**
- Type  
- Building  

**Index** All Nodes ([NODE0, …])

**created by**
- ThermalNetwork._init_(empty)  
- get_thermal_network_from_shapefile

**passed to**
- write_substation_temperatures_to_nodes_df (creating t_target_supply_df)  
- network_parameters (dictionary)  
- initial_diameter_guess  
- hourly_mass_flow_calculation (creating required_flow_rate_df)  
- substation_return_model_main  
- calc_mass_flow_edges  
- hourly_thermal_calculation

8.5.7 **ThermalNetwork.edge_df**

**type** GeoDataFrame  
**shape**
- initially: (number_of_edges, 7),  
- later: (number_of_edges, 15),  
  - merge with ThermalNetwork.pipe_properties in thermal_network_main to store data and output together in one file  

**Columns**
- initially:  
  - Type_mat  
  - Pipe_DN  
  - geometry  
  - coordinates  
  - pipe length  
  - start node  
  - end node  
- later:
- Type_mat
- Pipe_DN_x
- geometry
- coordinates
- pipe length
- start node
- end node
- Pipe_DN_y
- D_ext_m
- D_int_m
- D_ins_m
- Vdot_min_m3s
- Vdot_max_m3s
- mdot_min_kgs
- mdot_max_kgs

**Index**  All Edges ([PIPE0, . . .])

**created by**

- ThermalNetwork_init_
- get_thermal_network_from_shapefile

**passed to**

- network_parameters (dictionary)
- initial_diameter_guess
- hourly_mass_flow_calculation
- substation_return_model_main
- hourly_thermal_calculation

### 8.5.8 ThermalNetwork.edge_node_df

**type**  DataFrame

**shape**  (number_of_nodes, number_of_edges)

**Columns**  All Edges ([PIPE0, . . .])

**Index**  All Nodes ([NODE0, . . .])

**created by**

- ThermalNetwork_init_ (empty)
- get_thermal_network_from_shapefile

**passed to**

- network_parameters (dictionary)
• initial_diameter_guess
• hourly_mass_flow_calculation
• substation_return_model_main
• calc_mass_flow_edges
• hourly_thermal_calculation

8.5.9 ThermalNetwork.edge_mass_flow_df

type DataFrame
shape (8760, number_of_edges)
Columns All Edges ([PIPE0, PIPE1, ..., PIPEn])
Index Timesteps 0-8759
created by
• ThermalNetwork_init_(empty)
• calc_max_edge_flowrate
• load_max_edge_flowrate_from_previous_run (read from csv)
passed to
• network_parameters (dictionary)
• hourly_mass_flow_calculation
• hourly_thermal_calculation

8.5.10 ThermalNetwork.node_mass_flow_df

type DataFrame
shape (8760, number_of_nodes)
Columns All Edges ([NODE0, NODE1, ..., NODEn])
Index Timesteps 0-8759
created by
• ThermalNetwork_init_(empty)
• calc_max_edge_flowrate

8.5.11 T_return_all

type DataFrame
shape (1, len(building_names))
Columns building_names
Index 0

8.5. Data description for thermal_network_matrix.py
created by hourly_mass_flow_calculation
passed to

8.5.12 mdot_all

type DataFrame
shape (1, len(building_names))
Columns building_names
Index 0
created by hourly_mass_flow_calculation
passed to write_substation_values_to_nodes_df (creating required_flow_rate_df)

8.5.13 required_flow_rate_df

type DataFrame
shape (1, number_of_nodes)
Columns All Nodes ([NODE0, ...])
Index 0
created by write_substation_values_to_nodes_df
passed to calc_mass_flow_edges

8.5.14 max_edge_mass_flow_df

type DataFrame
shape (1, number_of_edges)
Columns All Edges ([PIPE0, ...])
Index 0
created by calc_max_edge_flowrate
passed to max_edge_mass_flow_df_kgs (rename when exiting calc_max_edge_flowrate function)

8.5.15 ThermalNetwork.pipe_properties

type DataFrame
shape (8, number_of_edges)
Columns All Edges ([PIPE0, ...])
Index
  • Pipe_DN
  • D_ext_m
  • D_int_m
8.6 Description of DataFrames and Lists written to csv by the thermal_network_matrix.py file

sorted in order of creation in the script

8.6.1 ThermalNetwork.all_nodes_df

type DataFrame
shape (number_of_nodes, 2)
Columns
  • Type
  • Building
Index All Nodes ([NODE0, ...])

8.6.2 ThermalNetwork.edge_df

type GeoDataFrame
shape (number_of_edges, 15),
Columns
  • Type_mat
  • Pipe_DN_x
  • geometry
  • coordinates
  • pipe length
  • start node
• end node
• Pipe_DN_y
• D_ext_m
• D_int_m
• D_ins_m
• Vdot_min_m3s
• Vdot_max_m3s
• mdot_min_kgs
• mdot_max_kgs

Index  All Edges ([PIPE0, ...])

8.6.3 csv_outputs[‘T_supply_nodes’]

type  DataFrame
shape  (8760, number_of_nodes),
Columns  All Nodes ([NODE0, ...])
Index  Timesteps 0-8759

8.6.4 csv_outputs[‘T_return_nodes’]

type  DataFrame
shape  (8760, number_of_nodes),
Columns  All Nodes ([NODE0, ...])
Index  Timesteps 0-8759

8.6.5 csv_outputs[‘q_loss_supply_edges’]

type  DataFrame
shape  (8760, number_of_edges),
Columns  All Edges ([PIPE0, ...])
Index  Timesteps 0-8759

8.6.6 csv_outputs[‘plant_heat_requirement’]

type  DataFrame
shape  (8760, number_of_plants),
Columns  Plant Buildings
Index  Timesteps 0-8759
8.6.7 csv_outputs['pressure_nodes_supply']

- **type**: DataFrame
- **shape**: (8760, number_of_nodes),
- **Columns**: All Nodes ([NODE0, ...])
- **Index**: Timesteps 0-8759

8.6.8 csv_outputs['pressure_nodes_return']

- **type**: DataFrame
- **shape**: (8760, number_of_nodes),
- **Columns**: All Nodes ([NODE0, ...])
- **Index**: Timesteps 0-8759

8.6.9 csv_outputs['pressure_loss_system_PA']

- **type**: DataFrame
- **shape**: (8760, 3),
- **Columns**
  - pressure_loss_supply_Pa
  - pressure_loss_return_Pa
  - pressure_loss_total_Pa
- **Index**: Timesteps 0-8759

8.6.10 csv_outputs['pressure_loss_system_kW']

- **type**: DataFrame
- **shape**: (8760, 3),
- **Columns**
  - pressure_loss_supply_kW
  - pressure_loss_return_kW
  - pressure_loss_total_kW
- **Index**: Timesteps 0-8759

8.6.11 csv_outputs['pressure_loss_supply_kW']

- **type**: DataFrame
- **shape**: (8760, number_of_edges),
- **Columns**: All Edges ([PIPE0, ...])
- **Index**: Timesteps 0-8759

8.6. Description of DataFrames and Lists written to csv by the thermal_network_matrix.py file
8.6.12 csv_outputs['q_loss_system']

- **type**: DataFrame
- **shape**: (8760, 3),
- **Columns**: 0
- **Index**: Timesteps 0-8759

8.6.13 csv_outputs['edge_mass_flows']

- **type**: DataFrame
- **shape**: (8760, number_of_edges),
- **Columns**: All Edges ([PIPE0, ...])
- **Index**: Timesteps 0-8759

8.7 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 / 10.5
- **Grasshopper** - the CEA implements a Rhino/Grasshopper component for running the CEA scripts
- **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.

8.7.1 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 SCRIPT [OPTIONS]
   to run a specific script
usage: cea --help SCRIPT
   to get additional help specific to a script
SCRIPT can be one of: benchmark-graphs, compile, data-helper,
dbf-to-excel, demand, demand-graphs, embodied-energy, emissions,
excel-to-dbf, extract-reference-case, install-toolbox,
latitude, list-demand-graphs-fields, locate, longitude, mobility,
operation-costs, photovoltaic, photovoltaic-thermal, read-config, read-config-
section,
retrofit-potential, scenario-plots, sensitivity-demand-analyze,
sensitivity-demand-samples, sensitivity-demand-simulate,
solar-collector, test, weather-files, weather-path, write-config
```
All scripts use the configuration file as the default source of parameters. See the Configuration File Details for information on the configuration file.

The parameters in the configuration file relevant to a script can be overridden. To see which parameters are used by a certain script, use the syntax `cea --help SCRIPT:

```bash
> cea --help data-helper

building properties algorithm

OPTIONS for data-helper:
--scenario: C:/reference-case-open/baseline
  Path to the scenario to run
--archetypes: ['comfort', 'architecture', 'HVAC', 'internal-loads']
  List of archetypes to process
```

This displays some documentation on the script as well as a list of parameters, their default values and a description of the parameter. Using this information, the `data-helper` script can be run like this:

```bash
> cea data-helper --scenario C:/reference-case-open/scenario1 --archetypes HVAC
```

**Note:** All options are optional and have default values as defined in the configuration file!

### 8.7.2 ArcGIS interface

Use the following command to install the interface for ArcScene 10.4 / 10.5:

```bash
> cea install-toolbox
```

Start ArcScene and check Toolboxes/My Toolboxes in the Catalog: You should see a toolbox called “City Energy Analyst.pyt” with all the interfaces.

### 8.7.3 Rhino / Grasshopper interface

Use the following command to install the interface for Rhino 5.0 / Grasshopper:

```bash
> cea install-grasshopper
```

This will install a Grasshopper component in the “CEA” section of the Grasshopper tool menu that can be used to execute CEA scripts. It works similar to the `cea` CLI command described above: Input the name of the script to run (e.g. “demand”) and an (optional) list of parameters. The list of valid parameters to use is the same as for the CLI interface. When specifying parameters for the “args” input to the component, use this syntax:

```python
scenario = {general:scenario}/../scenario1
weather = Zug
```

**Note:** You can use references to parameters in the configuration file using the `{SECTION:PARAMETER}` syntax as in the above example.
8.7.4 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

8.7.5 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

8.8 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:

8.8.1 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 \( \text{tsd} \) 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 \( \text{tsd} \) dictionary are the names of the state variables of the simulation.

The demand calculation also uses a variable \( \text{bpr} \) to store building properties of a single building.
8.8.2 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.

8.8.3 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.

8.8.4 “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.
9.1 cea package

exception cea.ConfigError
    Bases: exceptions.Exception
    Raised when the configuration of a tool contains some invalid values.
    \[ rc = 100 \]

exception cea.CustomDatabaseNotFoundException
    Bases: exceptions.Exception
    Raised when the InputLocator can’t find a user-provided database (region==’custom’) 
    \[ rc = 101 \]

exception cea.InvalidOccupancyNameException
    Bases: exceptions.Exception
    Raised when the occupancy.dbf has an invalid / unknown occupancy column
    \[ rc = 104 \]

exception cea.MissingInputDataException
    Bases: exceptions.Exception
    Raised when a script can’t run because some information is missing
    \[ rc = 103 \]

exception cea.ScriptNotFoundException
    Bases: exceptions.Exception
    Raised when an invalid script name is used.
    \[ rc = 102 \]

cea.suppress_3rd_party_debug_loggers()
    set logging level to WARN for fiona and shapely and others
9.1.1 Subpackages

cea.analysis package

Subpackages

cea.analysis.costs package

Submodules

cea.analysis.costs.operation_costs module

costs according to supply systems

```python
ccea.analysis.costs.operation_costs.main(config)
ccea.analysis.costs.operation_costs.operation_costs(locator, config)
```

cea.analysis.lca package

Submodules

cea.analysis.lca.embodied module

Embodied energy and related grey emissions model algorithm

```python
ccea.analysis.lca.embodied.calc_code(code1, code2, code3, code4)
ccea.analysis.lca.embodied.calc_if_existing(x, y)
```

Function to verify if one value is greater than or equal to another (then return 1) or not (return 0). This is used to verify whether a building’s construction or retrofits happened more than 60 years before the year to calculate. Since the embodied energy and emissions are calculated over 60 years, if the year of calculation is more than 60 years after construction, the results will be 0.

**Parameters**

- **\(x\) (long)** – Number of years since construction/retrofit
- **\(y\) (int)** – Number of years over which the embodied energy/emissions calculation is carried out (i.e., 60)

**Return value** 1 if \(x \leq y\); 0 otherwise

**Rtype value** int

```python
ccea.analysis.lca.embodied.calculate_contributions(archetype, cat_df, config, locator,
year_to_calculate, total_column, specific_column)
```

Calculate the embodied energy/emissions for each building based on their construction year, and the area and renovation year of each building component.

**Parameters**

- **archetype (str)** – String that defines whether the ‘EMBODIED_ENERGY’ or ‘EMBODIED_EMISSIONS’ are being calculated.
• cat_df (DataFrame) – DataFrame with joined data of all categories for each building, that is: occupancy, age, geometry, architecture, building component area, construction category and renovation category for each building component
• locator (InputLocator) – an InputLocator instance set to the scenario to work on
• year_to_calculate (int) – year in which the calculation is done; since the embodied energy and emissions are calculated over 60 years, if the year of calculation is more than 60 years after construction, the results will be 0
• total_column (str) – label for the column with the total results (e.g., ‘GEN_GJ’)
• specific_column (str) – label for the column with the results per square meter (e.g., ‘GEN_MJm2’)

Return result DataFrame with the calculation results (i.e., the total and specific embodied energy or emissions for each building)

Rtype result DataFrame

cea.analysis.lca.embodied.lca_embodied(year_to_calculate, locator, config)
Algorithm to calculate the embodied emissions and non-renewable primary energy of buildings according to the method of [Fonseca et al., 2015] and [Thoma et al., 2014]. The calculation method assumes a 60 year payoff for the embodied energy and emissions of a building, after which both values become zero.

The results are provided in total as well as per square meter:
• embodied non-renewable primary energy: E_nre_pen_GJ and E_nre_pen_MJm2
• embodied greenhouse gas emissions: E_ghg_ton and E_ghg_kgm2

As part of the algorithm, the following files are read from InputLocator:
• architecture.shp: shapefile with the architecture of each building locator.get_building_architecture()
• occupancy.shp: shapefile with the occupancy types of each building locator.get_building_occupancy()
• age.shp: shapefile with the age and retrofit date of each building locator.get_building_age()
• zone.shp: shapefile with the geometry of each building in the zone of study locator.get_zone_geometry()
• Archetypes_properties: csv file with the database of archetypes including embodied energy and emissions locator.get_archetypes_properties()

As a result, the following file is created:
• Total_LCA_embodied: .csv csv file of yearly primary energy and grey emissions per building stored in locator.get_lca_embodied()

Parameters
• year_to_calculate (int) – year between 1900 and 2100 indicating when embodied energy is evaluated to account for emissions already offset from building construction and retrofits more than 60 years ago.
• locator (InputLocator) – an instance of InputLocator set to the scenario

Returns This function does not return anything

Return type NoneType

Files read / written from InputLocator:
get_building_architecture  get_building_occupancy  get_building_age  get_zone_geometry
get_archetypes_embodied_energy  get_archetypes_embodied_emissions

**path_LCA_embodied_energy**: path to database of archetypes embodied energy file
  Archetypes_embodied_energy.csv

**path_LCA_embodied_emissions**: path to database of archetypes grey emissions file
  Archetypes_embodied_emissions.csv

**path_age_shp**: string path to building_age.shp

**path_occupancy_shp**: path to building_occupancy.shp

**path_geometry_shp**: path to building_geometry.shp

**path_architecture_shp**: path to building_architecture.shp

**path_results** [string] path to demand results folder emissions

**cea.analysis.lca.embodied.main**(config)

**cea.analysis.lca.main module**

Emissions analysis (LCA) This script is used to calculate the LCA

**cea.analysis.lca.main.**

emissions_main(locator, config)

**cea.analysis.lca.main** (config)

**cea.analysis.lca.mobility module**

Primary energy and CO2 emissions model algorithm for mobility.

**cea.analysis.lca.mobility.**

lca_mobility(locator, config)

Calculation of the primary energy and CO2 emissions for mobility in the area based on the present day values calculated for the 2000 Watt society target.

The current values for the Swiss case for each type of occupancy are based on the following sources:


Due to a lack of data, multiple values had to be assumed:

• ‘INDUSTRY’: assumed to be equal to the value for the use type ‘OFFICE’

• ‘HOSPITAL’: assumed to be equal to the value for the use type ‘HOTEL’

• ‘GYM’, ‘SWIMMING’: assumed to be equal to the value for use type ‘RETAIL’

• ‘SERVERROOM’, ‘COOLROOM’: assumed negligible

The following file is created as a side effect by this script:

• total_LCA_mobility (.csv) csv file of yearly non-renewable primary energy demand and emissions due to mobility for each building

Additional references:

• [SIA_Effizienzpfad_2011]
Parameters **locator** (*InputLocator*) – an *InputLocator* instance set to the scenario to work on

```python
ccea.analysis.lca.mobility.main(config)
```

### cea-analysis-lca-operation module

Non-renewable primary energy and CO2 emissions model algorithm for building operation

```python
ccea.analysis.lca.operation.lca_operation(locator)
```

Algorithm to calculate the primary energy and CO2 emissions of buildings according to the method used in the integrated model of [Fonseca-Schlueter-2015] and the performance factors of [ecobau.ch].

Parameters **locator** (*InputLocator*) – an *InputLocator* instance set to the scenario to work on

The following file is created by this script:

- **total_LCA_operation:.csv** csv file of yearly non-renewable primary energy demand and CO2 emissions per building for all energy services (i.e., heating, hot water, cooling, electricity) both total and per square meter

**Returns** This function does not return anything

**Return type** NoneType

```python
ccea.analysis.lca.operation.main(config)
```

### cea-analysis-multicriteria package

### Submodules

### cea-analysis-multicriteria-main module

Multi criteria decision analysis

```python
ccea.analysis.multicriteria.main.main(config)
```

```python
ccea.analysis.multicriteria.main.multi_criteria_main(locator, config)
```

```python
ccea.analysis.multicriteria.main.normalize_compiled_data(compiled_data_df)
```

```python
ccea.analysis.multicriteria.main.rank_normalized_data(compiled_data_df, config)
```

### cea-analysis-sensitivity package

Sensitivity analysis for demand_main.py

These scripts use the morris algorithm (morris 1991) (campolo 2011) and Sobol Algorithm Slalli 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.sensitivity.sensitivity_demand_analyze module

Analyze the results in the samples folder and write them out to an Excel file. This script assumes:

- a samples folder with the files `samples.npy` and `problem.pickle` was created with `sensitivity_demand_samples.py`
- all the results have been added to the samples folder in the format `result.%i.csv`, with `%i` replaced by the index into the samples array. Use the script `sensitivity_demand_simulate.py` to create the results.
- each result file has the same list of columns (the –output-parameters for the simulations were the same)
- the `analyze_sensitivity` function is called with the same method and arguments as the sampling routine (`sensitivity_demand_samples.py`).

```
cea.analysis.sensitivity.sensitivity_demand_analyze.analyze_sensitivity(samples_path, tempo-
ral_scale)
```

Run the analysis for each output parameter. The exact function to use is selected by the `method` parameter: Use “morris” for `SALib.analyze.morris` and “sobol” for `SALib.analyze.sobol`.

The samples folder `samples_path` contains the results of running a simulation with `sensitivity_demand_simulate.py` for each sample generated by `sensitivity_demand_samples.py`.

It also contains the `samples.npy` file with input values for the samples. It also contains the `problem.pickle` file which contains information about the problem as well as parameters that were set in the `sensitivity_demand_samples.py` call (method, N and method-specific parameters)

Parameters

- `samples_path` (`str`) – the path to the samples folder as created by `sensitivity_demand_samples.py`
- `temporal_scale` (`str`) – The scale at which to do the analysis. Valid values: ['monthly', 'yearly']

```
cea.analysis.sensitivity.sensitivity_demand_analyze.main(config)
```

```
cea.analysis.sensitivity.sensitivity_demand_analyze.morris_analyze_function(problem, X, Y)
```

Use the SALib.analyze.morris method to analyze the simulation results.

Parameters

- `problem` – The definition of the problem statement as defined for the sampling method.
- `X` – the X parameter of the morris method (The NumPy matrix containing the model inputs)
- `Y` – The NumPy array containing the model outputs
- `parameters` – dictionary containing the key ‘num_levels’ that is passed on to the SALib.analyze.morris method parameter of the same name.

Returns returns the result of the SALib.analyze.sobol method (from the documentation: a dictionary with keys `mu`, `mu_star`, `sigma`, and `mu_star_conf`, where each entry is a list of size D (the number of eters) containing the indices in the same order as the parameter file.)
cea.analysis.sensitivity.sensitivity_demand_analyze.read_results(samples_folder, samples_count, output_parameter, temporal_scale, month=0)

Read each results.%i.csv file from the samples folder into a DataFrame and return them as a list. Each such csv file has a column for each output parameter specified for the simulation runs and a row for each building.

Parameters

- **samples_folder (str)** – path to the samples folder containing the results files of the simulation runs
- **samples_count (str)** – the total number of results.%i.csv files in the samples folder. Use sensitivity_demand_count.py to calculate this for a given samples folder.
- **output_parameter** – output parameter of the simulation e.g., Qhsf, Ef

Returns matrix of length mxn where m: samples_count, n: buildings, content: result for output_parameter

Return type list of DataFrame

cea.analysis.sensitivity.sensitivity_demand_analyze.sobol_analyze_function(problem, Y)

Use the SALib.analyze.sobol method to analyze the simulation results.

Parameters

- **problem** – The definition of the problem statement as defined for the sampling method.
- **_** – placeholder for the X parameter of the morris method not used for sobol
- **Y** – The NumPy array containing the model outputs
- **parameters** – dictionary containing the key `calc_second_order` with a bool value to be passed to the SALib.analyze.morris calc_second_order parameter.

Returns returns the result of the SALib.analyze.sobol method (from the documentation: a dictionary with keys S1, S1_conf, ST, and ST_conf, where each entry is a list of size D (the number of parameters) containing the indices in the same order as the parameter file. If calc_second_order is True, the dictionary also contains keys S2 and S2_conf.)

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**

Create a list of samples in a specified folder as input for the demand sensitivity analysis.

This script creates:

- a samples folder with the files `samples.npy` and `problem.pickle` to be used by the scripts sensitivity_demand_count.py, sensitivity_demand_simulate.py and sensitivity_demand_analyze.py.

The file `samples.npy` is a NumPy array of samples to simulate, as generated by either the morris or the sobol sampler. Each sample is a row in the array and each row consists of a list of parameter values to use for the simulation. The parameters correspond to the parameter names defined by the `variable_groups` input. `variable_groups` refers to worksheets in the uncertainty db, an Excel file in cea/databases/CH/Uncertainty/uncertainty_distributions.xls, which specifies row-by-row the variables to sample and their distribution parameters.

The file `problem.pickle` is a python dictionary that is saved using the standard `pickle` module and contains the following data:

- `num_vars`: int, the number of variables being analyzed.
- `names`: list of str, the variable names in the same order as the values in each sample row. Used to apply the sample values to the input data using the override mechanism. See: - cea.analysis.sensitivity.sensitivity_demand_simulate.apply_sample_parameters (write overrides) - cea.demand.thermal_loads.BuildingProperties#__init__ (read overrides)
- `bounds`: list of tuple(min, max), lower and upper bounds for each variable to sample (only used by the sampler)
- `groups`: None (currently not used)

**cea.analysis.sensitivity.sensitivity_demand_samples.create_demand_samples** *(locator, method, num_samples, variable_groups, sampler_parameters)*

Create the samples to simulate using the specified method *(method)*, the sampling method parameter N *(num_samples)* and any additional sampling method-specific parameters specified in `sampler_parameters` for each variable defined in the uncertainty database worksheets referenced in `variable_groups`.

Parameters

- `method` *(str)* – The method to use. Valid values are ‘morris’ (default) and ‘sobol’.
- `num_samples` *(int)* – The parameter N for the sampling methods (sobol defines this as “The number of samples to generate”, but in reality, for both methods, the actual number of samples is a multiple of `num_samples`).
- `sampler_parameters`(dict of (str, _)) – additional, sampler-specific parameters. For `method='morris'` these are: [num_levels], for `method='sobol'` these are: [calc_second_order]
- `variable_groups` – list of names of groups of variables to analyse. Possible values are: ‘THERMAL’, ‘ARCHITECTURE’, ‘INDOOR_COMFORT’, ‘INTERNAL_LOADS’. This list links to the probability density functions of the variables contained in locator.get_uncertainty_db() and refers to the Excel worksheet names.
Returns (samples, problem) - samples is a list of configurations for each simulation to run, a configuration being a list of values for each variable in the problem. The problem is a dictionary with the keys 'num_vars', 'names' and 'bounds' and describes the variables being sampled: 'names' is a list of variable names of length 'num_vars' and 'bounds' is a list of tuples(lower-bound, upper-bound) for each of these variables. Further, the keys 'N' (num_samples) and 'method' (method) are set and the sampler_parameters are also added to problem.

cea.analysis.sensitivity.sensitivity_demand_samples.main(config)

cea.analysis.sensitivity.sensitivity_demand_samples.sampler(method, problem, num_samples, sampler_parameters)

Run the sampler specified by method and return the results.

Parameters

• method (str) – The method to use. Valid values: 'morris', 'sobol'

• problem (dict of (str, _)) – The problem dictionary as required by the sampling methods.

• num_samples (int) – The parameter N to the sampler methods of sobol and morris. NOTE: This is not the number of samples produced, but relates to the total number of samples produced in a manner dependent on the sampler method used. See the documentation of sobol and morris in the SALib for more information.

• sampler_parameters (dict of (str, _)) – Sampler method-specific parameters to be passed to the sampler function as keyword arguments.

Returns The list of samples generated as a NumPy array with one row per sample and each row containing one value for each variable name in problem['names'].

Return type ndarray

cea.analysis.sensitivity.sensitivity_demand_simulate module

Simulate a single sample or a batch of samples from the samples folder using the demand script.

The script sensitivity_demand_samples.py creates a samples folder containing a list of samples stored in the NumPy array samples.npy. Each sample is a list of parameter values to set for a list of variables (the names of the variables are stored in the file problem.pickle in the samples folder).

This script runs the samples --sample-index through --sample-index + --number-of-samples and writes the results out to the samples folder as files of the form results.$i.csv (with $i set to the index into the samples array).

cea.analysis.sensitivity.sensitivity_demand_simulate.apply_sample_parameters(sample_index, samples_path, scenario_path, simulation_path)

Copy the scenario from the scenario_path to the simulation_path. Patch the parameters from the problem statement. Return an InputLocator implementation that can be used to simulate the demand of the resulting scenario.

The simulation_path is modified by the demand calculation. For the purposes of the sensitivity analysis, these changes can be viewed as temporary and deleted / overwritten after each simulation.
Parameters

- **sample_index** *(int)* – zero-based index into the samples list, which is read from the file
  `${samples_path}/samples.npy`
- **samples_path** *(str)* – path to the pre-calculated samples and problem statement (created by `sensitivity_demand_samples.py`)
- **scenario_path** *(str)* – path to the scenario template
- **simulation_path** *(str)* – a (temporary) path for simulating a scenario that has been
  patched with a sample

NOTE: When simulating in parallel, special care must be taken that each process has a unique `simulation_path` value. For the Euler cluster, this is solved by ensuring the simulation is done with `multiprocessing = False` and setting the `simulation_path` to the special folder `$TMPDIR` that is set to a local scratch folder for each job by the job scheduler of the Euler cluster. Other setups will need to adopt an equivalent strategy.

Returns

InputLocator that can be used to simulate the demand in the `simulation_path`

```python
ccea.analysis.sensitivity.sensitivity_demand_simulate.main(config)
```

Parse the arguments passed to the script and run `simulate_demand_sample` for each sample in the current batch.

The current batch is the list of samples starting at `--sample-index`, up until `--sample-index + --number-of-simulations`.

Run this script with the argument `--help` to get an overview of the parameters.

```python
ccea.analysis.sensitivity.sensitivity_demand_simulate.read_problem(samples_path)
ccea.analysis.sensitivity.sensitivity_demand_simulate.read_samples(samples_path)
ccea.analysis.sensitivity.sensitivity_demand_simulate.simulate_demand_batch(sample_index, batch_size, samples_folder, scenario, simulation_folder, config, output_parameters)
```

Run the simulations for a whole batch of samples and write the results out to the samples folder.

Each simulation result is saved to the samples folder as `result.$i.csv` with `$i` representing the index into the
samples array.

Parameters

- **sample_index** *(int)* – The index into the first sample of the batch as defined in the
  `samples.npy` NumPy array in the samples folder.
- **batch_size** *(int)* – The number of simulations to perform, starting at `sample_index`. When
  computing on a cluster such as Euler, the batch size should be chosen to minimize overhead of starting up a node on the cluster. For our test purposes, the batch size 100 has proven to be a good default, but you might need to test this for your case.
- **samples_folder** *(str)* – The path to the samples folder, containing the `samples.npy` and
  `problem.pickle` files, as generated by the `sensitivity_demand_samples.py` script. This is
the folder where the output of the simulations is written to. It also contains information necessary for applying sample parameters to the input files.

- **scenario (str)** – The path to the scenario. This argument corresponds to the `scenario_path` parameter of the InputLocator constructor. For the sensitivity simulations, the scenario is used as a template that is copied to a simulation folder and then modified using the override mechanism to add the sample parameter values. See the function `apply_sample_parameters` for the details of copying the scenario to the simulation folder.

- **simulation_folder (str)** – The path to the folder to use for simulation. The scenario is copied to the simulation folder and updated with the sample parameter values using the function `apply_sample_parameters`.

- **weather (str)** – The path to the weather file (*.epw) to use for simulation. See the `weather_path` parameter in `cea.demand.demand_main.demand_calculation` for more information.

- **output_parameters (list of str)** – The list of output parameters to save to disk. This is a column-wise subset of the output of `cea.demand.demand_main.demand_calculation`.

**Returns** None

```
cea.analysis.sensitivity.sensitivity_demand_simulate.simulate_demand_sample(locator, config, output_parameters)
```

Run a demand simulation for a single sample. This function expects a locator that is already initialized to the simulation folder, that has already been prepared with `apply_sample_parameters`.

**Parameters**

- **locator (InputLocator)** – The InputLocator to use for the simulation

- **weather (str)** – The path to the weather file (*.epw) to use for simulation. See the `weather_path` parameter in `cea.demand.demand_main.demand_calculation` for more information.

- **output_parameters (list of str)** – The list of output parameters to save to disk. This is a column-wise subset of the output of `cea.demand.demand_main.demand_calculation`.

**Returns** Returns the columns of the results of `cea.demand.demand_main.demand_calculation` as defined in `output_parameters`.

**Return type** pandas.DataFrame

**cea.datamanagement package**

**Submodules**

**cea.datamanagement.create_new_project module**

A tool to create a new project / scenario with the CEA.

```
cea.datamanagement.create_new_project.create_new_project(locator, config)
```

```
cea.datamanagement.create_new_project.main(config)
```
**cea.datamanagement.data_helper module**

**calc_category**

```
(archetype_DB, age, field, type)
```

**calc_code**

```
(code1, code2, code3, code4)
```

**calc_comparison**

```
(array_second, array_max)
```

**calc_mainuse**

```
(uses_df, uses)
```

Calculate a building’s main use:

- **uses_df**: DataFrame containing the share of each building that corresponds to each occupancy type.
- **uses**: list of building uses actually available in the area.

**Return**: mainuse array containing each building’s main occupancy.

**calculate_average_multiuse**

```
(fields, properties_df, occupant_densities, list_uses, properties_DB)
```

This script calculates the average internal loads and ventilation properties for multiuse buildings.

**Parameters**

- **properties_df** (*DataFrame*): DataFrame containing the building’s occupancy type and the corresponding indoor comfort properties or internal loads.
- **occupant_densities** (*Dict*): DataFrame containing the number of people per square meter for each occupancy type based on the archetypes.
- **list_uses** (*list [str]*): list of uses in the project.
- **properties_DB** (*DataFrame*): DataFrame containing each occupancy type’s indoor comfort properties or internal loads based on the corresponding archetypes.

**Return**: properties_df, the same DataFrame as the input parameter, but with the updated properties for multiuse buildings.

**correct_archetype_areas**

```
(prop_architecture_df, architecture_DB, list_uses)
```

Corrects the heated area ‘Hs_ag’ and ‘Hs_bg’ for buildings with multiple uses.

**Variables**

- **prop_architecture_df** – DataFrame containing each building’s occupancy, construction and renovation data as well as the architectural properties obtained from the archetypes.
- **architecture_DB** – architecture database for each archetype.
- **list_uses** – list of all occupancy types in the project.

**Return**: Hs_ag_list, Hs_bg_list, Ns_list, Es_list, the corrected values for ‘Hs_ag’, ‘Hs_bg’, ‘Ns’, and ‘Es’ for each building.

...
cea.datamanagement.data_helper.data_helper(locator, region, overwrite_technology_folder, update_architecture_dbf, update_HVAC_systems_dbf, update_indoor_comfort_dbf, update_internal_loads_dbf, update_supply_systems_dbf, update_schedule_operation_cea, buildings)

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

• **update_architecture_dbf (boolean)** – if True, update the construction and architecture properties.

• **update_indoor_comfort_dbf (boolean)** – if True, get properties about thermal comfort.

• **update_HVAC_systems_dbf (boolean)** – if True, get properties about types of HVAC systems, otherwise False.

• **update_internal_loads_dbf (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.datamanagement.data_helper.get_list_of_uses_in_case_study(building_occupancy_df)

validates lists of uses in case study. refactored from data_helper function

Parameters **building_occupancy_df** (pandas.DataFrame) – dataframe of occupancy.dbf input (can be read in data-helper or in building-properties)

Returns list of uses in case study

Return type pandas.DataFrame.Index

cea.datamanagement.data_helper.get_prop_architecture(categories_df, architecture_DB, list_use)

This function obtains every building’s architectural properties based on the construction and renovation years.

Parameters

• **categories_df (DataFrame)** – DataFrame containing each building’s construction and renovation categories for each building component based on the construction and renovation years

• **architecture_DB** – DataFrame containing the archetypal architectural properties for each use type, construction and renovation year

Return **prop_architecture_df** DataFrame containing the architectural properties of each building in the area

Rtype **prop_architecture_df** DataFrame

cea.datamanagement.data_helper.get_technology_related_databases(locator, region)
cea.datamanagement.data_helper.main(config)

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.

cea.datamanagement.databases_verification module

Databases verification This tool is used as to check the format of each database

cea.datamanagement.databases_verification.assert_columns_names(zone_df, columns)

cea.datamanagement.databases_verification.assert_input_geometry_acceptable_values_floor_height(zone_df)

cea.datamanagement.databases_verification.assert_input_geometry_acceptable_values_floor_height_surroundings(surroundings_df)

cea.datamanagement.databases_verification.verify_input_age(age_df)

cea.datamanagement.databases_verification.verify_input_geometry_surroundings(surroundings_df)

cea.datamanagement.databases_verification.verify_input_geometry_zone(zone_df)

cea.datamanagement.databases_verification.verify_input_occupancy(occupancy_df)

cea.datamanagement.databases_verification.verify_input_terrain(terrain_raster)

cea.datamanagement.schedule_helper module

This script creates schedules per building in CEA

class cea.datamanagement.schedule_helper.ScheduleData(locator)
Bases: object

__init__(locator)

x.__init__(...) initializes x; see help(type(x)) for signature

fill_in_data()

cea.datamanagement.schedule_helper.calc_average(last, current, share_of_use)

function to calculate the weighted average of schedules

cea.datamanagement.schedule_helper.calc_mixed_schedule(locator, building_occupancy_df, buildings)

cea.datamanagement.schedule_helper.get_short_list_of_uses_in_case_study(building_occupancy_df)

gets only the list of land uses that all the building occupancy dataset has It avoids multiple iterations, when these landuses are not even selected in the database

Parameters building_occupancy_df (pandas.DataFrame) – dataframe of occupancy.dbf input (can be read in data-helper or in building-properties)

Returns list of uses in case study

Return type pandas.DataFrame.Index

cea.datamanagement.schedule_helper.main(config)
**cea.datamanagement.streets_helper module**

This script extracts streets from Open street maps

```python
cea.datamanagement.streets_helper.calcBoundingBox(shapefile_surroundings, shapefile_zone)
```

this is where the action happens if it is more than a few lines in `main`. NOTE: ADD YOUR SCRIPT’S DOCUMENTATION HERE (how) NOTE: RENAME THIS FUNCTION (SHOULD PROBABLY BE THE SAME NAME AS THE MODULE)

```python
cea.datamanagement.streets_helper.main(config)
```

This is the main entry point to your script. Any parameters used by your script must be present in the `config` parameter. The CLI will call this `main` function passing in a `config` object after adjusting the configuration to reflect parameters passed on the command line - this is how the ArcGIS interface interacts with the scripts BTW.

**Parameters**

- `config` (`cea.config.Configuration`) –
  Returns

**Returns**

**cea.datamanagement.surroundings_helper module**

This script extracts surrounding buildings of the zone geometry from Open street maps

```python
cea.datamanagement.surroundings_helper.alphaShape(points, alpha)
```

Compute the alpha shape (concave hull) of a set of points.

**Parameters**

- `points` – Iterable container of points.
- `alpha` – alpha value to influence the gooeyness of the border. Smaller numbers don’t fall inward as much as larger numbers. Too large, and you lose everything!

```python
cea.datamanagement.surroundings_helper.calcSurroundingArea(zone_gdf, buffer_m)
```

```python
cea.datamanagement.surroundings_helper.cleanAttributes(shapefile, buildings_height, buildings_floors, key)
```

```python
cea.datamanagement.surroundings_helper.eraseNoSurroundingAreas(all_surroundings, area_buffer)
```

this is where the action happens if it is more than a few lines in `main`. NOTE: ADD YOUR SCRIPT’S DOCUMENTATION HERE (how) NOTE: RENAME THIS FUNCTION (SHOULD PROBABLY BE THE SAME NAME AS THE MODULE)

```python
cea.datamanagement.surroundings_helper.main(config)
```

This is the main entry point to your script. Any parameters used by your script must be present in the `config` parameter. The CLI will call this `main` function passing in a `config` object after adjusting the configuration to reflect parameters passed on the command line - this is how the ArcGIS interface interacts with the scripts BTW.

**Parameters**

- `config` (`cea.config.Configuration`) –
  Returns

9.1. cea package
**cea.datamanagement.terrain_helper module**

This script extracts terrain elevation from NASA - SRTM https://www2.jpl.nasa.gov/srtm/

*cea.datamanagement.terrain_helper.calcBoundingBoxProjectedCoordinates*(shapefile_zone, shapefile_surroundings)

*cea.datamanagement.terrain_helper.calcRasterTerrainFixedElevation*(crs, elevation, grid_size, raster_path, locator, x_max, x_min, y_max, y_min)

*cea.datamanagement.terrain_helper.main*(config)

This is the main entry point to your script. Any parameters used by your script must be present in the config parameter. The CLI will call this main function passing in a config object after adjusting the configuration to reflect parameters passed on the command line - this is how the ArcGIS interface interacts with the scripts BTW.

**Parameters config** (cea.config.Configuration) –

**Returns**

*cea.datamanagement.terrain_helper.requestElevation*(lon, lat)

*cea.datamanagement.terrain_helper.terrainElevationExtractor*(locator, config)

this is where the action happens if it is more than a few lines in main. NOTE: ADD YOUR SCRIPT’S DOCUMENTATION HERE (how) NOTE: RENAME THIS FUNCTION (SHOULD PROBABLY BE THE SAME NAME AS THE MODULE)

**cea.datamanagement.weather_helper module**

The weather-helper script sets the weather data used (inputs/weather.epw) for simulations.

*cea.datamanagement.weather_helper.copyWeatherFile*(source_weather_file, locator)

Copy a weather file to the scenario’s inputs.

**Parameters**

* source_weather_file (string) – path to a weather file (*.epw)

* locator (cea.inputlocator.InputLocator) – use the InputLocator to find output path

**Returns** (this script doesn’t return anything)

*cea.datamanagement.weather_helper.main*(config)

This is the main entry point to your script. Any parameters used by your script must be present in the config parameter. The CLI will call this main function passing in a config object after adjusting the configuration to reflect parameters passed on the command line - this is how the ArcGIS interface interacts with the scripts BTW.

**Parameters config** (cea.config.Configuration) – Configuration object for this script
Returns

**cea.datamanagement.zone_helper module**

This is a template script - an example of how a CEA script should be set up. NOTE: ADD YOUR SCRIPT’S DOCUMENTATION HERE (what, why, include literature references)

`cea.datamanagement.zone_helper.calculate_age_file(zone_df, year_construction, age_output_path)`

This script fills in the age.dbf file with one year of construction :param zone_df: :param year_construction: :param age_output_path: :return:

`cea.datamanagement.zone_helper.calculate_occupancy_file(zone_df, occupancy_type, occupancy_output_path)`

This script fills in the occupancy.dbf file with one occupancy type :param zone_df: :param occupancy_type: :param occupancy_output_path: :return:

`cea.datamanagement.zone_helper.check_if_float(x)`

`cea.datamanagement.zone_helper.clean_attributes(shapefile, buildings_height, buildings_floors, buildings_height_below_ground, buildings_floors_below_ground, key)`

`cea.datamanagement.zone_helper.main(config)`

This script gets a polygon and calculates the zone.shp and the occupancy.dbf and age.dbf inputs files for CEA

`cea.datamanagement.zone_helper.polygon_to_zone(buildings_floors, buildings_height_below_ground, buildings_height, buildings_floors_below_ground, poly, shapefile_out_path)`

`cea.datamanagement.zone_helper.zone_helper(locator, config)`

This script gets a polygon and calculates the zone.shp and the occupancy.dbf and age.dbf inputs files for CEA :param locator: :param config: :return:

**cea.demand package**

Subpackages

**cea.demand.schedule_maker package**

Submodules

**cea.demand.schedule_maker.schedule_maker module**

`cea.demand.schedule_maker.schedule_maker.calc_hourly_value(date, array_week, array_sat, array_sun, norm_weekday_max, norm_sat_max, norm_sun_max, monthly_multiplier)`
**Calculates** the stochastic occupancy pattern for an individual based on Page et al. (2008). The so-called parameter of mobility mu is assumed to be a uniformly-distributed random float between 0 and 0.5 based on the range of values presented in the aforementioned paper.

**Parameters**

- **deterministic_schedule** (array(float)) – deterministic schedule of occupancy provided in the user inputs

**Return pattern**

- yearly occupancy pattern for a given occupant in a given occupancy type

**Rtype pattern** list[int]

---

**Calculate the profile of occupancy, electricity demand and domestic hot water consumption from the input schedules. For variables that depend on the number of people (humidity gains, heat gains and ventilation demand), additional schedules are created given the the number of people present at each time.**

Two occupant models are included: a deterministic one, which simply uses the schedules provided by the user; and a stochastic one, which is based on the two-state Markov chain model of Page et al. (2008). In the latter case, occupant presence is modeled based on the probability of occupant presence at the current and next time step as given by the occupant schedules used in the deterministic model.

**Parameters**

- **locator** – InputLocator instance
- **building** – name of current building
- **date_range** (DatetimeIndex) – range of dates being considered
- **daily_schedule_building** (dict(str: array)) – building schedules for occupancy, electricity demand, water consumption, and system operation
- **monthly_multiplier** (float) – percentage of the total number of occupants present at each month of the year
- **internal_loads_building** – internal loads for the current building (from case study inputs)
- **indoor_comfort_building** – indoor comfort properties for the current building (from case study inputs)
- **prop_geometry_building** – building geometry (from case study inputs)
- **stochastic_schedule** – Boolean that defines whether the stochastic occupancy model should be used

**Calculates** the transition probabilities at a given time step as defined by Page et al. (2008). These are the probability of arriving (T01) and the probability of staying in (T11) given the parameter of mobility mu, the probability of the present state (P0), and the probability of the next state t+1 (P1).

**Parameters**

- **mu** (float) – parameter of mobility
- **P0** (float) – probability of presence at the current time step t
- **P1 (float)** – probability of presence at the next time step t+1

Return **T01** probability of transition from absence to presence at current time step

Rtype **T01** float

Return **T11** probability of transition from presence to presence at current time step

Rtype **T11** float

```python
csa.demand.schedule_maker.schedule_maker.convert_schedule_string_to_temperature(schedule_string, schedule_type, Ths_set_C, Ths_setb_C, Tcs_set_C, Tcs_setb_C)
```

converts an archetypal temperature schedule consisting of strings to building-specific temperatures:

- param schedule_string: list of strings containing codes: ‘OFF’, ‘SETPOINT’, ‘SETBACK’
- type schedule_string: list of strings
- param schedule_type: type of the schedule, either ‘Ths_set’ or ‘Tcs_set’
- type schedule_type: str
- param bpr: BuildingPropertiesRow object, from here the setpoint and setback temperatures are extracted
- type bpr: BuildingPropertiesRow

:return: an array of temperatures containing np.nan when the system is OFF
- rtype: numpy.array

```python
csa.demand.schedule_maker.schedule_maker.get_random_presence(p)
```

Get the current occupant state (presence=1 or absence=0) at the current time step given a probability p.

**Parameters**

- p *(float)* – A probability (e.g. T01, T11)

**Returns**

The randomly-chosen state (0 or 1).

```python
csa.demand.schedule_maker.schedule_maker.get_yearly_vectors(date_range, days_in_schedule, schedule_array, monthly_multiplier, normalize_first_daily_profile=False)
```

```python
csa.demand.schedule_maker.schedule_maker.main(config)
```

```python
csa.demand.schedule_maker.schedule_maker.print_progress(i, n, args, result)
```

```python
csa.demand.schedule_maker.schedule_maker.schedule_maker_main(locator, config, building=None)
```

**Submodules**

**cea.demand.airconditioning_model module**

Air conditioning equipment component models

```python
csa.demand.airconditioning_model.central_air_handling_unit_cooling(m_ve_mech, t_ve_mech_after_hex, x_ve_mech, bpr)
```

the central air handling unit acts on the mechanical ventilation air stream it has a fixed coil and fixed supply temperature the input is the cooling load that should be achieved, however over-cooling is possible dehumidification/latent cooling is a by product as the ventilation air is supplied at the coil temperature dew point

Gabriel Happle, Feb. 2018
Parameters

- `m_ve_mech` -
- `t_ve_mech_after_hex` -
- `x_ve_mech` -
- `bpr` -

Returns

`cea.demand.airconditioning_model.central_air_handling_unit_heating` (`m_ve_mech`,
`t_ve_mech_after_hex`,
`x_ve_mech`,
`ta_hs_set`,
`T_int`,
`bpr`)

the central air handling unit acts on the mechanical ventilation air stream it has a fixed coil and fixed supply temperature the input is the heating load that should be achieved, however over-heating is possible

Gabriel Happle, Feb. 2018

Parameters

- `m_ve_mech` – mechanical ventilation air flow, outdoor air (kg/s)
- `t_ve_mech_after_hex` – supply air temperature of mechanical ventilation (i.e. after HEX)
- `x_ve_mech` – moisture content of ventilation airflows
- `ta_hs_set` – heating system set point air temperature
- `bpr` –

Returns

`cea.demand.airconditioning_model.electric_humidification_unit` (`g_hu`,
`m_ve_mech`)

Refactored from Legacy Central AC can have a humidification unit. If humidification load is present, only the mass flow of outdoor air to be humidified is relevant

Parameters

- `g_hu` (double) – humidification load, water to be evaporated (kg/s)
- `m_ve_mech` (double) – mechanical ventilation air flow, outdoor air (kg/s)

Returns `e_hs_lat_aux`, electric load of humidification (W)

Return type double

`cea.demand.airconditioning_model.local_air_recirculation_unit_cooling` (`qc_sen_demand_aru`,
`g_dhu_demand_aru`,
`t_int_prev`,
`x_int_prev`,
`bpr`,
`t_control`,
`x_control`)

the local air recirculation unit recirculates internal air it determines the mass flow of air according to the demand of sensible or latent cooling the air flow can be controlled by sensible OR latent load it has a fixed coil and fixed supply temperature dehumidification/latent cooling is a by product as the ventilation air is supplied at the coil temperature dew point

Gabriel Happle, Feb. 2018
Parameters

- **qc_sen_demand_aru**
- **g_dhu_demand_aru**
- **t_int_prev**
- **x_int_prev**
- **bpr**
- **t_control**
- **x_control**

Returns

```
cea.demand.airconditioning_model.local_air_recirculation_unit_heating(qh_sen_demand_aru, t_int_prev, bpr)
```

the local air recirculation unit recirculates internal air it determines the mass flow of air according to the demand of sensible heating it has a fixed coil and fixed supply temperature

Gabriel Happle, Feb. 2018

Parameters

- **qh_sen Demand_aru**
- **t_int_prev**
- **bpr**

Returns

```
cea.demand.airconditioning_model.local_sensible_cooling_unit()
```

### `cea.demand.building_properties` module

```python
class cea.demand.building_properties.BuildingProperties(locator, override_variables=False)
```

Bases: `object`

Groups building properties used for the calc-thermal-loads functions. Stores the full DataFrame for each of the building properties and provides methods for indexing them by name.

G. Happle BuildingPropsThermalLoads 27.05.2016

__getitem__ (building_name)

return a (read-only) BuildingPropertiesRow for the building

__init__ (locator, override_variables=False)

Read building properties from input shape files and construct a new BuildingProperties object.

Parameters

- **locator** (`cea.inputlocator.InputLocator`) – an InputLocator for locating the input files
- **override_variables** (`bool`) – override_variables from config

Returns BuildingProperties

Return type `BuildingProperties`
__len__()  
return length of list_building_names

apply_overrides(df)  
Apply the overrides to df. This works by checking each column in the selfoverrides dataframe and overwriting any columns in df with the same name. selfoverrides and df are assumed to have the same index.

calc_bounds_box_geom(geometry_shapefile)

calc_prop_rc_model(locator, occupancy, envelope, geometry, hvac_temperatures)
Return the RC model properties for all buildings. The RC model used is described in ISO 13790:2008, Annex C (Full set of equations for simple hourly method).

Parameters

- **occupancy (Gdf)** – The contents of the occupancy.shp file, indexed by building name. Each column is the name of an occupancy type (GYM, HOSPITAL, HOTEL, INDUSTRIAL, MULTI_RES, OFFICE, PARKING, etc.) except for the “PFloor” column which is a fraction of heated floor area. The occupancy types must add up to 1.0.
- **envelope (Gdf)** – The contents of the architecture.shp file, indexed by building name. It contains the following fields:
  - n50: Air tightness at 50 Pa [h^-1].
  - type_shade: shading system type.
  - win_wall: window to wall ratio.
  - U_base: U value of the floor construction [W/m2K]
  - U_roof: U value of roof construction [W/m2K]
  - U_wall: U value of wall construction [W/m2K]
  - U_win: U value of window construction [W/m2K]
  - Hs: Fraction of gross floor area that is heated/cooled {0 <= Hs <= 1}
  - Cm_Af: Internal heat capacity per unit of area [J/K.m2]
- **geometry (Gdf)** – The contents of the zone.shp file indexed by building name - the list of buildings, their floor counts, heights etc. Includes additional fields “footprint” and “perimeter” as calculated in read_building_properties.
- **hvac_temperatures (DataFrame)** – The return value of get_properties_technical_systems.

Returns RC model properties per building

Return type DataFrame

Sample result data calculated or manipulated by this method:

Name: B153767

datatype: float64
<table>
<thead>
<tr>
<th>Column</th>
<th>e.g.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atot</td>
<td>4.564827e+03</td>
<td>(area of all surfaces facing the building zone in [m²])</td>
</tr>
<tr>
<td>Aw</td>
<td>4.527014e+02</td>
<td>(area of windows in [m²])</td>
</tr>
<tr>
<td>Am</td>
<td>6.947967e+03</td>
<td>(effective mass area in [m²])</td>
</tr>
<tr>
<td>Aef</td>
<td>2.171240e+03</td>
<td>(floor area with electricity in [m²])</td>
</tr>
<tr>
<td>Af</td>
<td>2.171240e+03</td>
<td>(conditioned floor area (heated/cooled) in [m²])</td>
</tr>
<tr>
<td>Cm</td>
<td>6.513719e+08</td>
<td>(internal heat capacity in [J/k])</td>
</tr>
<tr>
<td>Htr_is</td>
<td>1.574865e+04</td>
<td>(thermal transmission coefficient between air and surface nodes in RC-model in [W/K])</td>
</tr>
<tr>
<td>Htr_em</td>
<td>5.829963e+04</td>
<td>(thermal transmission coefficient between exterior and thermal mass nodes in RC-model in [W/K])</td>
</tr>
<tr>
<td>Htr_ms</td>
<td>6.522650e+04</td>
<td>(thermal transmission coefficient between surface and thermal mass nodes in RC-model in [W/K])</td>
</tr>
<tr>
<td>Htr_op</td>
<td>5.776698e+02</td>
<td>(thermal transmission coefficient for opaque surfaces in [W/K])</td>
</tr>
<tr>
<td>Hg</td>
<td>2.857637e+02</td>
<td>(steady-state thermal transmission coefficient to the ground in [W/K])</td>
</tr>
<tr>
<td>HD</td>
<td>2.919060e+02</td>
<td>(direct thermal transmission coefficient to the external environment in [W/K])</td>
</tr>
<tr>
<td>Htr_w</td>
<td>1.403374e+03</td>
<td>(thermal transmission coefficient for windows and glazing in [W/K])</td>
</tr>
</tbody>
</table>

FIXME: rename Awall_all to something more sane...

**geometry_reader_radiation_daysim** (locator, envelope, occupancy, geometry)

Reader which returns the radiation specific geometries from Daysim. Adjusts the imported data such that it is consistent with other imported geometry parameters.

**Parameters**

- **locator** – an InputLocator for locating the input files
- **envelope** – The contents of the architecture.shp file, indexed by building name.
- **occupancy** – The contents of the occupancy.shp file, indexed by building name.
- **geometry** – The contents of the zone.shp file indexed by building name.
- **floor_height** – Height of the floor in [m].

**Returns**

Adjusted Daysim geometry data containing the following:

- Name: Name of building.
- Aw: Area of windows for each building (using mean window to wall ratio for building, excluding voids) [m²]
- Awall_ag: Opaque wall areas above ground (excluding voids, windows and roof) [m²]
- Aop_bg: Opaque areas below ground (including ground floor, excluding voids and windows) [m²]
- Aroof: Area of the roof (considered flat and equal to the building footprint) [m²]
- GFA_m2: Gross floor area [m²]
- floors: Sum of floors below ground (floors_bg) and floors above ground (floors_ag) [m²]
- surface_volume: Surface to volume ratio [m⁴⁻¹]

**Return type** DataFrame
Data is read from `cea.inputlocator.InputLocator.get_radiation_metadata()` (e.g. `C:/scenario/outputs/data/solar-radiation/{building_name}_geometry.csv`)

Note: File generated by the radiation script. It contains the fields Name, Freeheight, FactorShade, height_ag and Shape_Leng. This data is used to calculate the wall and window areas.

`get_overrides_columns()`
Return the list of column names in the `overrides.csv` file or an empty list if no such file is present.

`get_prop_age(name_building)`
get age properties of a building by name

`get_prop_comfort(name_building)`
get comfort properties of a building by name

`get_prop_envelope(name_building)`
get the architecture and thermal properties of a building by name

`get_prop_geometry(name_building)`
get geometry of a building by name

`get_prop_hvac(name_building)`
get HVAC properties of a building by name

`get_prop_internal_loads(name_building)`
get internal loads properties of a building by name

`get_prop_occupancy(name_building)`
get the occupancy properties of a building by name

`get_prop_rc_model(name_building)`
get RC-model properties of a building by name

`get_prop_supply_systems(name_building)`
get geometry of a building by name

`get_solar(name_building)`
get solar properties of a building by name

`list_building_names()`
get list of all building names

`list_uses()`
get list of all uses (occupancy types)

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

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

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

```python
class cea.demand.building_properties.BuildingPropertiesRow(name, geometry, envelope, occupancy, hvac, rc_model, comfort, internal_loads, age, solar, supply)
```

**Bases:** `object`

Encapsulate the data of a single row in the DataSets of BuildingProperties. This class meant to be read-only.
__init__ (name, geometry, envelope, occupancy, hvac, rc_model, comfort, internal_loads, age, solar, supply)

Create a new instance of BuildingPropertiesRow - meant to be called by BuildingProperties[building_name]. Each of the arguments is a pandas Series object representing a row in the corresponding DataFrame.

_calc_form ()

_calculate_pipe_transmittance_values ()

linear transmissivity coefficients of piping W/(m.K)

_get_properties_building_systems ()

Method for defining the building system properties, specifically the nominal supply and return temperatures, equivalent pipe lengths and transmittance losses. The systems considered include an ahu (air handling unit), rsu (air recirculation unit), and scu/shu (sensible cooling / sensible heating unit). Note: it is assumed that building with less than a floor and less than 2 floors underground do not require heating and cooling, and are not considered when calculating the building system properties.

Returns

building_systems dict containing the following information:

Pipe Lengths:

• Lcww_dis: length of hot water piping in the distribution circuit (???) [m]
• Lsww_dis: length of hot water piping in the distribution circuit (???) [m]
• Lvww_dis: length of hot water piping in the distribution circuit (???) [m]
• Lvww_c: length of piping in the heating system circulation circuit (ventilated/recirc?) [m]
• Lv: length vertical lines [m]

Heating Supply Temperatures:

• Ths_sup_ahu_0: heating supply temperature for AHU (C)
• Ths_sup_aru_0: heating supply temperature for ARU (C)
• Ths_sup_shu_0: heating supply temperature for SHU (C)

Heating Return Temperatures:

• Ths_re_ahu_0: heating return temperature for AHU (C)
• Ths_re_aru_0: heating return temperature for ARU (C)
• Ths_re_shu_0: heating return temperature for SHU (C)

Cooling Supply Temperatures:

• Tcs_sup_ahu_0: cooling supply temperature for AHU (C)
• Tcs_sup_aru_0: cooling supply temperature for ARU (C)
• Tcs_sup_scu_0: cooling supply temperature for SCU (C)

Cooling Return Temperatures:

• Tcs_re_ahu_0: cooling return temperature for AHU (C)
• Tcs_re_aru_0: cooling return temperature for ARU (C)
• Tcs_re_scu_0: cooling return temperature for SCU (C)

Water supply temperature??:

• Tww_sup_0: ?????
Thermal losses in pipes:

- Y: Linear trasmissivity coefficients of piping depending on year of construction [W/m.K]

Form Factor Adjustment:
- fforma: form factor comparison between real surface and rectangular ???

**Return type**  `dict`

```python
class cea.demand.building_properties.EnvelopeProperties(envelope)
    Bases: object

    Encapsulate a single row of the architecture input file for a building
    __init__(envelope)
        x.__init__(...) initializes x; see help(type(x)) for signature

class cea.demand.building_properties.SolarProperties(solar)
    Bases: object

    Encapsulates the solar properties of a building
    I_sol
    __init__(solar)
        x.__init__(...) initializes x; see help(type(x)) for signature
    __slots__ = ['I_sol']
```

```python
cea.demand.building_properties.calc_Isol_daysim(building_name, locator, prop_envelope, prop_rc_model, thermal_resistance_surface)

Reads Daysim geometry and radiation results and calculates the sensible solar heat loads based on the surface area and building envelope properties.

**Parameters**

- `building_name` – Name of the building (e.g. B154862)
- `locator` – an InputLocator for locating the input files
- `prop_envelope` – contains the building envelope properties.
- `prop_rc_model` – RC model properties of a building by name.

**Returns**  `I_sol`: numpy array containing the sensible solar heat loads for roof, walls and windows.

**Return type**  `np.array`

```python
cea.demand.building_properties.calc_useful_areas(df)
```

```python
cea.demand.building_properties.get_envelope_properties(locator, prop_architecture)

Gets the building envelope properties from databases/Systems/emission_systems.csv, including the following:

- prop_roof: Name, emissivity (e_roof), absorbtivity (a_roof), thermal resistance (U_roof), and fraction of heated space (Hs).
- prop_wall: Name, emissivity (e_wall), absorbtivity (a_wall), thermal resistance (U_wall & U_base), window to wall ratio of north, east, south, west walls (wwr_north, wwr_east, wwr_south, wwr_west).
- prop_win: Name, emissivity (e_win), solar factor (G_win), thermal resistance (U_win)
- prop_shading: Name, shading factor (rf_sh).
```
• prop_construction: Name, internal heat capacity (Cm_af), floor to ceiling voids (void_deck).
• prop_leakage: Name, exfiltration (n50).

Creates a merged df containing aforementioned envelope properties called envelope_prop.

Returns envelope_prop
Return type DataFrame

cea.demand.building_properties.get_prop_solar(locator, prop_rc_model, prop_envelope)

Gets the sensible solar gains from calc_Isol_daysim and stores in a dataframe containing building ‘Name’ and I_sol (incident solar gains).

Parameters
• locator – an InputLocator for locating the input files
• prop_rc_model – RC model properties of a building by name.
• prop_envelope – dataframe containing the building envelope properties.

Returns dataframe containing the sensible solar gains for each building by name called result.
Return type Dataframe

cea.demand.building_properties.get_properties_supply_systems(locator, properties_supply)

cea.demand.building_properties.get_properties_technical_systems(locator, prop_HVAC)

Return temperature data per building based on the HVAC systems of the building. Uses the emission_systems.xls file to look up properties

Parameters
• locator (cea.inputlocator.InputLocator) – an InputLocator for locating the input files
• prop_HVAC (geopandas.GeoDataFrame) – HVAC properties for each building (type of cooling system, control system, domestic hot water system and heating system. The values can be looked up in the contributors manual: https://architecture-building-systems.gitbooks.io/cea-toolbox-for-arcgis-manual/content/building_properties.html#mechanical-systems

Sample data (first 5 rows):

<table>
<thead>
<tr>
<th>Name</th>
<th>type_cs</th>
<th>type_ctrl</th>
<th>type_dhw</th>
<th>type_hs</th>
<th>type_vent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B154862</td>
<td>T0</td>
<td>T1</td>
<td>T1</td>
<td>T1</td>
</tr>
<tr>
<td>1</td>
<td>B153604</td>
<td>T0</td>
<td>T1</td>
<td>T1</td>
<td>T1</td>
</tr>
<tr>
<td>2</td>
<td>B153831</td>
<td>T0</td>
<td>T1</td>
<td>T1</td>
<td>T1</td>
</tr>
<tr>
<td>3</td>
<td>B302022960</td>
<td>T0</td>
<td>T0</td>
<td>T0</td>
<td>T0</td>
</tr>
<tr>
<td>4</td>
<td>B302034063</td>
<td>T0</td>
<td>T0</td>
<td>T0</td>
<td>T0</td>
</tr>
</tbody>
</table>

Returns A DataFrame containing temperature data for each building in the scenario. More information can be found in the contributors manual: https://architecture-building-systems.gitbooks.io/cea-toolbox-for-arcgis-manual/content/delivery_technologies.html

Return type DataFrame

Each row contains the following fields:
<table>
<thead>
<tr>
<th>Column</th>
<th>e.g.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>B154862</td>
<td>(building name)</td>
</tr>
<tr>
<td>type_hs</td>
<td>T1</td>
<td>(copied from input, code for type of heating system)</td>
</tr>
<tr>
<td>type_cs</td>
<td>T0</td>
<td>(copied from input, code for type of cooling system)</td>
</tr>
<tr>
<td>type_dhw</td>
<td>T1</td>
<td>(copied from input, code for type of hot water system)</td>
</tr>
<tr>
<td>type_ctrl</td>
<td>T1</td>
<td>(copied from input, code for type of controller for heating and cooling system)</td>
</tr>
<tr>
<td>type_vent</td>
<td>T1</td>
<td>(copied from input, code for type of ventilation system)</td>
</tr>
<tr>
<td>Tsh0_C</td>
<td>90</td>
<td>(heating system supply temperature at nominal conditions [C])</td>
</tr>
<tr>
<td>dThs0_C</td>
<td>20</td>
<td>(delta of heating system temperature at nominal conditions [C])</td>
</tr>
<tr>
<td>Qhsmax_Wm2</td>
<td>500</td>
<td>(maximum heating system power capacity per unit of gross built area [W/m2])</td>
</tr>
<tr>
<td>dThs_C</td>
<td>0.15</td>
<td>(correction temperature of emission losses due to type of heating system [C])</td>
</tr>
<tr>
<td>Tcs0_C</td>
<td>0</td>
<td>(cooling system supply temperature at nominal conditions [C])</td>
</tr>
<tr>
<td>dTcs0_C</td>
<td>0</td>
<td>(delta of cooling system temperature at nominal conditions [C])</td>
</tr>
<tr>
<td>Qcsmax_Wm2</td>
<td>0</td>
<td>(maximum cooling system power capacity per unit of gross built area [W/m2])</td>
</tr>
<tr>
<td>dTcs_C</td>
<td>0.5</td>
<td>(correction temperature of emission losses due to type of cooling system [C])</td>
</tr>
<tr>
<td>dT_Qhs</td>
<td>1.2</td>
<td>(correction temperature of emission losses due to control system of heating [C])</td>
</tr>
<tr>
<td>dT_Qcs</td>
<td>-1.2</td>
<td>(correction temperature of emission losses due to control system of cooling [C])</td>
</tr>
<tr>
<td>Tsww0_C</td>
<td>60</td>
<td>(dhw system supply temperature at nominal conditions [C])</td>
</tr>
<tr>
<td>Qwwmax_Wm2</td>
<td>500</td>
<td>(maximum dhw system power capacity per unit of gross built area [W/m2])</td>
</tr>
<tr>
<td>MECH_VENT</td>
<td>True</td>
<td>(copied from input, ventilation system configuration)</td>
</tr>
<tr>
<td>WIN_VENT</td>
<td>False</td>
<td>(copied from input, ventilation system configuration)</td>
</tr>
<tr>
<td>HEAT_REC</td>
<td>True</td>
<td>(copied from input, ventilation system configuration)</td>
</tr>
<tr>
<td>NIGHT_FLSH</td>
<td>True</td>
<td>(copied from input, ventilation system control strategy)</td>
</tr>
<tr>
<td>ECONOMIZER</td>
<td>False</td>
<td>(copied from input, ventilation system control strategy)</td>
</tr>
</tbody>
</table>

Data is read from `cea.inputlocator.InputLocator.get_technical_emission_systems()` (e.g. `db/Systems/emission_systems.csv`)

- `cea.demand.building_properties.verify_has_season(building_name, start, end)`
- `cea.demand.building_properties.verify_overlap_season(building_name, has_heating_season, has_cooling_season, heat_start, heat_end, cool_start, cool_end)`
- `cea.demand.building_properties.weird_division(n, d)`

**cea.demand.constants module**

This file contains the constants used in the building energy demand calculations

**cea.demand.control_heating_cooling_systems module**

- `cea.demand.control_heating_cooling_systems.convert_date_to_hour(date)`
  converts date in ‘DD/MM’ format into hour of the year (first hour of the day) i.e. ‘01/01’ results in 0
**Parameters** `date (str)` – date in ‘DD|MM’ format (from .xlsx database input)

**Returns** hour of the year (first hour of the day)

**Return type** `int`

`cea.demand.control_heating_cooling_systems.cooling_system_is_active (bpr, tsd, t)`

Checks whether the cooling system is active according to rules for a specific hour of the year i.e., is there a set point temperature?

**Parameters**

- `tsd (dict)` – a dictionary of time step data mapping variable names to ndarrays for each hour of the year.
- `t (int)` – hour of the year, simulation time step [0…HOURS_IN_YEAR]

**Returns** True or False

**Return type** `bool`

`cea.demand.control_heating_cooling_systems.get_cooling_system_set_point (t, Tcs_set_C, bpr)`

**Parameters**

- `people` –
- `t (int)` – hour of the year, simulation time step [0…HOURS_IN_YEAR]
- `bpr (cea.demand.building_properties.BuildingPropertiesRow)` – BuildingPropertiesRow
- `Tcs_set_C` –

**Returns** cooling system set point temperature [°C]

**Return type** `double`

`cea.demand.control_heating_cooling_systems.get_heating_system_set_point (t, Ths_set_C, bpr)`

**Parameters**

- `people` –
- `t (int)` – hour of the year, simulation time step [0…HOURS_IN_YEAR]
- `bpr (cea.demand.building_properties.BuildingPropertiesRow)` – BuildingPropertiesRow
- `Ths_set_C` –

**Returns** heating system set point temperature [°C]

**Return type** `double`

`cea.demand.control_heating_cooling_systems.get_temperature_setpoints_incl_seasonality (tsd, bpr, schedules)`

**Parameters**

9.1. cea package
• **tsd** *(dict)* – a dictionary of time step data mapping variable names to ndarrays for each hour of the year.

• **bpr** *(cea.demand.building_properties.BuildingPropertiesRow)* – BuildingPropertiesRow

• **weekday** –

  Returns  tsd with updated columns

  Return type  dict

**cea.demand.control_heating_cooling_systems**.has_3for2_cooling_system(*bpr*)

Checks if building has 3for2 cooling system

Parameters  

  **bpr** *(cea.demand.building_properties.BuildingPropertiesRow)* – BuildingPropertiesRow

Returns  True or False

Return type  bool

**cea.demand.control_heating_cooling_systems**.has_ceiling_cooling_system(*bpr*)

Checks if building has ceiling cooling system

Parameters  

  **bpr** *(cea.demand.building_properties.BuildingPropertiesRow)* – BuildingPropertiesRow

Returns  True or False

Return type  bool

**cea.demand.control_heating_cooling_systems**.has_central_ac_cooling_system(*bpr*)

Checks if building has central AC cooling system

Parameters  

  **bpr** *(cea.demand.building_properties.BuildingPropertiesRow)* – BuildingPropertiesRow

Returns  True or False

Return type  bool

**cea.demand.control_heating_cooling_systems**.has_central_ac_heating_system(*bpr*)

Checks if building has central AC heating system

Parameters  

  **bpr** *(cea.demand.building_properties.BuildingPropertiesRow)* – BuildingPropertiesRow

Returns  True or False

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** *(cea.demand.building_properties.BuildingPropertiesRow)* – BuildingPropertiesRow

Returns  True or False

Return type  bool

**cea.demand.control_heating_cooling_systems**.has_floor_cooling_system(*bpr*)

Checks if building has ceiling cooling system

Parameters  

  **bpr** *(cea.demand.building_properties.BuildingPropertiesRow)* – BuildingPropertiesRow
Returns True or False
Return type bool

`cea.demand.control_heating_cooling_systems.has_floor_heating_system(bpr)`
Checks if building has floor heating system

Parameters `bpr (cea.demand.building_properties.BuildingPropertiesRow)` – BuildingPropertiesRow

Returns True or False
Return type bool

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

Parameters `bpr (cea.demand.building_properties.BuildingPropertiesRow)` – BuildingPropertiesRow

Returns True or False
Return type bool

`cea.demand.control_heating_cooling_systems.has_local_ac_cooling_system(bpr)`
Checks if building has mini-split unit AC cooling system

Parameters `bpr (cea.demand.building_properties.BuildingPropertiesRow)` – BuildingPropertiesRow

Returns True or False
Return type bool

`cea.demand.control_heating_cooling_systems.has_radiator_heating_system(bpr)`
Checks if building has radiator heating system

Parameters `bpr (cea.demand.building_properties.BuildingPropertiesRow)` – BuildingPropertiesRow

Returns True or False
Return type bool

`cea.demand.control_heating_cooling_systems.heating_system_is_active(tsd, t)`
Checks whether the heating system is active according to rules for a specific hour of the year i.e., is there a set point temperature

Parameters
- `tsd (dict)` – a dictionary of time step data mapping variable names to ndarrays for each hour of the year.
- `t (int)` – hour of the year, simulation time step [0..HOURS_IN_YEAR]

Returns True or False
Return type bool

`cea.demand.control_heating_cooling_systems.is_cooling_season(t, bpr)`
Checks if time step is part of the cooling season for the building

Parameters
- `t (int)` – hour of the year, simulation time step [0..HOURS_IN_YEAR]
• bpr (cea.demand.building_properties.BuildingPropertiesRow) – BuildingPropertiesRow

Returns True or False

Return type bool

cea.demand.control_heating_cooling_systems.is_heating_season(t, bpr)

checks if time step is part of the heating season for the building

Parameters

• t (int) – hour of the year, simulation time step [0…HOURS_IN_YEAR]

• bpr (cea.demand.building_properties.BuildingPropertiesRow) – BuildingPropertiesRow

Returns True or False

Return type bool

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_day_time(t)

Check if a certain hour of the year is during the daytime or not

Parameters t –

Returns

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 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)
### 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

### Check if a certain hour of year is during night or not

**Parameters** $t$ –

**Returns**

### it calculates final loads

**Returns**

**Return type**  bool

### Analytical energy demand model algorithm

**Algorithm** to calculate the hourly demand of energy services in buildings using the integrated model of [Fonseca2015].

**Produces a demand file per building and a total demand file for the whole zone of interest:**

9.1. **cea package**
• a csv file for every building with hourly demand data.
• Total_demand.csv, csv file of yearly demand data per building.

Parameters

• **locator** (*cea.inputlocator.InputLocator*) – An InputLocator to locate input files
• **weather_path** (*str*) – A path to the EnergyPlus weather data file (.epw)
• **use_dynamic_infiltration_calculation** (*bool*) – Set this to True if the (slower) dynamic infiltration calculation method (*cea.demand.ventilation_air_flows_detailed.calc_air_flows*) should be used instead of the standard.
• **multiprocessing** (*bool*) – Set this to True if the multiprocessing module should be used to speed up calculations by making use of multiple cores.

Returns None

**Return type** NoneType

`cea.demand.demand_main.main(config)`

`cea.demand.demand_main.print_progress(i, n, args, result)`

`cea.demand.demand_main.radiation_files_exist(config, locator)`

**cea.demand.demand_writers module**

A collection of classes that write out the demand results files. 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*.

```python
class cea.demand.demand_writers.DemandWriter (loads, massflows, temperatures)
    Bases: object
    This is meant to be an abstract base class: Use the subclasses of this class instead. Subclasses are expected to:
    - set the *vars_to_print* field in the constructor (FIXME: describe the *vars_to_print* structure.
    - implement the *write_to_csv* method

    __init__ (loads, massflows, temperatures)
    x.__init__(...) initializes x; see help(type(x)) for signature

    calc_hourly_dataframe (building_name, date, tsd)

    calc_yearly_dataframe (bpr, building_name, tsd)

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

    results_to_hdf5 (tsd, bpr, locator, date, building_name)

class cea.demand.demand_writers.HourlyDemandWriter (loads, massflows, temperatures)
    Bases: cea.demand.demand_writers.DemandWriter

    Write out the hourly demand results

    __init__ (loads, massflows, temperatures)
    x.__init__(...) initializes x; see help(type(x)) for signature

    write_to_csv (building_name, columns, hourly_data, locator)
```
write_to_hdf5 (building_name, columns, hourly_data, locator)

class cea.demand.demand_writers.MonthlyDemandWriter (loads, massflows, temperatures)
Bases: cea.demand.demand_writers.DemandWriter

Write out the monthly demand results

__init__ (loads, massflows, temperatures)
    x.__init__(...) initializes x; see help(type(x)) for signature

calc_monthly_dataframe (building_name, hourly_data)
write_to_csv (building_name, columns, hourly_data, locator)
write_to_hdf5 (building_name, columns, hourly_data, locator)

class cea.demand.demand_writers.YearlyDemandWriter (loads, massflows, temperatures)
Bases: cea.demand.demand_writers.DemandWriter

Write out the hourly demand results

__init__ (loads, massflows, temperatures)
    x.__init__(...) initializes x; see help(type(x)) for signature

write_to_csv (list_buildings, locator)
    read in the temporary results files and append them to the Totals.csv file.

write_to_hdf5 (list_buildings, locator)
    read in the temporary results files and append them to the Totals.csv file.

demand.electrical_loads module

Electrical loads

demand.electrical_loads.calc_E_sys (tsd)
    Calculate the compound of end use electrical loads

demand.electrical_loads.calc_Eal_Epro (tsd, 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
• schedules (List[numpy.ndarray]) – The list of schedules defined for the project
    - in the same order as list_uses

Returns tsd with new keys: ['Eaf', 'Elf', 'Ealf']

Return type: Dict[str, numpy.ndarray]

demand.electrical_loads.calc_Eaux (tsd)
    Calculate the compound of final electricity loads with contain the end-use demand,

demand.electrical_loads.calc_Eaux_Qhs_Qcs (tsd, bpr)
    Auxiliary electric loads from Legacy Following EN 15316-3-2:2007 Annex F

Parameters

• tsd (dict) – Time series data of building
- **bpr** ([`cea.demand.thermal_loads.BuildingPropertiesRow`]) – Building Properties Row object

**Returns**

- `cea.demand.electrical_loads.calc_Eaux_fw(tsd, bpr, schedules)`
- `cea.demand.electrical_loads.calc_Eaux_ww(tsd, bpr)`
- `cea.demand.electrical_loads.calc_Eauxf_cs_dis(Qcs_sys, Qcs_sys0, deltaP_kPa, b, ts, tr)`
- `cea.demand.electrical_loads.calc_Eauxf_fw(Vfw_m3h, deltaP_kPa, b)`  #Following EN 15316-3-2:2007 Annex F :param Vfw_m3h: :param nf: :return:
- `cea.demand.electrical_loads.calc_Eauxf_hs_dis(Qhs_sys, Qhs_sys0, deltaP_kPa, b, ts, tr)`
- `cea.demand.electrical_loads.calc_Eauxf_ve(tsd)`

Calculation of auxiliary electricity consumption of mechanical ventilation and AC fans

**Parameters**

- `tsd(dict)` – Time series data of building

**Returns**

- Electrical energy for fans of mechanical ventilation in [Wh/h]

**Return type** float

- `cea.demand.electrical_loads.calc_Eauxf_ww(Qww, deltaP_kPa, b, m_kgs)`  

- `cea.demand.electrical_loads.calc_Ef(bpr, tsd)`

Calculate the compound of final electricity loads with contain the end-use demand,

**cea.demand.hotwater_loads module**

Hotwater load (it also calculates fresh water needs)

- `cea.demand.hotwater_loads.calc_DH_ww_with_tank_losses(T_ext_C, T_int_C, Qww, Vww, Qww_dis_ls_r, Qww_dis_ls_nr)`

Calculates the heat flows within a fully mixed water storage tank for HOURS_IN_YEAR time-steps. 

**Parameters**

- `T_ext_C` – external temperature in [C]
- `T_int_C` – room temperature in [C]
- `Qww` – hourly DHW demand in [Wh]
- `Vww` – hourly DHW demand in [m3]
- `Qww_dis_ls_r` – recoverable loss in distribution in [Wh]
- `Qww_dis_ls_nr` – non-recoverable loss in distribution in [Wh]

**Return**


- `cea.demand.hotwater_loads.calc_Qww(bpr, tsd, schedules)`

Calculates the DHW demand according to the supply temperature and flow rate. 

**Parameters**

- `mdot_dhw_kgps` – required DHW flow rate in [kg/s]
- `T_dhw_sup_C` – Domestic hot water supply set point temperature.
- `T_dhw_re_C` – Domestic hot water tank return temperature in C, this temperature is the ground water temperature, set according to norm. 
- `Cpw` – heat capacity of water [kJ/kgK]

**Return**

`Q_dhw_W` – Heat demand for DHW in [W]

- `cea.demand.hotwater_loads.calc_Qww_dis_ls_nr(tair, Qww, Lvww_dis, Lvww_c, Y, Qww_0, V, twws, te)`

**Parameters**

- `tair`
- `Qww`
- `Lvww_dis`
Returns

`cea.demand.hotwater_loads.calc_Qww_dis_ls_r(Tair, Qww, Lsww_dis, Lcww_dis, Y, Qww_0, \( V, \text{twws} \))`

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 bpr: Building Properties
- :type bpr: BuildingPropertiesRow
- :param tsd: Time series data of building
- :type tsd: dict
- :return: modifies tsd
- :type: None
- :return: mcp_tap_water_kWperK: tap water capacity mass flow rate in kW_C

`cea.demand.hotwater_loads.calc_Qwwf(bpr, tsd)`

`cea.demand.hotwater_loads.calc_disls(tamb, Vww, V, twws, Lsww_dis, Y)`

Calculates distribution losses in Wh according to Fonseca & Schlueter (2015) Eq. 24, which is in turn based on Annex A of ISO EN 15316 with pipe mass m_p,dis = 0.

Parameters

- `tamb` – Room temperature in C
- `Vww` – volumetric flow rate of hot water demand (in m3)
- `V` – volume of water accumulated in the distribution network in m3
- `twws` – Domestic hot water supply set point temperature in C
- `Lsww_dis` – length of circulation/distribution pipeline in m
- `p` – water density kg/m3
- `cpw` – heat capacity of water in kJ/kgK
- `Y` – linear trasmissivity coefficient of piping in distribution network in W/m*K

Return losses recoverable/non-recoverable losses due to distribution of DHW

`cea.demand.hotwater_loads.calc_mww(schedule, water_lpd)`

Algorithm to calculate the hourly mass flow rate of water

Parameters

- `schedule` – hourly DHW demand profile [person/d.h]
- `water_lpd` – water demand per person per day in [L/person/day]

`cea.demand.hotwater_loads.calc_water_temperature(T_ambient_C, depth_m)`


`cea.demand.hotwater_loads.has_hot_water_technical_system(bpr)`

Checks if building has a hot water system
Parameters \( \text{bpr} \) (\( \text{cea.demand.building_properties.BuildingPropertiesRow} \)) – BuildingPropertiesRow

Returns True or False

Return type \( \text{bool} \)

c.\text{ea.demand.hourly_procedure_heating_cooling_system_load module}

c.\text{ea.demand.hourly_procedure_heating_cooling_system_load.calc_cool_loads_3for2} (\( \text{bpr} \), \( \text{t} \), \( \text{tsd} \))

Calculation procedure for cooling system loads of AHU, ARU and SCU subsystems of 3for2 system
Gabriel Happle, Feb. 2018

Parameters
- \( \text{bpr} \) – building properties row object
- \( \text{t} \) – time step / hour of year \([0..8760]\)
- \( \text{tsd} \) – time series data dict

Returns \( \text{dict of rc_model_temperatures} \)

c.\text{ea.demand.hourly_procedure_heating_cooling_system_load.calc_cool_loads_central_ac} (\( \text{bpr} \), \( \text{t} \), \( \text{tsd} \))

Calculation procedure for cooling system loads of AHU and ARU subsystems of a central AC system
Gabriel Happle, Feb. 2018

Parameters
- \( \text{bpr} \) – building properties row object
- \( \text{t} \) – time step / hour of year \([0..8760]\)
- \( \text{tsd} \) – time series data dict

Returns

c.\text{ea.demand.hourly_procedure_heating_cooling_system_load.calc_cool_loads_mini_split_ac} (\( \text{bpr} \), \( \text{t} \), \( \text{tsd} \))

Calculation procedure for cooling system loads of an ARU subsystem of a mini-split AC system

Parameters
- \( \text{bpr} \) – building properties row object
- \( \text{t} \) – time step / hour of year \([0..8760]\)
- \( \text{tsd} \) – time series data dict

Returns

c.\text{ea.demand.hourly_procedure_heating_cooling_system_load.calc_cool_loads_radiator} (\( \text{bpr} \), \( \text{t} \), \( \text{tsd} \))

Procedure for hourly cooling system load calculation for a building with a radiative cooling system.
Gabriel Happle, February 2018

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

**Returns**

`cea.demand.hourly_procedure_heating_cooling_system_load.calc_heat_loads_central_ac(bpr, t, tsd)`

Procedure for hourly heating system load calculation for a building with a central AC heating system.

Gabriel Happle, February 2018

**Parameters**

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

**Returns**

`cea.demand.hourly_procedure_heating_cooling_system_load.calc_heat_loads_radiator(bpr, t, tsd)`

Procedure for hourly heating system load calculation for a building with a radiative heating system.

Gabriel Happle, February 2018

**Parameters**

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

**Returns**

`cea.demand.hourly_procedure_heating_cooling_system_load.calc_heating_cooling_loads(bpr, tsd, t)`

**Parameters**

• **bpr** –
• **tsd** –
• **t** –

**Returns**

`cea.demand.hourly_procedure_heating_cooling_system_load.calc_rc_cooling_demand(bpr, tsd, t)`

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]

**Returns** phi_c_act, rc_model_temperatures

`cea.demand.hourly_procedure_heating_cooling_system_load.calc_rc_heating_demand(bpr, tsd, t)`

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]

**Returns** phi_h_act, rc_model_temperatures

`cea.demand.hourly_procedure_heating_cooling_system_load.calc_rc_no_loads(bpr, tsd, t)`

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]

**Returns** dict of rc_model_temperatures

`cea.demand.hourly_procedure_heating_cooling_system_load.detailed_thermal_balance_to_tsd(tsd, bpr, t, rc_model_temperatures)`

Back calculate energy flows in RC model for dashboard of energy balance visualization

**Parameters**

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

**Returns** None

`cea.demand.hourly_procedure_heating_cooling_system_load.rc_temperatures_to_tsd(rc_model_temperatures, tsd, t)`
cea.demand.hourly_procedure_heating_cooling_system_load.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.hourly_procedure_heating_cooling_system_load.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.latent_loads module

cea.demand.latent_loads.calc_Qgain_lat(tsd, schedules)

Parameters
- \( schedules \) (list[ndarray[float]]) – The list of schedules defined for the project - in the same order as list_uses

Returns \( w_{\text{int}} \) yearly schedule

cea.demand.latent_loads.calc_dehumidification_moisture_load(bpr, tsd, t)

(72) in ISO 52016-1:2017

Gabriel Happle, Feb. 2018

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

Returns dehumidification load (kg/s)

Return type double

cea.demand.latent_loads.calc_humidification_moisture_load(bpr, tsd, t)

(71) in ISO 52016-1:2017

Gabriel Happle, Feb. 2018

Parameters
- \( bpr \) (BuildingPropertiesRow) – Building Properties
- \( tsd \) (dict) – Time series data of building
- \( t \) (int) – time step / hour of the year
Returns humidification load (kg/s)

Return type double

tsa.demand.latent_loads.calc_max_moisture_set_point(bpr, tsd, t)

(76) in ISO 52016-1:2017

Gabriel Happle, Feb. 2018

Parameters

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

Returns max moisture set point (kg/kg_dry_air)

Return type double

tsa.demand.latent_loads.calc_min_moisture_set_point(bpr, tsd, t)

(75) in ISO 52016-1:2017

Gabriel Happle, Feb. 2018

Parameters

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

Returns min moisture set point (kg/kg_dry_air)

Return type double

tsa.demand.latent_loads.calc_moisture_content_airflows(tsd, t)

calculate relative humidity of ventilation airflows to moisture content

Gabriel Happle, Feb. 2018

Parameters

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

Returns adds moisture content of ventilation air flows to tsd

Return type None

tsa.demand.latent_loads.calc_moisture_content_in_zone_local(bpr, tsd, t)

(84) in ISO 52016-1:2017

Gabriel Happle, Feb. 2018

Parameters

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

Returns writes zone internal moisture content to tsd
Return type  None

```python
c ea.demand.latent_loads.calc_moisture_in_zone_central(bpr, tsd, t)
```

Gabriel Happle, Feb. 2018

Parameters

- `bpr`  \((\text{BuildingPropertiesRow})\) – Building Properties
- `tsd`  \((\text{dict})\) – Time series data of building
- `t`  \((\text{int})\) – time step / hour of the year

Returns  dehumidification load (kg/s)

Return type  double

```python
c ea.demand.latent_loads.calc_required_moisture_mech_vent_dhu(tsd, t)
c ea.demand.latent_loads.calc_required_moisture_mech_vent_hu(tsd, t)
c ea.demand.latent_loads.calc_saturation_pressure(theta)
```

(77) in ISO 52016-1:2017

Gabriel Happle, Feb. 2018

Parameters

- `theta`  \((\text{double})\) – air temperature (C)

Returns  saturation pressure (Pa)

Return type  double

```python
c ea.demand.latent_loads.convert_rh_to_moisture_content(rh, theta)
```

convert relative humidity to moisture content

Gabriel Happle, Feb. 2018

Parameters

- `rh`  \((\text{double})\) – relative humidity (%)
- `theta`  \((\text{double})\) – temperature (C)

Returns  moisture content (kg/kg\_dry\_air)

Return type  double

```python
c ea.demand.latent_loads.total_moisture_in_zone(bpr, x_int)
```

calculate total mass of moisture in zone

Gabriel Happle, Feb. 2018

Parameters

- `bpr`  \((\text{BuildingPropertiesRow})\) – Building Properties
- `x_int`  \((\text{double})\) – moisture content in zone (kg/kg\_dry\_air)

Returns  total mass of moisture in zone (kg)

Return type  double
cea.demand.rc_model_SIA module

cea.demand.rc_model_SIA._calc_rc_model_temperatures(Eaf, Elf, Epro, Htr_op, Htr_w, I_sol, Qs, T_ext, a_m, a_t, a_w, c_m, m_ve_inf_simple, m_ve_mech, m_ve_window, phi_hc_cv, phi_hc_r, theta_m_t_1, theta_ve_mech)

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

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_ic(a_t, a_m, h_ec)

Parameters

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

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_l, 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, Epro)
cea.demand.rc_model_SIA.calc_phi_i_l(Elf)
cea.demand.rc_model_SIA.calc_phi_i_p(Qs)
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

- \texttt{phi\_hc} (float) – Heating or cooling energy demand of building
- \texttt{bpr} (BuildingPropertiesRow) – Building Properties
- \texttt{tsd} (dict) – Time series data of building
- \texttt{t} (int) – time step / hour of the year

Returns R-C-Building-Model node temperatures
Return type  
\texttt{dict}

\texttt{cea.demand.rc\_model\_SIA.calc\_rc\_model\_temperatures\_heating}(\texttt{phi\_hc}, \texttt{bpr}, \texttt{tsd}, \texttt{t})

This function executes the equations of SIA 2044 R-C-Building-Model to calculate the node temperatures for a given heating energy demand

\texttt{Py:func  cea.demand.rc\_model\_SIA.lookup\_f\_hc\_cv\_heating}
\texttt{Py:func  cea.demand.rc\_model\_SIA.calc\_phi\_hc\_cv}
\texttt{Py:func  cea.demand.rc\_model\_SIA.calc\_phi\_hc\_r}
\texttt{Py:func  cea.demand.rc\_model\_SIA.calc\_rc\_model\_temperatures}

Author: Gabriel Happle Date: FEB 2017

Parameters

- \texttt{phi\_hc (float)} – Heating or cooling energy demand of building
- \texttt{bpr (BuildingPropertiesRow)} – Building Properties
- \texttt{tsd (dict)} – Time series data of building
- \texttt{t (int)} – time step / hour of the year

Returns  R-C-Building-Model node temperatures

Return type  \texttt{dict}

\texttt{cea.demand.rc\_model\_SIA.calc\_rc\_model\_temperatures\_no\_heating\_cooling}(\texttt{bpr}, \texttt{tsd}, \texttt{t})

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

\texttt{Py:func  cea.demand.rc\_model\_SIA.calc\_rc\_model\_temperatures\_no\_heating\_cooling}

Author: Gabriel Happle Date: FEB 2017

Parameters

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

Returns  R-C-Model node temperatures

Return type  \texttt{dict}

\texttt{cea.demand.rc\_model\_SIA.calc\_theta\_c(\texttt{phi\_a}, \texttt{phi\_c}, \texttt{theta\_ea}, \texttt{theta\_ec}, \texttt{theta\_m}, \texttt{h\_1}, \texttt{h\_mc}, \texttt{h\_ec}, \texttt{h\_ea})}

\texttt{cea.demand.rc\_model\_SIA.calc\_theta\_e\_star()}

\texttt{cea.demand.rc\_model\_SIA.calc\_theta\_ea(\texttt{m\_ve\_mech}, \texttt{m\_ve\_window}, \texttt{m\_ve\_inf\_simple}, \texttt{theta\_ve\_mech}, \texttt{T\_ext})}

\texttt{cea.demand.rc\_model\_SIA.calc\_theta\_ec(\texttt{T\_ext})}

\texttt{cea.demand.rc\_model\_SIA.calc\_theta\_em(\texttt{T\_ext})}

\texttt{cea.demand.rc\_model\_SIA.calc\_theta\_m(\texttt{theta\_m\_t}, \texttt{theta\_m\_t\_1})}

\texttt{cea.demand.rc\_model\_SIA.calc\_theta\_m\_t(\texttt{phi\_m\_tot}, \texttt{theta\_m\_t\_1}, \texttt{h\_em}, \texttt{h\_3}, \texttt{c\_m})}

\texttt{cea.demand.rc\_model\_SIA.calc\_theta\_o(\texttt{T\_int}, \texttt{theta\_c})}
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.

\textbf{Py:func} \texttt{cea.demand.rc_model_SIA.calc_rc_model_temperatures_no_heating_cooling}

Author: Gabriel Happle Date: FEB 2017

Parameters

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

Returns True or False

Return type \texttt{bool}

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.

\textbf{Py:func} \texttt{cea.demand.rc_model_SIA.calc_rc_model_temperatures_no_heating_cooling}

Author: Gabriel Happle Date: FEB 2017

Parameters

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

Returns True or False

Return type \texttt{bool}

\texttt{cea.demand.rc_model_SIA.lookup_f_hc_cv_cooling} \texttt{(bpr)}

\texttt{cea.demand.rc_model_SIA.lookup_f_hc_cv_heating} \texttt{(bpr)}

c\texttt{ea.demand.refrigerationLoads\_module}

refrigeration loads

\texttt{cea.demand.refrigerationLoads\_calc\_Q\text{\_}cre\_sys} \texttt{(bpr, tsd, schedules)}

\texttt{cea.demand.refrigerationLoads\_calc\_Q\text{\_}ref} \texttt{(locator, bpr, tsd)}

\texttt{it calculates final loads}

\texttt{cea.demand.refrigerationLoads\_calc\_refrigeration\_temperature\_and\_massflow} \texttt{(Q\text{\_}cre\_sys)}

Calculate refrigeration supply and return temperatures and massflows based on the refrigeration load. This function is intended to be used in np.vectorize form.

\textbf{param Q\text{\_}cre\_sys: refrigeration load including losses} \textbf{return:}

refrigeration massflow, refrigeration supply temperature, refrigeration return temperature

9.1. \texttt{cea package}
cea.demand.refrigeration_loads.\textcolor{red}{\texttt{has\_refrigeration\_load}}(bpr)

Checks if building has a hot water system

\textbf{Parameters} \hspace{10pt} bpr (cea.demand.building_properties.BuildingPropertiesRow) – BuildingPropertiesRow

\textbf{Returns} True or False

\textbf{Return type} \hspace{10pt} bool

\texttt{cea.demand.sensible\_loads} module

Sensible space heating and space cooling loads EN-13970

cea.demand.sensible_loads.\texttt{calc\_I\_rad}(t, tsd, bpr)

This function calculates the solar radiation re-irradiated from a building to the sky according to ISO 13790 See Eq. (46) in 11.3.5

\textbf{Parameters}

• \texttt{t} – hour of the year
• \texttt{tsd} – time series dataframe
• \texttt{bpr} – building properties object

\textbf{Returns} \texttt{I\_rad}: vector of solar radiation re-irradiated to the sky.

cea.demand.sensible_loads.\texttt{calc\_I\_sol}(t, bpr, tsd)

This function calculates the net solar radiation (incident - reflected - re-irradiated) according to ISO 13790 see Eq. (23) in 11.3.1

\textbf{Parameters}

• \texttt{t} – hour of the year
• \texttt{bpr} – building properties object
• \texttt{tsd} – time series dataframe

\textbf{Returns} \texttt{I\_sol\_net}: vector of net solar radiation to the building
\texttt{I\_rad}: vector of solar radiation re-irradiated to the sky.
\texttt{I\_sol\_gross}: vector of incident radiation to the building.

cea.demand.sensible_loads.\texttt{calc\_Qgain\_sen}(t, tsd, bpr)

cea.demand.sensible_loads.\texttt{calc\_Qhs\_Qcs\_loss}(bpr, tsd)

Calculate distribution losses of emission systems.

\textbf{Modified from legacy:} calculates distribution losses based on ISO 15316

Gabriel Happle, Feb. 2018

\textbf{Parameters}

• \texttt{bpr} (BuildingPropertiesRow) – Building Properties
• \texttt{tsd} (dict) – Time series data of building

\textbf{Returns} modifies \texttt{tsd}

\textbf{Return type} \hspace{10pt} None

cea.demand.sensible_loads.\texttt{calc\_Qhs\_Qcs\_sys\_max}(Af, prop\_HVAC)

cea.demand.sensible_loads.\texttt{calc\_Qhs\_sys\_Qcs\_sys}(tsd)
The function calculates the external radiative heat transfer coefficient according to ISO 13790 see Eq. (51) in section 11.4.6

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

**Returns**
- hr:

The function calculates the temperature of emission systems. Using radiator function also for cooling ('radiators.calc_radiator')

**Parameters**
- **bpr** (BuildingPropertiesRow) – Building Properties
- **tsd** (dict) – Time series data of building

**Returns**
- modifies tsd

**Return type**
- None

---

**cea.demand.set_point_from_predefined_file module**

tsd, bpr, weekday, building_name, locator

This function is used to set the space cooling and space heating set points from a predefined hourly set points xlsx file. This will help in finding the building demand for a flexible building scenario.

This script is developed as part of the CONCEPT project (an Intra-CREATE collaboration between FCL and TUM CREATE)

The excel files with the predefined hourly temperatures are to be stored in scenarioinputs predefined-hourly-setpoints folder. This folder in turn has two subfolders namely space-heating and space-cooling. Inside these folders the setpoints are to be provided in an excel file. The excel files need to be provided in Building-Name_temperature.xlsx format and the necessary columns are time and temperature.

In case the files are not present, the calculations will still carry on by taking the archetypical set points present in CEA databases. Though it will print a message saying predefined set points file is not provided.

**Parameters**
- **tsd** (dict) – a dictionary of time step data mapping variable names to ndarrays for each hour of the year.
- **bpr** (cea.demand.building_properties.BuildingPropertiesRow) – BuildingPropertiesRow
- **weekday** –
Returns  tsd with updated columns

Return type  dict

ccea.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(bpr)

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  bpr (BuildingPropertiesRow object) – BuildingPropertiesRow

Returns  delta T to correct the set point temperature for cooling

Return type  double

cea.demand.space_emission_systems.calc_delta_theta_int_inc_heating(bpr)

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  bpr (BuildingPropertiesRow object) – BuildingPropertiesRow

Returns  delta T to correct the set point temperature for heating

Return type  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 (double) – heating power of emission system (W)
•  delta_theta_int_inc (double) – delta temperature caused by all losses (K)
•  theta_int_inc (double) – equivalent room temperature (°C)
•  theta_e_comb (double) – ?comb? outdoor temperature (°C)
• \texttt{q\_em\_max(double)} – maximum emission capacity of heating/cooling system [W]

Returns emission losses of heating/cooling system [W]

Return type double

\texttt{cea.demand.space\_emission\_systems.calc\_q\_em\_ls\_cooling(bpr, tsd, t)}
calculation procedure for space emissions losses in the cooling case [prEN 15316-2:2014]

Parameters

• \texttt{bpr(BuildingPropertiesRow object)} – building properties row

• \texttt{tsd(dict)} – time step data

• \texttt{t(int)} – hour of year (0..8759)

Returns emission losses of cooling system for time step [W]

Return type double

\texttt{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]

Parameters

• \texttt{bpr(BuildingPropertiesRow object)} – building properties row

• \texttt{tsd(dict)} – time step data

• \texttt{hoy(int)} – hour of year (0..8759)

Returns emission losses of heating system for time step [W]

Return type double

\texttt{cea.demand.space\_emission\_systems.calc\_theta\_e\_comb\_cooling(theta\_e, bpr)}
Eq. (10) in [prEN 15316-2:2014]

Parameters

• \texttt{theta\_e(double)} – outdoor temperature [C]

• \texttt{bpr(BuildingPropertiesRow object)} – BuildingPropertiesRow

Returns temperature [C]

Return type double

\texttt{cea.demand.space\_emission\_systems.calc\_theta\_e\_comb\_heating(theta\_e)}
Eq. (9) in [prEN 15316-2:2014]

Parameters \texttt{theta\_e(double)} – outdoor temperature [C]

Returns temperature [C]

Return type double

\texttt{cea.demand.space\_emission\_systems.calc\_theta\_int\_inc(theta\_int\_ini, delta\_theta\_int\_inc)}
Eq. (1) in [prEN 15316-2:2014]

Parameters

• \texttt{theta\_int\_ini(double)} – temperature [C]

• \texttt{delta\_theta\_int\_inc(double)} – temperature [C]

Returns sum of temperatures [C]
Return type double

```
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**

Demand model of thermal loads

```
cea.demand.thermal_loads.calc_QH_sys_QC_sys(tsd)
```

```
cea.demand.thermal_loads.calc_Qcs_sys(bpr, tsd)
```

```
cea.demand.thermal_loads.calc_Qhs_Qcs(bpr, tsd, use_dynamic_infiltration_calculation)
```

```
cea.demand.thermal_loads.calc_Qhs_sys(bpr, tsd)
```

it calculates final loads

```
cea.demand.thermal_loads.calc_set_points(bpr, date, tsd, building_name, config, locator, schedules)
```

```
cea.demand.thermal_loads.calc_thermal_loads(building_name, bpr, weather_data, date_range, locator, use_dynamic_infiltration_calculation, resolution_outputs, loads_output, mass-flows_output, temperatures_output, config, debug)
```

Calculate thermal loads of a single building with mechanical or natural ventilation. Calculation procedure follows the methodology of ISO 13790

The structure of usage_schedules is:

```
{
    'list_uses': ['ADMIN', 'GYM', ...],
    'schedules': [ ([...], [...], [...], [...]), (), (), () ]
}
```

- each element of the ‘list_uses’ entry represents a building occupancy type.
- each element of the ‘schedules’ entry represents the schedules for a building occupancy type.
- the schedules for a building occupancy type are a 4-tuple (occupancy, electricity, domestic hot water, probability of use), with each element of the 4-tuple being a list of hourly values (HOURS_IN_YEAR values).  

Side effect include a number of files in two folders:

- scenario/outputs/data/demand
  - ${Name}.csv for each building
- temporary folder (as returned by tempfile.gettempdir())
  - ${Name}T.csv for each building
daren-thomas: as far as I can tell, these are the only side-effects.

Parameters

- **building_name** (*str*) – name of building
- **bpr** (*BuildingPropertiesRow*) – a collection of building properties for the building used for thermal loads calculation
- **weather_data** (*pandas.DataFrame*) – data from the .epw weather file. Each row represents an hour of the year. The columns are: `drybulb_C`, `relhum_percent`, and `windspd_ms`
- **locator** –
- **use_dynamic_infiltration_calculation** –

Returns This function does not return anything

Return type NoneType

cfa.demand.thermal_loads.get_hours(bpr)

Parameters **bpr** – BuildingPropertiesRow

Returns

cfa.demand.thermal_loads.initialize_inputs(bpr, weather_data, locator)

Parameters

- **bpr** (*BuildingPropertiesRow*) – a collection of building properties for the building used for thermal loads calculation
- **weather_data** (*pandas.DataFrame*) – data from the .epw weather file. Each row represents an hour of the year. The columns are: `drybulb_C`, `relhum_percent`, and `windspd_ms`
- **date_range** (*pd.date_range*) – the pd.date_range of the calculation year
- **locator** (*cea.inpultlocator.InputLocator*) – the input locator

Returns one dict of schedules, one dict of time step data

Return type dict

cfa.demand.thermal_loads.initialize_timestep_data(bpr, weather_data)

initializes the time step data with the weather data and the minimum set of variables needed for computation.

Parameters

- **bpr** (*BuildingPropertiesRow*) – a collection of building properties for the building used for thermal loads calculation
- **weather_data** (*pandas.DataFrame*) – data from the .epw weather file. Each row represents an hour of the year. The columns are: `drybulb_C`, `relhum_percent`, and `windspd_ms`

Returns returns the `tsd` variable, a dictionary of time step data mapping variable names to ndarrays for each hour of the year.

Return type dict

cfa.demand.thermal_loads.update_timestep_data_no_conditioned_area(tsd)

Update time step data with zeros for buildings without conditioned area

Author: Gabriel Happle Date: 01/2017
**Parameters**

tsd – time series data dict

**Returns**
update tsd

cEA.demand.thermal_loads.write_results(bpr, building_name, date, loads_output, locator, massflows_output, resolution_outputs, temperatures_output, tsd, debug)

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 (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)

```python
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)

```python
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)

```python
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)

```python
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)

```python
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)

```python
cea.demand.ventilation_air_flows_detailed.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:

<table>
<thead>
<tr>
<th>angle_window</th>
<th>area_window</th>
<th>height_window_above_ground</th>
<th>1.910858</th>
<th>1.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>2.276739</td>
<td></td>
<td></td>
<td>4.5</td>
</tr>
<tr>
<td>90</td>
<td>2.276739</td>
<td></td>
<td>7.5</td>
<td>10.5</td>
</tr>
<tr>
<td>90</td>
<td>2.276739</td>
<td></td>
<td>10.5</td>
<td>1.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>height_window_in_zone</th>
<th>name_building</th>
<th>orientation_window</th>
<th>1.5</th>
<th>180</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
<td>B140589</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>B140590</td>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>B140590</td>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.5</td>
<td>B140590</td>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.5</td>
<td>B140590</td>
<td>180</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**cea.demand.ventilation_air_flows_detailed.get_building_geometry_ventilation** *(gdf_building_geometry)*

: **param** gdf_building_geometry : GeoDataFrame contains single building

**Returns** building properties for natural ventilation calculation

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

gdf_geometry_building : GeoDataFrame containing geometry properties of single building
gdf_architecture_building : GeoDataFrame containing architecture props of single building

: **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_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** (cea.demand.thermal_loads.BuildingPropertiesRow) – Building properties row object
- **tsd** (Dict[str, numpy.ndarray]) – Timestep data
- **t** (int) – time step [0..HOURS_IN_YEAR]

**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** (cea.demand.thermal_loads.BuildingPropertiesRow) – Building properties row object
- **tsd** (Dict[str, numpy.ndarray]) – Timestep data
- **t** (int) – time step [0..HOURS_IN_YEAR]

**Returns** updates tsd

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

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** (cea.demand.thermal_loads.BuildingPropertiesRow) – Building properties row object
- **tsd** (Dict[str, numpy.ndarray]) – Timestep data
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 `tsd (Dict[str, numpy.ndarray])` - Timestep data

Returns updates tsd

`cea.demand.ventilation_air_flows_simple.calc_theta_ve_mech(bpr, tsd, t)`

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

Author: Gabriel Happle Date: 01/2017

Parameters

- `bpr (cea.demand.thermal_loads.BuildingPropertiesRow)` - Building properties row object
- `tsd (Dict[str, numpy.ndarray])` - Timestep data
- `t (int)` - time step [0..HOURS_IN_YEAR]

Returns updates tsd

**cea.examples package**

**Submodules**

**cea.examples.extract_reference_case module**

Extract the reference case (`cea/examples/reference-case-open.zip`).

`cea.examples.extract_reference_case.main(config)`

Extract the reference case in `reference-case-open.zip` to the destination folder.

Parameters `config (cea.config.Configuration)` - Contains the PathParameter

    config.extract_reference_case.destination

Returns

**cea.examples.template module**

This is a template script - an example of how a CEA script should be set up.

NOTE: ADD YOUR SCRIPT’S DOCUMENTATION HERE (what, why, include literature references)

`cea.examples.template.main(config)`

This is the main entry point to your script. Any parameters used by your script must be present in the `config` parameter. The CLI will call this `main` function passing in a `config` object after adjusting the configuration to reflect parameters passed on the command line - this is how the ArcGIS interface interacts with the scripts.

Parameters `config (cea.config.Configuration)`

Returns
this is where the action happens if it is more than a few lines in main. NOTE: ADD YOUR SCRIPT’S DOCUMENTATION HERE (how) NOTE: RENAME THIS FUNCTION (SHOULD PROBABLY BE THE SAME NAME AS THE MODULE)

cea.interfaces package

Subpackages

cea.interfaces.arcgis package

Submodules

cea.interfaces.arcgis.CityEnergyAnalyst module

ArcGIS Toolbox for integrating the CEA with ArcGIS.

ArcGIS starts by creating an instance of Toolbox, which in turn names the tools to include in the interface.

These tools shell out to cli.py because the ArcGIS python version is old and can’t be updated. Therefore we would decouple the python version used by CEA from the ArcGIS version.

See the script install_toolbox.py for the mechanics of installing the toolbox into the ArcGIS system.

class cea.interfaces.arcgis.CityEnergyAnalyst.DemandTool
    Bases: cea.interfaces.arcgis.arcgishelper.CeaTool

    integrate the demand script with ArcGIS

    __init__(
        x.__init__(...) initializes x; see help(type(x)) for signature

    override_parameter_info (parameter_info, parameter)
        Override this method if you need to use a non-default ArcGIS parameter handling

class cea.interfaces.arcgis.CityEnergyAnalyst.RadiationDaysimTool
    Bases: cea.interfaces.arcgis.arcgishelper.CeaTool

    __init__(
        x.__init__(...) initializes x; see help(type(x)) for signature

    override_parameter_info (parameter_info, parameter)
        Override this method if you need to use a non-default ArcGIS parameter handling

class cea.interfaces.arcgis.CityEnergyAnalyst.Toolbox
    Bases: object

    List the tools to show in the toolbox.

    __init__(
        x.__init__(...) initializes x; see help(type(x)) for signature

    generate_tools()

cea.interfaces.arcgis.arcgishelper module

A library module with helper functions for creating the City Energy Analyst python toolbox for ArcGIS.
class cea.interfaces.arcgis.arcgishelper.BuildingsParameterInfoBuilder(cea_parameter, config)

encode_value(cea_parameter, parameter)

get_parameter_info()

on_update_parameters(parameter_name, parameters)
    Called each time the parameters are changed (except for first time, on_dialog_show). Subclasses can use this to customize behavior.

class cea.interfaces.arcgis.arcgishelper.CeaTool

Bases: object

A base class for creating tools in an ArcGIS toolbox. Basically, the user just needs to subclass this, specify the usual ArcGIS stuff in the __init__ method as well as set self.cea_tool to the corresponding tool name. The rest is auto-configured based on default.config and scripts.yml

__init__(cea_script)
    Allow initialization from the create_cea_tool

disable_Identifies()

execute(parameters, _)

getParameterInfo()
    Return the list of arcgis Parameter objects for this tool. The general:weather parameter is treated specially: it is represented as two parameter_infos, weather_name and weather_path.

override_parameter_info(parameter_info, parameter)
    Override this method if you need to use a non-default ArcGIS parameter handling

updateMessages(parameters)
    Give the builders a chance to update messages / perform some validation

updateParameters(parameters)

class cea.interfaces.arcgis.arcgishelper.ChoiceParameterInfoBuilder(cea_parameter, config)

get_parameter_info()

class cea.interfaces.arcgis.arcgishelper.FileParameterInfoBuilder(cea_parameter, config)

get_parameter_info()

class cea.interfaces.arcgis.arcgishelper.ListParameterInfoBuilder(cea_parameter, config)

get_parameter_info()

class cea.interfaces.arcgis.arcgishelper.MultiChoiceParameterInfoBuilder(cea_parameter, config)

get_parameter_info()
class `cea.interfaces.arcgis.arcgishelper.OptimizationIndividualListParameterInfoBuilder` (cea_parameter, config)

Bases: `cea.interfaces.arcgis.arcgishelper.ParameterInfoBuilder`

code_value (cea_parameter, parameter)
get_filters (project_path)
get_parameter_info()
on_dialog_show (parameter_name, parameters)

Build a nested list of the values
on_update_messages (parameter_name, parameters)

Make sure all the values are valid
on_update_parameters (parameter_name, parameters)

Called each time the parameters are changed (except for first time, on_dialog_show). Subclasses can use this to customize behavior.

class `cea.interfaces.arcgis.arcgishelper.OptimizationIndividualParameterInfoBuilder` (cea_parameter, config)

Bases: `cea.interfaces.arcgis.arcgishelper.ParameterInfoBuilder`

code_value (cea_parameter, parameter)
get_parameter_info()
on_dialog_show (parameter_name, parameters)
on_update_parameters (parameter_name, parameters)

Update the parameter value with the values of the additional dropdowns, setting their filters appropriately.

class `cea.interfaces.arcgis.arcgishelper.ParameterInfoBuilder` (cea_parameter, config)

Bases: `object`

A base class for building arcpy.Parameter objects based on `cea.config.Parameter` objects.

__init__ (cea_parameter, config)
x.__init__(...) initializes x; see help(type(x)) for signature
code_value (cea_parameter, parameter)
get_parameter_info()
on_dialog_show (parameter_name, parameters)
on_update_messages (parameter_name, parameters)

Called for each cea parameter during updateMessages. Subclasses may want to use this to customize behavior.
on_update_parameters (parameter_name, parameters)

Called each time the parameters are changed (except for first time, on_dialog_show). Subclasses can use this to customize behavior.

class `cea.interfaces.arcgis.arcgishelper.PathParameterInfoBuilder` (cea_parameter, config)

Bases: `cea.interfaces.arcgis.arcgishelper.ParameterInfoBuilder`

code_value (cea_parameter, parameter)
get_parameter_info()
class `cea.interfaces.arcgis.arcgishelper.ScalarParameterInfoBuilder` (cea_parameter, config)

Bases: `cea.interfaces.arcgis.arcgishelper.ParameterInfoBuilder`

DATA_TYPE_MAP = {
    `<class 'cea.config.BooleanParameter'>`: 'GPBoolean',
    `<class 'cea.config.IntegerParameter'>`: 'GPLong',
    ...'

get_parameter_info()
get_value()

class `cea.interfaces.arcgis.arcgishelper.ScenarioParameterInfoBuilder` (cea_parameter, config)

Bases: `cea.interfaces.arcgis.arcgishelper.ParameterInfoBuilder`

get_parameter_info()

class `cea.interfaces.arcgis.arcgishelper.SingleBuildingParameterInfoBuilder` (cea_parameter, config)

Bases: `cea.interfaces.arcgis.arcgishelper.ChoiceParameterInfoBuilder`

class `cea.interfaces.arcgis.arcgishelper.StringParameterInfoBuilder` (cea_parameter, config)

Bases: `cea.interfaces.arcgis.arcgishelper.ParameterInfoBuilder`

get_parameter_info()
get_value()

class `cea.interfaces.arcgis.arcgishelper.SubfoldersParameterInfoBuilder` (cea_parameter, config)

Bases: `cea.interfaces.arcgis.arcgishelper.ParameterInfoBuilder`

encode_value (cea_parameter, parameter)

get_parameter_info()

cea.interfaces.arcgis.arcgishelper.add_message (msg, **kwargs)
Log to arcpy.AddMessage() instead of print to STDOUT

ccea.interfaces.arcgis.arcgishelper.check_radiation_exists (parameters, scenario)
Make sure the radiation files exist.

ccea.interfaces.arcgis.arcgishelper.check_scenario_exists (parameters)
Makes sure the scenario exists. Create a dictionary of the parameters at the same time

ccea.interfaces.arcgis.arcgishelper.create_cea_tool (cea_script)
Create a subclass of CeaTool based on the information in the :py:param 'cea_script'

cc weave.interfaces.arcgis.arcgishelper.create_weather_parameters (config)
Create the weather_name and weather_path parameters used for choosing the weatherfile.

ccea.interfaces.arcgis.arcgishelper.demand_graph_fields (scenario)
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).

ccea.interfaces.arcgis.arcgishelper.dict_parameters (parameters)

ccea.interfaces.arcgis.arcgishelper.get_cea_parameters (config, cea_tool)
Return a list of cea.config.Parameter objects for each cea_parameter associated with the tool.

ccea.interfaces.arcgis.arcgishelper.get_db_weather_name (weather_path)
cea.interfaces.arcgis.arcgishelper.get_environment()
Return the system environment to use for the execution - this is based on the location of the python interpreter in get_python_exe

cea.interfaces.arcgis.arcgishelper.get_parameter_info(cea_parameter, config)
Create an arcpy Parameter object based on the configuration in the Default-config. The name is set to “section_name:parameter_name” so parameters created with this function are easily identified (`:` in parameter.name)

cea.interfaces.arcgis.arcgishelper.get_python_exe()
Return the path to the python interpreter that was used to install CEA

cea.interfaces.arcgis.arcgishelper.get_weather_parameter_info(config)
Create two arcpy Parameter objects to deal with the weather

cea.interfaces.arcgis.arcgishelper.get_weather_path_from_parameters(parameters)
Return the path to the weather file to use depending on wether weather_name or weather_path is set by user

cea.interfaces.arcgis.arcgishelper.is_builtin_weather_path(weather_path)
Return True, if the weather path resolves to one of the builtin weather files shipped with the CEA.

cea.interfaces.arcgis.arcgishelper.is_db_weather(weather_path)
True, if the weather_path is one of the pre-installed weather files that came with the CEA

cea.interfaces.arcgis.arcgishelper.list_buildings(scenario)
Shell out to the CEA python and read in the output

cea.interfaces.arcgis.arcgishelper.parse_boolean(s)
Return True or False, depending on the value of s as defined by the ConfigParser library.

cea.interfaces.arcgis.arcgishelper.run_cli(script_name, **parameters)
Run the CLI in a subprocess without showing windows

cea.interfaces.arcgis.arcgishelper.update_weather_parameters(parameters)
Update the weather_name and weather_path parameters

**cea.interfaces.arcgis.install_toolbox module**

Install the toolbox into ArcGIS Desktop 10.4 and 10.5

cea.interfaces.arcgis.install_toolbox.copy_config(toolbox_folder)
Copy the cea/config.py, cea/default.config and an empty __init__.py file to the toolbox_folder

cea.interfaces.arcgis.install_toolbox.copy_inputlocator(toolbox_folder)
Copy the cea/inputlocator.py file to the toolbox_folder and create the cea/databases.pth file

cea.interfaces.arcgis.install_toolbox.copy_library(toolbox_folder, debug=False)
Copy the library functions

cea.interfaces.arcgis.install_toolbox.copy_scripts(toolbox_folder)
Copy the cea/scripts.py and the cea/scripts.pickle files to the toolbox_folder

cea.interfaces.arcgis.install_toolbox.copy_weather_files(toolbox_folder)

cea.interfaces.arcgis.install_toolbox.find_toolbox_destination()
Find the destination path for the toolbox file (City Energy Analyst.pyt) - hint: the folder is similar to “%APPDATA%ESRIDesktop10.4ArcToolboxMy Toolboxes”

cea.interfaces.arcgis.install_toolbox.find_toolbox_src()
Find the source path of the toolbox file (CityEnergyAnalyst.py) - hint: it is relative to the current file!
cea.interfaces.arcgis.install_toolbox.get_arcgis_paths()

Use the windows registry to figure out the paths to the following folders:

• bin
• arcpy
• scripts

as subfolders of the installation directory.

cea.interfaces.arcgis.install_toolbox.get_arcgis_version()

Check the registry for ArcGIS and return the version. Checks the following two locations:

• HKLMsoftwarewow6432NodeesriArcgisRealVersion
• HKLMSOFTWAREESRIArcGISRealVersion

returns the version string as "major.minor", so "10.4" or "10.5"

cea.interfaces.arcgis.install_toolbox.get_cea_dst_folder(toolbox_folder)

Perform the following steps:

• add a link to the python.exe that ran setup.py to user’s home directory in the file cea_python.pth
• copy the file “CityEnergyAnalyst.py” to the “My Toolboxes” folder of ArcGIS Desktop and rename the extension to “.pyt”
• copy cea.config and the default.config to the “My Toolboxes/cea” folder.
• sets up .pth files to access arcpy from the cea python interpreter.
• copy the inputlocator.py file
• create the databases.pth file in the “My Toolboxes/cea” directory

cea.interfaces.arcgis.list_buildings module

List all the buildings in a scenario (using the locator)

The reason for this script is that the ArcGIS interface is running with a different python and therefore we can’t just use that locator - it does not have access to the geopandas library.

cea.interfaces.arcgis.list_buildings.main(scenario)

cea.interfaces.arcgis.modules module

Ensure the ArcGIS modules arcpy and arcgisscripting can be imported properly by adding the paths (as written by the cea install-toolbox command to ~/cea_arcgis.pth) to sys.path.

cea.interfaces.cli package

Submodules

cea.interfaces.cli.cea_config module

Update the user configuration file and show the current settings.
cea.interfaces.cli.cea_config.main(config=None)

Parameters config (cea.config.Configuration) – the configuration file to use (instead of creating a new one)

Returns

cea.interfaces.cli.cea_config.print_help()
Print out the help message for the cea-config tool

cea.interfaces.cli.cea_doc module

Run documentation scripts

cea.interfaces.cli.cea_doc.main(config=None)

Parameters config (cea.config.Configuration) – the configuration file to use (instead of creating a new one)

Returns

cea.interfaces.cli.cea_doc.print_help(config, remaining_args)
Print out the help message for the cea-doc command line interface

cea.interfaces.cli.cea_doc.print_valid_script_names()
Print out the list of scripts by category.

cea.interfaces.cli.cli module

This file implements the cea command line interface script. Basically, it uses the first argument passed to it to look up a module to import in scripts.yml, imports that and then calls the main function on that module. The rest of the command line arguments are passed to the cea.config.Configuration object for processing.

cea.interfaces.cli.cli.main(config=None)

Parameters config (cea.config.Configuration) – the configuration file to use (instead of creating a new one)

Returns

cea.interfaces.cli.cli.print_help(config, remaining_args)
Print out the help message for the cea command line interface

cea.interfaces.cli.cli.print_valid_script_names()
Print out the list of scripts by category.

cea.interfaces.cli.dbf_to_excel module

Use the py:mod:cea.utilities.dbf module to convert a dbf file to an excel file.

cea.interfaces.cli.dbf_to_excel.main(config)

Convert a DBF file (.dbf) to an Excel file (.xls). The configuration uses the section dbf-tools with the parameters dbf-file (path to the output) and excel-file (path to the input)

Parameters config (cea.config.Configuration) – uses config.dbf_tools.
excel_file and config.dbf_tools.dbf_file

Returns
**cea.interfaces.cli.excel_to_dbf module**

Use the py:mod:`cea.utilities.dbf` module to convert an excel file to a dbf file.

```python
c.cea.interfaces.cli.excel_to_dbf.main(config)
Convert an Excel file (.xls) to a DBF file (.dbf). The configuration uses the section dbf-tools with the
parameters excel-file (path to the input) and dbf-file (path to the output)

Parameters | config (cea.config.Configuration) — uses config.dbf_tools.
| excel_file and config.dbf_tools.dbf_file

Returns
```

**cea.interfaces.cli.excel_to_shapefile module**

Implements the CEA script `excel-to-shapefile`

Similar to how `excel-to-db` takes a dBase database file (example.dbf) and converts that to Excel format, this
does the same with a Shapefile.

It uses the `geopandas.GeoDataFrame` class to read in the shapefile. The geometry column is serialized to a
nested list of coordinates using the JSON notation.

```python
c.cea.interfaces.cli.excel_to_shapefile.excel_to_shapefile(excel_file, shapefile, index, crs, polygon=True)
Expects the Excel file to be in the format created by `cea shapefile-to-excel`

Parameters | polygon (bool) — Set this to False if the Excel file contains polyline data in the
gemetry column instead of the default polygon data. (polylines are used for representing
streets etc.)

cea.INTERFACE.CLI.EXCEL_TO_SHAPEFILE.main(config)
Run `excel_to_shapefile()` with the values from the configuration file, section [shapefile-tools].

Parameters | config (cea.config.Configuration) — the configuration object to use

Returns
```

**cea.interfaces.cli.list_demand_graphs_fields module**

List the fields that can be used for the demand-graphs --analysis-fields parameter given a scenario

```python
c.cea.interfaces.cli.list_demand_graphs_fields.demand_graph_fields(scenario)
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.INTERFACE.CLI.LIST_DEMAND_GRAPHS_FIELDS.main(config)
print the available fields for the demand graphs to STDOUT.

Parameters | config (cea.config.Configuration) —

Returns
```

**cea.interfaces.cli.shapefile_to_excel module**

Implements the CEA script `shapefile-to-excel`
Similar to how dbf-to-excel takes a dBase database file (example.dbf) and converts that to Excel format, this does the same with a Shapefile.

It uses the geopandas.GeoDataFrame class to read in the shapefile. And serializes the geometry column to Excel as well as a serialized list of tuples.

```
cea.interfaces.cli.shapefile_to_excel.main(config)
```

Run `shapefile_to_excel()` with the values from the configuration file, section `shapefile-tools`.

Parameters `config` (cea.config.Configuration)

Returns

```
cea.interfaces.cli.shapefile_to_excel.serialize_geometry(geometry)
```

Take a shapely.geometry.polygon.Polygon and represent it as a string of tuples (x, y)

Parameters `geometry` (shapely.geometry.polygon.Polygon) – a polygon or polyline to extract the points from and represent as a json object

```
cea.interfaces.cli.shapefile_to_excel.shapefile_to_excel(shapefile, excel_file, index=None)
```

Expects shapefile to be the path to an ESRI Shapefile with the geometry column called `geometry`

cea.interfaces.dashboard package

Subpackages

cea.interfaces.dashboard.api package

Submodules

cea.interfaces.dashboard.api.dashboard module

cea.interfaces.dashboard.api.glossary module

cea.interfaces.dashboard.api.inputs module

cea.interfaces.dashboard.api.project module

cea.interfaces.dashboard.api.tools module

cea.interfaces.dashboard.api.utils module

cea.interfaces.dashboard.base package

Submodules

cea.interfaces.dashboard.base.forms module

cea.interfaces.dashboard.base.routes module

```
cea.interfaces.dashboard.base.routes.access_forbidden(error)
```
cea.interfaces.dashboard.base.routes.internal_error(error)
cea.interfaces.dashboard.base.routes.not_found_error(error)
cea.interfaces.dashboard.base.routes.route_default()
cea.interfaces.dashboard.base.routes.route_errors(error)
cea.interfaces.dashboard.base.routes.route_fixed_template(template)
cea.interfaces.dashboard.base.routes.route_glossary_search()
cea.interfaces.dashboard.base.routes.shutdown()

cea.interfaces.dashboard.inputs package

Submodules

cea.interfaces.dashboard.inputs.routes module

cea.interfaces.dashboard.inputs.routes.df_to_json(file_location)
cea.interfaces.dashboard.inputs.routes.dir_last_updated()
cea.interfaces.dashboard.inputs.routes.read_inputs_field_types()
  Parse the inputs.yaml file and create the dictionary of column types
cea.interfaces.dashboard.inputs.routes.route_geojson(db)
  Return a GeoJSON representation of the input file for use in Leaflet.js
cea.interfaces.dashboard.inputs.routes.route_geojson_networks(type)
cea.interfaces.dashboard.inputs.routes.route_geojson_streets()
cea.interfaces.dashboard.inputs.routes.route_get_building_properties()
cea.interfaces.dashboard.inputs.routes.route_save_building_properties()

cea.interfaces.dashboard.landing package

Submodules

cea.interfaces.dashboard.landing.routes module

cea.interfaces.dashboard.plots package

Submodules

cea.interfaces.dashboard.plots.routes module

cea.interfaces.dashboard.server package

Submodules
cea.interfaces.dashboard.server.jobs module

cea.interfaces.dashboard.server.streams module

cea.interfaces.dashboard.tools package

Submodules

cea.interfaces.dashboard.tools.routes module

cea.interfaces.dashboard.tools.worker module

Wrap up the cea.api interface for use in multiprocessing - using multiprocessing.Connection objects to pipe the STD-OUT and STDERR of the scripts.

```
cea.interfaces.dashboard.tools.worker.main(script_name, **kwargs)
```

This is the main interface to start a worker process. The returned multiprocessing.Process object has already been start()'ed. The `Connection` has a recv() method that returns a tuple (name, message), with name being either 'stdout' or 'stdin' and message being the string printed to that stream.

Returns: tuple (Process, Connection)

```
cea.interfaces.dashboard.tools.worker.run_script(script_name, connection, kwargs)
```

```
cea.interfaces.dashboard.tools.worker.test_worker()
```

Run a simple test with `cea test` to see if the worker works

Submodules

cea.interfaces.dashboard.dashboard module

```
cea.interfaces.dashboard.dashboard.get_drives()
```

Get a list of valid drive letters on windows:

```
In [12]: get_drives() Out[12]: ['C:', 'I:', 'K:', 'S:', 'Y:', 'Z:']
```

On on-windows systems, returns None

```
cea.interfaces.dashboard.dashboard.list_tools()
```

List the tools known to the CEA. The result is grouped by category.

```
cea.interfaces.dashboard.dashboard.main(config)
```

cea.interfaces.grasshopper package

Submodules

cea.interfaces.grasshopper.ghhelper module

A library of helper-functions for working with the CEA from grasshopper. Install this with cea install-grasshopper.

The main function used is `run` which runs a CEA script (as defined in the `scripts.yml`)

9.1. cea package
Note: This module is meant to be run from grasshopper, which uses an IronPython interpreter. PyCharm will have a hard time with some of the imports here.

```python
cea.interfaces.grasshopper.ghhelper.get_cea_parameters(config, cea_tool)
    Return a list of cea.config.Parameter objects for each cea_parameter associated with the tool.

cea.interfaces.grasshopper.ghhelper.get_python_exe()
    Return the path to the python interpreter that was used to install CEA

cea.interfaces.grasshopper.ghhelper.run(script, args)
    Run a script, given a config file.
    Parameters
    • `script` (basestring) -- a script name, as defined in the scripts.yml file
    • `args` (dict[str, str]) -- a dictionary consisting of name = value pairs, one per line, for each parameter to override the value should be formatted as it would be in the config file.

cea.interfaces.grasshopper.ghhelper.run_cli(script_name, **parameters)
    Run the CLI in a subprocess without showing windows
```

**cea.interfaces.grasshopper.install_grasshopper module**

Install the grasshopper interface. This assumes that the python path for grasshopper Python scripts is in %APPDATA%\McNeel\Rhinoceros.0\scripts.

```python
cea.interfaces.grasshopper.install_grasshopper.copy_cea_ghuser(user_objects_folder)
cea.interfaces.grasshopper.install_grasshopper.copy_config(scripts_folder)
    Copy the cea/concept_parameters.py, cea/default.config and an empty __init__.py file to the toolbox_folder
cea.interfaces.grasshopper.install_grasshopper.copy_inputlocator(scripts_folder)
    Copy the cea/inputlocator.py file to the toolbox_folder and create the cea/databases.pth file
cea.interfaces.grasshopper.install_grasshopper.copy_library(scripts_folder)
    Copy the library functions
cea.interfaces.grasshopper.install_grasshopper.copy_scripts(scripts_folder)
    Copy the cea/scripts.py, cea/scripts.yml to the toolbox_folder
cea.interfaces.grasshopper.install_grasshopper.get_cea_dst_folder(toolbox_folder)
cea.interfaces.grasshopper.install_grasshopper.main(_)
    Install a subset of the CEA for use inside grasshopper Python scripts.
    Since we don’t know for sure, if `cea install-toolbox` (for the ArcGIS interface) was run, we need to re-create some stuff done there too, like storing the path to the python executable to use.
    Parameters `_` (cea.config.Configuration) -- ignored.
    Returns
```

**cea.optimization package**

**Subpackages**
This class just sets-up constants of the linear model of the distribution. These results are extracted from the work of Florian at the chair. Unfortunately his work only worked for this case study and could not be used elsewhere. See the paper of Fonseca et al 2015 of the city energy analyst for more info on how that procedure used to work.

```python
__init__(district_heating_network, district_cooling_network, locator)
    x.__init__(...) initializes x; see help(type(x)) for signature
```

### Extra costs to an individual

```python
buildings_connected_costs_and_emissions(district_heating_costs, district_cooling_costs, district_microgrid_costs, district_microgrid_requirements_dispatch, district_heating_fuel_requirements_dispatch, district_cooling_fuel_requirements_dispatch, district_electricity_demands, prices, lca)
```
cea.optimization.master.cost_model.buildings_disconnected_costs_and_emissions(column_names_buildings_heating, column_names_buildings_cooling, lca, master_to_slave_vars)

cea.optimization.master.cost_model.calc_costs_emissions_decentralized_DC(DCN_barcode, buildings_names_with_cooling_load)

cea.optimization.master.cost_model.calc_costs_emissions_decentralized_DH(DHN_barcode, buildings_names_with_heating_load)


cea.optimization.master.cost_model.calc_generation_costs_capacity_installed_cooling(locator, master_to_slave_vars, supply_system, mdot_nMax_kgpers)

cea.optimization.master.cost_model.calc_generation_costs_capacity_installed_heating(locator, master_to_slave_vars, config, storage_activation_data, mdot_nMax_kgpers)

Returns returns the objectives addCosts, addCO2, addPrim

Return type tuple

cea.optimization.master.cost_model.calc_generation_costs_cooling_storage (locator, master_to_slave_variables, config, daily_storage)

cea.optimization.master.cost_model.calc_network_costs_cooling (locator, master_to_slave_vars, network_features, network_type, prices)

cea.optimization.master.cost_model.calc_network_costs_heating (locator, master_to_slave_vars, network_features, network_type, prices)

cea.optimization.master.cost_model.calc_seasonal_storage_costs (config, locator, storage_activation_data)

cea.optimization.master.cost_model.calc_substations_costs_cooling (building_names, master_to_slave_vars, district_network_barcode, locator)

cea.optimization.master.cost_model.calc_substations_costs_heating (building_names, district_network_barcode, locator)

cea.optimization.master.cost_model.calc_variable_costs_connected_buildings (sum_natural_gas_imports_W, sum_wet_biomass_imports_W, sum_dry_biomass_imports_W, sum_electricity_imports_W, sum_electricity_exports_W, prices)

cea.optimization.master.cost_model.summary_fuel_electricity_consumption (district_cooling_fuel_requirements_dispatch, district_heating_fuel_requirements_dispatch, district_microgrid_requirements, district_electricity_demands)

cia.optimization.master.crossover module

Crossover routines
cea.optimization.master.crossover.crossover_main\(\text{ind1, ind2, indpb, column_names, heating_unit_names_share, cooling_unit_names_share, column_names_buildings_heating, column_names_buildings_cooling, district_heating_network, district_cooling_network}\)

cea.optimization.master.emissions_model module

cea.optimization.master.emissions_model.calc_emissions_Whyr_to_tonCO2yr\((E_{\text{Wh yr}}, \text{factor}_k\text{g}CO2_\text{to MJ})\)

cea.optimization.master.emissions_model.calc_pen_Whyr_to_MJoilyr\((E_{\text{Wh yr}}, \text{factor}_\text{MJ}_\text{to MJ})\)

cea.optimization.master.evaluation module

Evaluation function of an individual

cea.optimization.master.evaluation.evaluation_main\(\text{individual, building_names_all, locator, network_features, config, prices, lca, ind_num, gen, column_names_individual, column_names_buildings_heating, column_names_buildings_cooling, building_names_heating, building_names_cooling, building_names_electricity, district_heating_network, district_cooling_network}\)

This function evaluates an individual

**Parameters**

- **individual**(list) – list with values of the individual
- **column_names_buildings_all**(list) – list with names of buildings
- **locator**(cea.inputlocator.InputLocator) – locator class
- **solar_features**(class) – solar features call to class
- **network_features**(class) – network features call to class
- **optimization_constants**(class) – class containing constants used in optimization
- **config**(class) – configuration file
- **prices**(class) – class of prices used in optimization

**Returns** Resulting values of the objective function. costs, CO2, prim

**Return type** tuple
cea.optimization.master.evaluation.save_results(master_to_slave_vars, locator, buildings_connected_costs, buildings_connected_emissions, buildings_disconnected_costs, buildings_disconnected_emissions, heating_dispatch, cooling_dispatch, electricity_dispatch, electricity_requirements, performance_totals_dict, building_connectivity_dict, district_heating_capacity_installed_dict, district_cooling_capacity_installed_dict, district_electricity_capacity_installed_dict, buildings_disconnected_heating_capacities, buildings_disconnected_cooling_capacities)

cea.optimization.master.generation module

Create individuals

cea.optimization.master.generation.calc_building_connectivity_dict(building_names_all, building_names_heating, building_names_cooling, DHN_barcode, DCN_barcode)

cea.optimization.master.generation.generate_main(individual_with_names_dict, column_names, column_names_buildings_heating, column_names_buildings_cooling, district_heating_network, district_cooling_network)

Creates an individual configuration for the evolutionary algorithm. The individual is divided into four parts namely Heating technologies, Cooling Technologies, Heating Network and Cooling Network Heating Technologies: This block consists of heating technologies associated with % of the peak capacity each technology is going to supply, i.e. 10.1520.2030, which translates into technology 1 corresponding to 15% of peak capacity, technology 2 corresponding to 20% and technology 3 corresponding to 0%. 0% can also be just done by replacing 3 with 0. The technologies block is then followed by supply temperature of the DHN and the number of units it is supplied to among AHU, ARU, SHU. So if it is 6 degrees C supplied by DHN to AHU and ARU, it is represented as 6.02. The temperature is represented with 1 decimal point. Cooling Technologies: This follows the same syntax as heating technologies, but will be represented with cooling technologies. The block length of heating and cooling can be different. Heating Network: Network of buildings connected to centralized heating. Cooling Network: Network of buildings connected to centralized cooling. Both these networks can be different, and will always have a fixed length corresponding to the total number of buildings in the neighborhood.

:param nBuildings: number of buildings
:type nBuildings: int
:return: individual: representation of values taken by the individual
:rtype: list

9.1. cea package 169
cea.optimization.master.generation.individual_to_barcode(individual, building_names_all, building_names_heating, building_names_cooling, column_names, column_names_buildings_heating, column_names_buildings_cooling)

Reads the 0-1 combination of connected/disconnected buildings and creates a list of strings type barcode i.e. (“1231111123012”) :param individual: list containing the combination of connected/disconnected buildings :type individual: list :return: indCombi: list of strings :rtype: list

type: list

cea.optimization.master.generation.populate_individual(empty_individual_with_names_dict, name_share_conversion_technologies, columns_buildings_name)

ccea.optimization.master.master_main module
c

dict: dictionary

dict: dictionary
dict: dictionary
dict: dictionary
dict: dictionary

c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
c.ie.
Objective function is used to calculate the costs, CO2, primary energy and the variables corresponding to the individual:

- **individual**: Input individual
- **type**: list
- **return**: returns costs, CO2, primary energy and the master_to_slave_vars

**cea.optimization.master.master_main.objective_function()**

Wrap arguments because multiprocessing only accepts one argument for the function

**cea.optimization.master.master_main.save_generation_dataframes()**

```python
generation, selected_individuals, locator, DCN_network_list_selected, DHN_network_list_selected
```

**cea.optimization.master.master_main.save_generation_halloffame_individuals()**

```python
locator, generation, record_individuals_tested, hall_of_fame
```
`cea.optimization.master.master_main.save_generation_individuals` *(columns_of_saved_files, generation, invalid_ind, locator)*

`cea.optimization.master.master_main.save_generation_pareto_individuals` *(locator, generation, record_individuals_tested, paretofrontier)*

**cea.optimization.master.master_to_slave module**

Evaluation function of an individual

`cea.optimization.master.master_to_slave.calc_available_area_solar` *(locator, buildings, share_allowed, technology)*

`cea.optimization.master.master_to_slave.calc_available_area_solar_collectors` *(locator, buildings, share_allowed, technology)*

`cea.optimization.master.master_to_slave.calc_connected_names` *(building_names, barcode)*
cea.optimization.master.master_to_slave.calc_master_to_slave_variables(locator, gen, ind_num, individual_with_names_dict, building_names, DHN_barcode, DCN_barcode, network_file_name_heating, network_file_name_cooling, Q_heating_nom_W, Q_cooling_nom_W, Q_wasteheat_datacentre_nom_W, district_heating_network, district_cooling_network, building_names_heating, building_names_cooling, building_names_electricity)

This function reads the list encoding a configuration and implements the corresponding for the slave routine’s to use :param individual_with_names_dict: list with individual :param Q_heating_max_W: peak heating demand :param locator: locator class :type individual_with_names_dict: list :type Q_heating_max_W: float :type locator: string :return: master_to_slave_vars :class MasterSlaveVariables :rtype: class

cea.optimization.master.master_to_slave.createTotalNtwCsv(barcode, locator, building_names)

Create and saves the total file for a specific DH or DC configuration to make the distribution routine possible :param indCombi: string of 0 and 1: 0 if the building is disconnected, 1 if connected :param locator: path to raw files :type indCombi: string :type locator: string :return: name of the total file :rtype: string
lea.optimization.master.master_to_slave.export_data_to_master_to_slave_class
(locator, gen, ind_num, individual_with_name_dict, building_names, building_names_heating, building_names_cooling, building_names_electric, DHN_barcode, DCN_barcode, district_heating_network, district_cooling_network)

lea.optimization.master.master_to_slave.extract_loads_individual
(locator, individual_with_name_dict, DCN_barcode, DHN_barcode, district_heating_network, district_cooling_network, column_names_buildings_heating, column_names_buildings_cooling)

lea.optimization.master.master_to_slave.master_to_slave_district_cooling_technologies
(Q_cooling_nom_W, individual_with_names_dict, master_to_slave_vars)

lea.optimization.master.master_to_slave.master_to_slave_district_heating_technologies
(Q_heating_nom_W, Q_wasteheat_datacentre_nom_W, individual_with_names_dict, locator, master_to_slave_vars)
cea.optimization.master.master_to_slave.master_to_slave_electrical_technologies

cea.optimization.master.master_to_slave.minimum_valuedc(technology)
cea.optimization.master.master_to_slave.minimum_valuedh(technology)

cea.optimization.master.mutations module

Mutation routines
cea.optimization.master.mutations.mutation_main(individual, indpb, column_names, heating_unit_names_share, cooling_unit_names_share, column_names_buildings_heating, column_names_buildings_cooling, district_heating_network, district_cooling_network)

cea.optimization.master.normalization module

cea.optimization.master.normalization.minmax_scaler(value, min_value, max_value)

cea.optimization.master.normalization.normalize_fitnesses(scaler_dict, fitnesses_population)

cea.optimization.master.normalization.scaler_for_normalization(number_of_objectives, fitnesses)

cea.optimization.master.performance_aggregation module

cea.optimization.master.performance_aggregation.summarize_results_individual(master_to_slave_vars, buildings_connected_costs, buildings_connected_emissions, buildings_disconnected_costs, buildings_disconnected_emissions)

cea.optimization.master.summarize_network module

Hydraulic - thermal network

cea.optimization.master.summarize_network.calc_min_flow(m0, m1)
This function calculates the minimum flow of a distribution by comparison of two vectors. This is useful when looking up at multiple buildings in a for loop. :param m0: last minimum mass flow rate :param m1: current minimum mass flow rate :type m0: float :type m1: float :return: mmin: new minimum mass flow rate :rtype: float
This function estimates the average thermal losses of a distribution for an hour of the year:

\[ \text{param Tnet}_K: \text{current temperature of the pipe} \]
\[ \text{param m_max}_k\text{kgpers: maximum mass flow rate in the pipe} \]
\[ \text{param m_min}_k\text{kgpers: minimum mass flow rate in the pipe} \]
\[ \text{param L: length of the pipe} \]
\[ \text{param Tg: ground temperature} \]
\[ \text{param K: linear transmittance coefficient (it accounts for insulation and pipe diameter)} \]
\[ \text{param cp: specific heat capacity} \]
\[ \text{type Tnet}_K: \text{float} \]
\[ \text{type m_max}_k\text{kgpers: float} \]
\[ \text{type m_min}_k\text{kgpers: float} \]
\[ \text{type L: float} \]
\[ \text{type Tg: float} \]
\[ \text{type K: float} \]
\[ \text{type cp: float} \]
\[ \text{return: Qloss: thermal losses in the pipe.} \]
\[ \text{rtype: float} \]

This function estimates the average thermal losses of a distribution for an hour of the year:

\[ \text{param Tnet}_K: \text{current temperature of the pipe} \]
\[ \text{param m_max}_k\text{kgpers: maximum mass flow rate in the pipe} \]
\[ \text{param m_min}_k\text{kgpers: minimum mass flow rate in the pipe} \]
\[ \text{param L: length of the pipe} \]
\[ \text{param Tg: ground temperature} \]
\[ \text{param K: linear transmittance coefficient (it accounts for insulation and pipe diameter)} \]
\[ \text{param cp: specific heat capacity} \]
\[ \text{type Tnet}_K: \text{float} \]
\[ \text{type m_max}_k\text{kgpers: float} \]
\[ \text{type m_min}_k\text{kgpers: float} \]
\[ \text{type L: float} \]
\[ \text{type Tg: float} \]
\[ \text{type K: float} \]
\[ \text{type cp: float} \]
\[ \text{return: Qloss: thermal losses in the pipe.} \]
\[ \text{rtype: float} \]

This function calculates the return temperature of the distribution for a time step. It is a weighted average of all the return temperatures (from the substations) in the network. This is an approximation of the return temperature of the network to the centralized plant:

\[ \text{param sum}_t\text{m: sum of temperature times mass flow rate} \]
\[ \text{param sum}_m: \text{sum of mass flow rate} \]
\[ \text{type sum}_t\text{m: float} \]
\[ \text{type sum}_m: \text{float} \]
\[ \text{return: tr: vector return temperature} \]
\[ \text{rtype: float} \]

This function calculates the supply temperature of the distribution for a time step:

\[ \text{param tr: current return temperature} \]
\[ \text{param Q: load including thermal losses} \]
\[ \text{param m: mass flow rate} \]
\[ \text{param cp: specific heat capacity} \]
\[ \text{param case: ‘DH’ or something else??} \]
\[ \text{type tr: float} \]
\[ \text{type Q: float} \]
\[ \text{type m: float} \]
\[ \text{type cp: float} \]
\[ \text{type case: string} \]
\[ \text{return: ts: new temperature of the distribution accounting for thermal losses in the grid} \]
\[ \text{rtype: float} \]

This function calculates the new temperature of the distribution including losses:

\[ \text{param t0}_K: \text{current distribution temperature} \]
\[ \text{param Q}_W: \text{thermal losses in the corresponding network (either supply or return)} \]
\[ \text{param m_kg pers: mass flow rate} \]
\[ \text{param cp: specific heat capacity} \]
\[ \text{param case: “positive”: if there is an addition to the losses, “negative” otherwise} \]
\[ \text{type t0}_K: \text{float} \]
\[ \text{type Q}_W: \text{float} \]
\[ \text{type m_kg pers: float} \]
\[ \text{type cp: float} \]
\[ \text{type case: string} \]
\[ \text{return: t1: new temperature of the distribution accounting for thermal losses in the grid} \]
\[ \text{rtype: float} \]

This function returns the index of an array on which the maximum value is at:

\[ \text{param array: ndarray, Array of observations} \]
\[ \text{Each row represents a day and each column the hourly data of that day} \]
\[ \text{type array: list} \]
\[ \text{return: max_index_hour: integer, max_index_hour: tells on what hour it happens (hour of the year) to use: e.g. data_array[max_index_hour] will give the maximum data of the year} \]
\[ \text{Return type list} \]

This function summarizes the distribution demands and will give them as:

\[ \text{- absolute values (design values = extreme values)} \]
\[ \text{- hourly operation scheme of input/output of distribution} \]
Parameters

- **locator (class)** – locator class
- **total_demand (list)** – dataframe with total demand of buildings
- **buildings_in_this_network (vector)** – vector with names of buildings
- **key (int)** – when called by the optimization, a key will provide an id for the individual and the generation.

Returns csv file stored in locator.get_optimization_network_results_folder() as fName_result where fName_result: FIXME: what?

Return type Nonetype

ccea.optimization.master.validation module

Validation

ccea.optimization.master.validation.validation_main(individual_with_name_dict, column_names_buildings_heating, column_names_buildings_cooling, district_heating_network, district_cooling_network)

ccea.optimization.preprocessing package

Submodules

ccea.optimization.preprocessing.decentralized_building_main module

Disconnected buildings

This computes the close-to-optimal supply system for single buildings.

ccea.optimization.preprocessing.decentralized_building_main.disconnected_building_main(locator, total_demand, config, prices, lca)

This functions optimizes disconnected buildings individually

Parameters **locator (class)** – locator class

Returns elecCosts, elecCO2, elecPrim

Return type tuple

ccea.optimization.preprocessing.decentralized_building_main.main(config)

ccea.optimization.preprocessing.decentralized_buildings_cooling module

Operation for decentralized buildings
This function calculates the load distribution side of the district heating distribution. 

:param mdot_kgpers: mass flow
:param T_sup_K: chilled water supply temperature
:param T_re_K: chilled water return temperature
:type mdot_kgpers: float
:type T_sup_DH: float
:type T_re_K: float
:return: Q_cooling_load: load of the distribution
:type: float
Computes the parameters for the operation of disconnected buildings output results in csv files. There is no optimization at this point. The different cooling energy supply system configurations are calculated and compared 1 to 1 to each other. It is a classical combinatorial problem. The six supply system configurations include:

- (VCC: Vapor Compression Chiller, ACH: Absorption Chiller, CT: Cooling Tower, Boiler) (AHU: Air Handling Units, ARU: Air Recirculation Units, SCU: Sensible Cooling Units) - config 0: Direct Expansion / Mini-split units (NOTE: this configuration is not fully built yet)
- config 1: VCC_to_AAS (AHU + ARU + SCU) + CT
- config 2: FP + single-effect ACH_to_AAS (AHU + ARU + SCU) + Boiler + CT
- config 3: ET + single-effect ACH_to_AAS (AHU + ARU + SCU) + Boiler + CT
- config 4: VCC_to_AA (AHU + ARU) + VCC_to_S (SCU) + CT
- config 5: VCC_to_AA (AHU + ARU) + single effect ACH_S (SCU) + CT + Boiler

Note: 1. Only cooling supply configurations are compared here. The demand for electricity is supplied from the grid, and the demand for domestic hot water is supplied from electric boilers. 2. Single-effect chillers are coupled with flat-plate solar collectors, and the double-effect chillers are coupled with evacuated tube solar collectors.

```python
ccea.optimization.preprocessing.decentralized_buildings_cooling.get_SC_data(building_name, locaton, panel_type)
ccea.optimization.preprocessing.decentralized_buildings_cooling.get_tech_unit_size_and_number(Qc_nom_W, max_tech_size_W)
ccea.optimization.preprocessing.decentralized_buildings_cooling.initialize_result_tables_for_supply_configurations(Qc_nom_SCU_W)
```

The cooling technologies are listed as follow:

```python
ccea.optimization.preprocessing.decentralized_buildings_cooling.main(config)
```

run the whole preprocessing routine

```python
ccea.optimization.preprocessing.decentralized_buildings_cooling.rank_results(TAC_USD, TotalCO2, TotalPrim, number_of_configurations)
```

**cea.optimization.preprocessing.decentralized_buildings_heating module**

Operation for decentralized buildings
cea.optimization.preprocessing.decentralized_buildings_heating.calc_GHP_operation\( (Q_{\text{nom GHP}}, T_{\text{ground}}, T_{\text{exit GHP}}, T_{\text{ret}}, T_{\text{sup}}, \dot{m}_{\text{kgpers}}, q_{\text{load Wh}}) \)

This function calculates the load distribution side of the district heating distribution. 

:param \( \dot{m}_{\text{kgpers}} \): mass flow 
:param \( T_{\text{supDH}} \): supply temperature 
:param \( T_{\text{ret}} \): return temperature 
:type \( \dot{m}_{\text{kgpers}} \): float 
:type \( T_{\text{supDH}} \): float 
:type \( T_{\text{ret}} \): float 
:return: \( Q_{\text{load W}} \): load of the distribution 
:type: float

cke.optimization.preprocessing.decentralized_buildings_heating.calc_new_load\( (\dot{m}_{\text{kgpers}}, T_{\text{supDH}}, T_{\text{ret}}) \)

cke.optimization.preprocessing.decentralized_buildings_heating.disconnected_buildings_heating_main\( (\text{locator}, \text{total demand}, \text{building names}, \text{config}, \text{prices}, \text{lca}, \text{loca}\text{tor, prices}) \)

Computes the parameters for the operation of disconnected buildings output results in csv files. There is no optimization at this point. The different technologies are calculated and compared 1 to 1 to each technology. It is a classical combinatorial problem. 

:param \( \text{locator} \): locator class 
:param \( \text{building names} \): list with names of buildings 
:type \( \text{locator} \): class 
:type \( \text{building names} \): list 
:return: results of operation of buildings located in \( \text{locator.get_optimization_decentralized_folder} \) 
:type: Nonetype

cke.optimization.preprocessing.decentralized_buildings_heating.disconnected_heating_for_building\( (\text{building name}, \text{supply systems}, T_{\text{ground}}, \text{geothermal potential data}, \text{lca}, \text{loca}\text{tor, prices}) \)

cke.optimization.preprocessing.preprocessing_main module

Pre-processing algorithm

cke.optimization.preprocessing.preprocessing_main.get_building_names_with_load\( (\text{total demand, load name}) \)
This function aims at preprocessing all data for the optimization.

**Parameters**

- `locator (class)` – path to locator function
- `total_demand (list)` – dataframe with total demand and names of all building in the area
- `building_names (list)` – dataframe with names of all buildings in the area
- `weather_file (string)` – path to weather file

**Returns**

- `extraCosts`: extra pareto optimal costs due to electricity and process heat (these are treated separately and not considered inside the optimization)
- `extraCO2`: extra pareto optimal emissions due to electricity and process heat (these are treated separately and not considered inside the optimization)
- `extraPrim`: extra pareto optimal primary energy due to electricity and process heat (these are treated separately and not considered inside the optimization)
- `solar_features`: extraction of solar features form the results of the solar technologies calculation.

**Return type** float, float, float, float

### `cea.optimization.preprocessing.processheat module`

**Boiler Pre-treatment for Heat Processing**

At the moment, process heat is excluded form the optimization process. It is considered that whenever the case, the most competitive alternative is to have a dedicated natural gas boiler

This function calculates the contribution to the pareto optimal results of process heating.

**Parameters**

- `locator (class)` – locator class
- `total_demand (class)` – dataframe with building demand

**Returns** hpCosts, hpCO2, hpPrim

**Return type** tuple
cea.optimization.slave package

Subpackages

cea.optimization.slave.daily_storage package

Submodules

cea.optimization.slave.daily_storage.load_leveling module

class cea.optimization.slave.daily_storage.load_leveling.LoadLevelingDailyStorage(storage_on, Qc_tank_charging_limit_W, T_tank_fully_charged_K, T_tank_fully_discharged_K, T_tank_K, T_ground_average_K)

Bases: object

def __init__(storage_on, Qc_tank_charging_limit_W, T_tank_fully_charged_K, T_tank_fully_discharged_K, T_tank_K, T_ground_average_K)
x.__init__(...) initializes x: see help(type(x)) for signature

def charge_storage(Q_request_W)

def discharge_storage(Q_request_W)

def storage_temperature(Q_storage_possible_W, activation)

cea.optimization.slave.seasonal_storage package

Submodules

cea.optimization.slave.seasonal_storage.Import_Network_Data_functions module

Import Network Data:

This File reads all relevant thermal data for further analysis in the Slave Routine, Namely: Thermal (J+) and Solar Data (J+)

ccea.optimization.slave.seasonal_storage.Import_Network_Data_functions.import_solar_thermal_data(fName)

importing and preparing raw data for analysis of the district distribution

Parameters fName – name of file where solar data is stored in

Returns Arrays containing all relevant data for further processing: 
mdot_sst_heat, mdot_sst_cool, T_sst_heat_return, T_sst_heat_supply, T_sst_cool_return, Q_DH_building, Q_DC_building, Q_DH_building_max, Q_DC_building_max, T_sst_heat_supply_ofmaxQh, T_sst_heat_return_ofmaxQh, T_sst_cool_return_ofmaxQc

Return type list

cea.optimization.slave.seasonal_storage.SolarPowerHandler_incl_Losses module

Slave Sub Function - Treat solar power! In this file, all sub-functions are stored that are used for storage design and operation. They are called by either the operation or optimization of storage.
This function is a first filter for solar energy handling: If there is excess solar power, this will be specified and stored. If there is not enough solar power, the lack will be calculated.

Parameters

- `Q_solar_available_Wh (float)` – solar energy available at a given time step
- `Q_network_demand_W (float)` – network load at a given time step
- `P_HP_max_W (float)` – storage??

Return: `Q_to_storage` Thermal Energy going to the Storage Tanks (excl. conversion losses)

Return type: `float, float, int`

Calculates the temperature of storage when charging `Q_to_storage_new_W = including losses`

Parameters

- `T_storage_old_K (float)` –
- `Q_to_storage_lossfree_W (float)` –
- `T_DH_ret_K (float)` –
- `Q_in_storage_old_W (float)` –
- `STORAGE_SIZE_m3 (float)` –
- `context (string)` –

Returns: `T_storage_new, Q_to_storage_new_W, E_aux, Q_in_storage_new ??`

Return type: `float, float, float, float ??`

Discharging of the storage, no outside thermal losses in the model

Parameters
Calculates the storage Loss for every time step, assume D : H = 3 : 1

Parameters

- \( T_{\text{storage\_old\_K}} \) (float) – temperature of storage at time step, without any losses
- \( T_{\text{amb\_K}} \) (float) – ambient temperature
- \( \text{STORAGE\_SIZE\_m3} \) (float)
- \( \text{context} \)

Returns

Energy loss due to non perfect insulation in Wh/h

Return type float
• T_DH_sup_K -
• T_amb_K -
• Q_in_storage_old_W -
• T_DH_return_K -
• mdot_DH_kgps -
• STORAGE_SIZE_m3 -
• context -
• P_HP_max_W -

Returns

Return type


USE ONLY IF Q solar is not sufficient! This function derives the temperature just before the power plant, after
solar energy is injected.

Parameters

• `Q_network_demand (float)` – network load at a given time step
• `Q_solar_available (float)` – solar energy available at a given time step
• `mdot_DH (float)` – ??
• `T_return_DH (float)` – ??

Returns temperature before powerplant

Return type `float`

`cea.optimization.slave.seasonal_storage.design_operation module`

Storage Design And Operation This File is called “Storage_Optimizer_incl_Losses_main.py” (Optimization
Routine) and will operate the storage according to the inputs given by the main file.

The operation data is stored

`cea.optimization.slave.seasonal_storage.design_operation.Storage_Design(CSV_NAME,
T_storage_old_K, Q_in_storage_old_W, locator, STORAGE_SIZE_m3, solar_technologies_data, master_to_slave_vars, P_HP_max_W, config)`
Parameters

- CSV_NAME
- SOLCOL_TYPE
- T_storage_old_K
- Q_in_storage_old_W
- locator
- STORAGE_SIZE_m3
- STORE_DATA
- master_to_slave_vars
- P_HP_max_W

Returns

Return type

`cea.optimization.slave.seasonal_storage.design_operation.get_heating_provided_by_onsite_energy_sources(Q_PVT_gen_W, Q_SC_ET_gen_W, Q_SC_FP_gen_W, Q_Server_gen_initial_W, Solar_Tscr_th_PVT_K, Solar_Tscr_th_SC_ET_K, Solar_Tscr_th_SC_FP_K,T_DH_sup_K, master_to_slave_vars)`

`cea.optimization.slave.seasonal_storage.design_operation.read_data_from_Network_summary(CSV_NAME, locator)`

**cea.optimization.slave.seasonal_storage.storage_main module**

storage sizing

This script sizes the storage and in a second part, it will plot the results of iteration. Finally, the storage operation is performed with the parameters found in the storage optimization. All results are saved in the folder of “locator.get_optimization_slave_results_folder()”. - Data_with_Storage_applied.csv : Hourly Operation of Storage, especially Q_missing and E_aux is important for further usage - Storage_Sizing_Parameters.csv : Saves the parameters found in the storage optimization. IMPORTANT : Storage is used for solar thermal energy ONLY! It is possible to turn off the plots by setting Tempplot = 0 and Qplot = 0

`cea.optimization.slave.seasonal_storage.storage_main.calc_T_initial_from_Q_and_V(Q_initial_W, T_ST_MIN, V_storage_initial_m3)`

`cea.optimization.slave.seasonal_storage.storage_main.calc_available_generation_PVT(locator, buildings, share_allowed)`
cea.optimization.slave.seasonal_storage.storage_main.calc_available_generation_solar(locator, buildings, share_allowed, type)

cea.optimization.slave.seasonal_storage.storage_main.calc_storage_volume_from_heat_requirement(Q_required_in_storage_W, T_ST_MAX, T_ST_MIN)

cea.optimization.slave.seasonal_storage.storage_main.calc_temperature_convergence(Q_storage_content_final_W)

cea.optimization.slave.seasonal_storage.storage_main.read_solar_technologies_data(locator, master_to_slave_vars)

cea.optimization.slave.seasonal_storage.storage_main.storage_optimization(locator, master_to_slave_vars, config)

This function performs the storage optimization and stores the results in the designated folders:
:param locator: locator class
:param master_to_slave_vars: class MastertoSlaveVars containing the value of variables to be passed to the slave optimization for each individual
:type locator: class
:type master_to_slave_vars: class
:return: The function saves all files when it’s done in the location locator.get_potentials_solar_folder():
:rtype: Nonetype

Submodules

**cea.optimization.slave.cooling_main module**

District Cooling Network Calculations.

Use free cooling from Lake as long as possible (HP Lake operation from slave) If Lake exhausted, then use other supply technologies

cea.optimization.slave.cooling_main.calc_network_summary_DCN(locator, master_to_slave_vars)

cea.optimization.slave.cooling_main.district_cooling_network(locator, master_to_slave_variables, config, prices, network_features)

Computes the parameters for the cooling of the complete DCN

**Parameters**

- **locator**(cea.inputlocator.InputLocator) – path to res folder
- **network_features**(class) – network features
- **prices**(class) – Prices imported from the database

**Returns** costs, co2, prim

**Return type** tuple
**cea.optimization.slave.cooling_resource_activation module**

**calc_chiller_absorption_operation**

```
(cea.optimization.slave.cooling_resource_activation.calc_chiller_absorption_operation(Qc_ACH_req_W,
T_DCN_re_K, T_DCN_sup_K, T_ACH_in_C, T_ground_K, chiller_prop, size_ACH_W))
```

**calc_vcc_CT_operation**

```
(cea.optimization.slave.cooling_resource_activation.calc_vcc_CT_operation(Qc_from_VCC_W, T_DCN_re_K, T_DCN_sup_K, T_source_K, size_chiller_CT))
```

**calc_vcc_operation**

```
(cea.optimization.slave.cooling_resource_activation.calc_vcc_operation(Qc_from_VCC_W, T_DCN_re_K, T_DCN_sup_K, T_source_K))
```

**cooling_resource_activator**

```
(cea.optimization.slave.cooling_resource_activation.cooling_resource_activator(Q_thermal_req, T_district_cooling_supply_K, T_district_cooling_return_K, Q_therm_Lake_W, T_source_average_Lake_K, daily_storage_class, T_ground_K, master_to_slave_variables, absorption_chiller, CCGT_operation_data))
```

**Parameters**

- `Q_thermal_req`
- `T_district_cooling_supply_K`
- `T_district_cooling_return_K`
- `Q_therm_Lake_W`
- `T_source_average_Lake_K`
- `daily_storage_class`
- `T_ground_K`
- `master_to_slave_variables`
- `absorption_chiller`

**Returns**
**cea.optimization.slave.electricity_main module**

Electricity imports and exports script

This file takes in the values of the electricity activation pattern (which is only considering buildings present in network and corresponding district energy systems) and adds in the electricity requirement of decentralized buildings and recalculates the imports from grid and exports to the grid

```python
cea.optimization.slave.electricity_main.calc_available_generation_PV(locator, buildings, share_allowed)
```

```python
cea.optimization.slave.electricity_main.calc_district_system_electricity_generated(master_to_slave_vars)
```

```python
cea.optimization.slave.electricity_main.calc_district_system_electricity_requirements(master_to_slave_vars, building_names, locator, DH_el, DC_el)
```

```python
cea.optimization.slave.electricity_main.calc_electricity_performance_costs(locator, E_GRID_directload_W, master_to_slave_vars)
```

```python
cea.optimization.slave.electricity_main.calc_electricity_performance_emissions(lca, E_PV_gen_export_W, E_GRID_directload_W)
```

```python
cea.optimization.slave.electricity_main.electricity_activation_curve(E_CHP_gen_W, E_PVT_gen_W, E_Furnace_dry_gen_W, E_Furnace_wet_gen_W, E_Trigen_NG_gen_W, E_PV_gen_W, E_req_hour_W)
```

```python
cea.optimization.slave.electricity_main.electricity_calculations_of_all_buildings(locator, master_to_slave, district_heating, district_cooling)
```
cee.optimization.slave.electricity_main.extract_electricity_demand_buildings(master_to_slave_vars, building_names, locator)

cee.optimization.slave.electricity_main.extract_fuels_demand_buildings(master_to_slave_vars, building_names, locator)

cee.optimization.slave.heating_main.calc_network_summary_DHN(locator, master_to_slave_vars)

cee.optimization.slave.heating_main.district_heating_network(locator, master_to_slave_variables, config, prices, lca, network_features)

Computes the parameters for the heating of the complete DHN

Parameters

- locator (class) – locator class
- master_to_slave_variables (class) – class MastertoSlaveVars containing the value of variables to be passed to the slave optimization for each individual
- solar_features (class) – solar features class

Returns

- E_oil_eq_MJ: MJ oil Equivalent used during operation
- CO2_kg_eq: kg of CO2-Equivalent emitted during operation
- cost_sum: total cost in CHF used for operation
- Q_source_data[:,7]: uncovered demand

Return type float, float, float, array
cea.optimization.slave.heating_resource_activation module

cea.optimization.slave.heating_resource_activation.heating_source_activator(Q_therm_req_W, master_to_slave_vars, Q_therm_GHP_W, Tret-GHP-Array_K, Tret-LakeArray_K, Q_therm_Lake_W, Q_therm_Sew_W, Tret-sewAr-ray_K, tdhsup_K, tdhret req_K)

Parameters

- Q_therm_req_W (float) -
- hour (int) -
- context (list) -

Returns cost_data_centralPlant_op, source_info, Q_source_data, E_coldsource_data, E_PP_el_data, E_gas_data, E_wood_data, Q_excess

Return type

cea.optimization.slave.natural_gas_main module

Natural Gas Imports Script

This script calculates the imports of natural gas for a neighborhood to provide heating/cooling. It has two loops: one for each of heating network and cooling network This is then combined to calculate the total natural gas imports and the corresponding file is saved in the respective folder

cea.optimization.slave.natural_gas_main.fuel_imports(master_to_slave_vars, heating_dispatch, cooling_dispatch)

cea.optimization.slave.natural_gas_main.main(config)

cea.optimization.slave.test module

cea.optimization.slave.test.main()

Submodules

9.1. cea package
**cea.optimization.constants module**

This file contains the constants used in objective function calculation in optimization.

**cea.optimization.lca_calculations module**

This file imports the price details from the cost database as a class. This helps in preventing multiple importing of the corresponding values in individual files.

```python
class cea.optimization.lca_calculations.LcaCalculations(locator):
    Bases: object

    __init__(locator)
    x.__init__(...) initializes x; see help(type(x)) for signature
```

**cea.optimization.optimization_main module**

multi-objective optimization of supply systems for the CEA

```python
cea.optimization.optimization_main.check_input_files(config, locator)
    Raise a ValueError if any of the required input files are missing. :param cea.config.Configuration config:
    The config object to use :param cea.config.InputLocator locator: The input locator to use :return: None

demand_files_exist(locator)
    verify that the necessary demand files exist

main(config)
    run the whole optimization routine

moo_optimization(locator, weather_file, config)
    This function optimizes the conversion, storage and distribution systems of a heating distribution for the case study. It requires that the energy demand, technology potential and thermal networks are simulated, as follows:
    - energy demand simulation: run cea/demand/demand_main.py
    - PV potential: run cea/technologies/solar/photovoltaic.py
    - PVT potential: run cea/technologies/solar/photovoltaic_thermal.py
    - flat plate solar collector potential: run cea/technologies/solar/solar_collector.py with config.solar.type_scpanel = ‘FP’
    - evacuated tube solar collector potential: run cea/technologies/solar/solar_collector.py with config.solar.type_scpanel = ‘ET’
    - waste water heat recovery: run cea/resources/sewage_heat_exchanger.py
    - lake water potential: run cea/resources/water_body_potential.py
    - thermal network simulation: run cea/technologies/thermal_network/thermal_network.py if no network is currently present in the case study, consider running network_layout/main.py first
    - decentralized building simulation: run cea/optimization/preprocessing/decentralized_building_main.py

    Parameters
    - locator(cea.inputlocator.InputLocator) – path to input locator
    - weather_file(string) – path to weather file

    Returns None
```
Return type  Nonetype

cea.optimization.prices module

This file imports the price details from the cost database as a class. This helps in preventing multiple importing of the corresponding values in individual files.

class cea.optimization.prices.Prices(supply_systems, detailed_electricity_pricing)
    Bases: object
    __init__(supply_systems, detailed_electricity_pricing)
    x.__init__(...) initializes x; see help(type(x)) for signature

cea.optimization.slave_data module

Data required for Slave from Master This File sets all variables for the slave optimization, that have to be set by the Master

class cea.optimization.slave_data.SlaveData
    Bases: object
    __init__()
    x.__init__(...) initializes x; see help(type(x)) for signature

cea.plots package

class cea.plots.Dashboard(config, dashboard_dict, cache)
    Bases: object
    Implements a dashboard - an editable collection of configured plots.
    __init__(config, dashboard_dict, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature
    add_plot(category, plot_id, index=None)
    Add a new plot to the specified index in the dashboard
    remove_plot(plot_index)
    Remove a plot by index
    set_scenario(scenario)
    Set all scenario parameters of dashboard plots
    to_dict()
    Return a dict representation for storing in yaml

cea.plots.dashboard_yml_path(config)
    The path to the dashboard_yml file

cea.plots.default_dashboard(config, cache)
    Return a default Dashboard

cea.plots.delete_dashboard(config, dashboard_index)
    Remove the dashboard with that index from the dashboard configuration file

cea.plots.duplicate_dashboard(config, cache, name, dashboard_index)

cea.plots.load_plot(project, plot_definition, cache)
    Load a plot based on a plot definition dictionary as used in the dashboard_yml file

9.1. cea package  193
cea.plots.main(config)
    Test the dashboard functionality. Run it twice, because the dashboard.yml might have been created as a result.

cea.plots.new_dashboard(config, cache, name, layout, grid_width=None)
    Append a new dashboard to the dashboard configuration and write it back to disk. Returns the index of the new
dashboard in the dashboards list.

cea.plots.read_dashboards(config, cache)
    Return a list of dashboard configurations for a given project. The dashboard is loaded from the dashboard.yml
file located in the project_path (parent folder of the scenario). If no such file is found, a default one is returned
(but not written to disk).

cea.plots.write_dashboards(config, dashboards)
    Write a list of Dashboard objects to disk

Subpackages

cea.plots.comparisons package

class cea.plots.comparisons.ComparisonsPlotBase(config, parameters, cache)
    Bases: cea.plots.base.PlotBase

    Implements properties / methods used by all plots in this category

    __init__(config, parameters, cache)

    Parameters

        • project – The project to base plots on (some plots span scenarios)
        • parameters – The plot parameters as, e.g., per the dashboard.yml file
        • cache(cea.plotsPlotCache) – a PlotCache instance for speeding up plotting

calc_input_variables()

category_name = 'comparisons'

expected_parameters = {'urban-energy-system-scenarios': 'plots-scenario-comparisons:urban-energy-system-scenarios'}

locator

preprocessing_annual_costs_scenarios()
**cea.plots.comparisons.old.emissions_intensity module**

`cea.plots.comparisons.old.emissions_intensity.calc_graph(analysis_fields, data_frame)`

`cea.plots.comparisons.old.emissions_intensity.calc_table(analysis_fields, data_frame)`

`cea.plots.comparisons.old.emissions_intensity.emissions_intensity(data_frame, analysis_fields, title, output_path)`

**cea.plots.comparisons.old.energy_demand module**

`cea.plots.comparisons.old.energy_demand.calc_graph(analysis_fields, data_frame)`

`cea.plots.comparisons.old.energy_demand.calc_table(analysis_fields, data_frame)`

`cea.plots.comparisons.old.energy_demand.energy_demand_district(data_frame, analysis_fields, title, output_path)`

**cea.plots.comparisons.old.energy_supply_mix module**

`cea.plots.comparisons.old.energy_supply_mix.calc_graph(analysis_fields, data_frame)`

`cea.plots.comparisons.old.energy_supply_mix.energy_supply_mix(data_frame, analysis_fields, title, yaxis_title, output_path)`

**cea.plots.comparisons.old.energy_use_intensity module**

`cea.plots.comparisons.old.energy_use_intensity.calc_graph(analysis_fields, data_frame)`

`cea.plots.comparisons.old.energy_use_intensity.calc_table(analysis_fields, data_frame)`

`cea.plots.comparisons.old.energy_use_intensity.energy_use_intensity(data_frame, analysis_fields, title, output_path)`

**cea.plots.comparisons.old.main module**

This is the dashboard of CEA
class cea.plots.comparisons.old.main.Plots:
    __init__(project, urban_scenarios, energy_system_scenarios_generation, energy_system_scenarios_individual, config)

    comparison_demand
    comparison_demand_intensity
    comparison_emissions
    comparison_emissions_intensity
    comparison_primary_energy
    comparison_primary_energy_intensity
    comparison_supply_mix
    comparison_supply_mix_intensity
    erase_zeros
    occupancy_types_comparison
    preprocessing_costs_scenarios
    preprocessing_demand_scenarios
    preprocessing_lca_scenarios
    preprocessing_occupancy_type_comparison
    preprocessing_supply_scenarios

cea.plots.comparisons.old.main.main(config)

cea.plots.comparisons.old.main.plots_main(config)

cea.plots.comparisons.old.occupancy_types.calc_graph
cea.plots.comparisons.old.occupancy_types.calc_table

cea.plots.comparisons.old.operation_costs.calc_graph
cea.plots.comparisons.old.operation_costs.calc_table
cea.plots.comparisons.old.operation_costs.operation_costs_district(data_frame, analysis_fields, title, yaxis_title, output_path)

cea.plots.comparisons.old.primary_energy module

cea.plots.comparisons.old.primary_energy.calc_graph(analysis_fields, data_frame)
cea.plots.comparisons.old.primary_energy.calc_table(analysis_fields, data_frame)
cea.plots.comparisons.old.primary_energy.primary_energy(data_frame, analysis_fields, title, output_path)

cea.plots.comparisons.old.primary_energy_intensity module

cea.plots.comparisons.old.primary_energy_intensity.calc_graph(analysis_fields, data_frame)
cea.plots.comparisons.old.primary_energy_intensity.calc_table(analysis_fields, data_frame)
cea.plots.comparisons.old.primary_energy_intensity.primary_energy_intensity(data_frame, analysis_fields, title, output_path)

Submodules

cea.plots.comparisons.Annual_costs module

class cea.plots.comparisons.Annual_costs.ComparisonsAnnualCostsPlot(project, parameters, cache)

Bases: cea.plots.comparisons.ComparisonsPlotBase

Implement the “CAPEX vs. OPEX of centralized system in generation X” plot

__init__(project, parameters, cache)

Parameters

• project – The project to base plots on (some plots span scenarios)
• parameters – The plot parameters as, e.g., per the dashboard.yml file
• cache(cea.plots.PlotCache) – a PlotCache instance for speeding up plotting
```
calc_graph()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

layout
name = 'Comparisons annual costs'
output_path
title
cea.plots.comparisons.Annual_costs.main()
    Test this plot

cea.plots.demand package

class cea.plots.demand.DemandPlotBase (project, parameters, cache)
Bases: cea.plots.base.PlotBase

    Implements properties / methods used by all plots in this category
__init__ (project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature
_calculate_hourly_loads()

add_fields (df1, df2)
    Add the demand analysis fields together - use this in reduce to sum up the summable parts of the dfs

add_timeframe (data_demand)
calculate_external_temperature()
calculate_hourly_loads ()
category_name = 'demand'
data
expected_parameters = {'buildings': 'plots:buildings', 'scenario-name': 'general:scenario-name'}

hourly_loads
    Returns the hourly loads, summed up for all the builidings being considered by the plot. Uses the PlotCache
to speed up self._calculate_hourly_loads()

yearly_loads

class cea.plots.demand.DemandSingleBuildingPlotBase (project, parameters, cache)
Bases: cea.plots.demand.DemandPlotBase

    A base class for demand plots that only work on a single building
__init__ (project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature

expected_parameters = {'building': 'plots:building', 'scenario-name': 'general:scenario-name'}

Submodules

```
cea.plots.demand.comfort_chart module

class cea.plots.demand.comfort_chart.ComfortChartPlot(project, parameters, cache)
Bases: cea.plots.demand.DemandSingleBuildingPlotBase

__init__(project, parameters, cache)
x.__init__(…) initializes x; see help(type(x)) for signature

calc_graph()  
Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_table()  
draws table of monthly energy balance

data
dict_graph

expected_parameters = {'building': 'plots:building', 'scenario-name': 'general:scenario-name'}

layout

name = 'Comfort Chart'

cea.plots.demand.comfort_chart.calc_constant_rh_curve(t_array, rh, p)
Calculates curves of humidity ratio at different temperatures for a constant relative humidity and pressure

Parameters

• t_array (numpy.array) – array of temperatures [°C]
• rh (double) – relative humidity [-]
• p (double) – atmospheric pressure [Pa]

Returns humidity ratio [g / kg dry air]

Return type numpy.array

cea.plots.demand.comfort_chart.calc_data(data_frame, locator)

split up operative temperature and humidity points into 4 categories for plotting (1) occupied in heating season (2) un-occupied in heating season (3) occupied in cooling season (4) un-occupied in cooling season

param data_frame results from demand calculation
type data_frame pandas.DataFrame
param config cea config
type config cea.config.Configuration
param locator cea input locator
type locator cea.inputlocator.InputLocator

return dict of lists with operative temperatures and moistures

or 4 conditions (summer (un)occupied, winter (un)occupied)

rtype dict

cea.plots.demand.comfort_chart.calc_graph(dict_graph)
creates scatter of comfort and curves of constant relative humidity
Parameters `dict_graph` *(dict)* – contains comfort conditions to plot, output of `comfort_chart.calc_data()`

**Returns** traces of scatter plot of 4 comfort conditions

**Return type** list of plotly.graph_objs.Scatter

`cea.plots.demand.comfort_chart.calc_table(dict_graph)`

draws table of monthly energy balance

Parameters `dict_graph` *(dict)* – dict containing the lists of summer, winter, occupied and unoccupied operative temperatures and moisture ratios, i.e. the results of `comfort_chart.calc_data`

**Returns** plotly table trace

**Return type** plotly.graph_objs.Table

`cea.plots.demand.comfort_chart.check_comfort(temperature, moisture, vertices_comfort_area)`

checks if a point of operative temperature and moisture ratio is inside the polygon of comfort defined by its vertices, the function only works if the polygon has constant moisture ratio edges

**Parameters**

- `temperature` *(list)* – operative temperature [°C]
- `moisture` *(list)* – moisture ratio [g/kg dry air]
- `vertices_comfort_area` *(list of tuples)* – vertices of operative temperature and moisture ratio ([°C],[g/kg dry air])

**Returns** hours of comfort, hours of uncomfort

**Return type** double, double

`cea.plots.demand.comfort_chart.comfort_chart(data_frame, title, output_path, config, locator)`

Main function of comfort chart plot

**Parameters**

- `data_frame` *(pandas.DataFrame)* – results from demand calculation
- `title` *(string)* – title of plot
- `output_path` *(system path)* – path to output folder

**Returns**

`cea.plots.demand.comfort_chart.create_layout(title)`

Creates layout of plot, including polygon comfort areas

**Parameters** `title` *(string)* – title of plot

**Returns** trace_layout, layout

**Return type** plotly.graph_objs.trace, plotly.graph_objs.layout

`cea.plots.demand.comfort_chart.create_relative_humidity_lines()`

calculates curves of constant relative humidity for plotting (10% - 100% in steps of 10%)

**Returns** list of plotly table trace

**Return type** list of plotly.graph_objs.Scatter

`cea.plots.demand.comfort_chart.datetime_in_season(dt, season_start, season_end)`
small function to determine if a datetime index of the results dataframe is in heating season (winter) or cooling season (summer)

Parameters
- **dt** (*datetime.datetime*) – datetime, index of resulting csv of `cea.demand_main`
- **season_start** (*string*) – start of season [“DD|MM”]
- **season_end** (*string*) – end of season [“DD|MM”]

Returns True or False
Return type bool

`cea.plots.demand.comfort_chart.hum_ratio_from_p_w_and_p(p_w, p)`
Calculate humidity ratio from water vapor pressure and atmospheric pressure Eq(22) in “CHAPTER 6 - PSYCHROMETRICS” in “2001 ASHRAE Fundamentals Handbook (SI)”

Parameters
- **p_w** (*double*) – water vapor pressure [Pa]
- **p** (*double*) – atmospheric pressure [Pa]

Returns humidity ratio [g / kg dry air]
Return type double

`cea.plots.demand.comfort_chart.p_w_from_rh_p_and_ws(rh, p_ws)`
Calculate water vapor pressure from relative humidity and water vapor saturation pressure Eq(6) in “CHAPTER 6 - PSYCHROMETRICS” in “2001 ASHRAE Fundamentals Handbook (SI)”

Parameters
- **rh** (*double*) – relative humidity [-]
- **p_ws** (*double*) – water vapor saturation pressure [Pa]

Returns water vapor pressure [Pa]
Return type double

`cea.plots.demand.comfort_chart.p_ws_from_t(t_celsius)`
Calculate water vapor saturation pressure over liquid water for the temperature range of 0 to 200Â°C Eq (6) in “CHAPTER 6 - PSYCHROMETRICS” in “2001 ASHRAE Fundamentals Handbook (SI)”

Parameters **t_celsius** (*double*) – temperature [Â°C]

Returns water vapor saturation pressure [Pa]
Return type double

`cea.plots.demand.energy_balance module`

```python
class cea.plots.demand.energy_balance.EnergyBalancePlot(project, parameters, cache)
    Bases: cea.plots.demand.DemandSingleBuildingPlotBase
    __init__(project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature
```

9.1. cea package
calc_graph()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_table()
    draws table of monthly energy balance
    :param self :return: table_df

data_frame_month
layout
name = 'Energy balance'

cea.plots.demand.energy_balance.calc_graph(analysis_fields, data_frame)
draws building heat balance graph

    Parameters
    • analysis_fields –
    • data_frame –

    Returns

cea.plots.demand.energy_balance.calc_monthly_energy_balance(data_frame, normalize_value)
calculates heat flux balance for buildings on hourly basis

    Parameters
    • data_frame – demand information of building in pd.DataFrame
    • normalize_value – value for normalization of thermal energy fluxes, usually GFA

    Returns

cea.plots.demand.energy_balance.calc_table(data_frame_month)
draws table of monthly energy balance

    Parameters data_frame_month – data frame of monthly building energy balance

    Returns

cea.plots.demand.energy_balance.energy_balance(data_frame, analysis_fields, normalize_value, title, output_path)

cea.plots.demand.energy_balance.main()
layout
name = 'Energy End-use'

cea.plots.demand.energy_end_use.calc_graph(analysis_fields, data)

cea.plots.demand.energy_end_use.calc_table(analysis_fields, data_frame)

cea.plots.demand.energy_end_use.calc_top_three_anchor_loads(data_frame, field)

cea.plots.demand.energy_end_use.energy_demand_district(data_frame, analysis_fields, title, output_path)

cea.plots.demand.energy_end_use.main()

cea.plots.demand.energy_end_use_intensity module

class cea.plots.demand.energy_end_use_intensity.EnergyUseIntensityPlot (project, parameters, cache)

Bases: cea.plots.demand.DemandPlotBase

__init__(project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature

calc_graph()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

layout
name = 'Energy End-use Intensity'

cea.plots.demand.energy_end_use_intensity.energy_use_intensity(data_frame, analysis_fields, title, output_path)

cea.plots.demand.energy_end_use_intensity.energy_use_intensity_district(data_frame, analysis_fields, title, output_path)

cea.plots.demand.energy_final_use module

Implements the Energy Supply pot.

class cea.plots.demand.energy_final_use.EnergySupplyPlot (project, parameters, cache)
    Bases: cea.plots.demand.energy_end_use.EnergyDemandDistrictPlot

    Implement the energy-supply plot, inherits most of it's functionality from EnergyDemandDistrictPlot

9.1. cea package
__init__(project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature

layout
    name = 'Energy Final Use'

cea.plots.demand.energy_final_use.main()

cea.plots.demand.energy_use_intensity module

class cea.plots.demand.energy_use_intensity.EnergySupplyIntensityPlot (project, parameters, cache)

    Bases: cea.plots.demand.DemandPlotBase

    __init__(project, parameters, cache)
        x.__init__(...) initializes x; see help(type(x)) for signature

    calc_graph()
        Calculate a plotly Data object as to be passed to the data attribute of Figure

    layout
        name = 'Energy Final Use Intensity'

cea.plots.demand.energy_use_intensity.main()

cea.plots.demand.heating_reset_schedule module

class cea.plots.demand.heating_reset_schedule.HeatingResetSchedulePlot (project, parameters, cache)

    Bases: cea.plots.demand.DemandSingleBuildingPlotBase

    __init__(project, parameters, cache)
        x.__init__(...) initializes x; see help(type(x)) for signature

    calc_graph()
        Calculate a plotly Data object as to be passed to the data attribute of Figure

    data

    layout

    name = 'Heating Reset Schedule'

cea.plots.demand.heating_reset_schedule.heating_reset_schedule (data_frame, analysis_fields, title, output_path)

cea.plots.demand.heating_reset_schedule.main()
cea.plots.demand.load_curve module

class cea.plots.demand.load_curve.LoadCurvePlot (project, parameters, cache)
   Bases: cea.plots.demand.DemandPlotBase

   __init__ (project, parameters, cache)
      x.__init__(...) initializes x; see help(type(x)) for signature

calc_graph ()
   Calculate a plotly Data object as to be passed to the data attribute of Figure

   expected_parameters = {'buildings': 'plots:buildings', 'scenario-name': 'general:scenario-name'}

   layout
      name = 'Load Curve'

   title
      Override the version in PlotBase

cea.plots.demand.load_curve_supply module

Implements the Load Curve Supply plot.

class cea.plots.demand.load_curve_supply.LoadCurveSupplyPlot (project, parameters, cache)
   Bases: cea.plots.demand.DemandPlotBase

   __init__ (project, parameters, cache)
      x.__init__(...) initializes x; see help(type(x)) for signature

calc_graph ()
   Calculate a plotly Data object as to be passed to the data attribute of Figure

   expected_parameters = {'buildings': 'plots:buildings', 'scenario-name': 'general:scenario-name'}

   layout
      name = 'Load Curve Supply'

   title
      Override the version in PlotBase

cea.plots.demand.load_duration_curve module

class cea.plots.demand.load_duration_curve.LoadDurationCurvePlot (project, parameters, cache)
   Bases: cea.plots.demand.DemandPlotBase

   __init__ (project, parameters, cache)
      x.__init__(...) initializes x; see help(type(x)) for signature

calc_graph ()
   Calculate a plotly Data object as to be passed to the data attribute of Figure

   calc_table ()
      Calculates a pandas.DataFrame to display as table.
data
layout
  name = 'Load Duration Curve'
cea.plots.demand.load_duration_curve.calc_graph(analysis_fields, data_frame)
cea.plots.demand.load_duration_curve.calc_table(analysis_fields, data_frame)
cea.plots.demand.load_duration_curve.evaluate_utilization(x, y)
cea.plots.demand.load_duration_curve.load_duration_curve(data_frame, analysis_fields, title, output_path)

cea.plots.demand.load_duration_curve_supply module

Implements the Load Duration Curve Supply plot.

class cea.plots.demand.load_duration_curve_supply.LoadDurationCurveSupplyPlot (project, parameters, cache)
Bases: cea.plots.demand.load_duration_curve.LoadDurationCurvePlot

Implement the load-duration-curve-supply plot

__init__(project, parameters, cache)
  x.__init__(...) initializes x; see help(type(x)) for signature

calc_graph()  
  Calculate a plotly Data object as to be passed to the data attribute of Figure

layout
  name = 'Load Duration Curve Supply'

cea.plots.demand.main module

This file runs all plots of the CEA

class cea.plots.demand.main.Plots (locator, config, buildings)
Bases: object

__init__(locator, config, buildings)
  x.__init__(...) initializes x; see help(type(x)) for signature

comfort_chart (category)
energy_balance (category)
energy_supply (category)
energy_supply_intensity (category)
energy_use (category)
energy_use_intensity (category)
erase_zeros (data, fields)
heating_reset_schedule (category)
load_curve (category)
load_curve_final (category)
load_duration_curve (category)
load_duration_curve_final (category)
peak_load (category)
peak_load_final (category)
preprocess_buildings (buildings)
preprocess_plot_outputpath (buildings)
preprocess_plot_title (buildings)
preprocessing_building_demand ()

cea.plots.demand.main.main (config)
cea.plots.demand.main.plots_main (locator, config)

**cea.plots.demand.peak_load module**

**class** cea.plots.demand.peak_load.PeakLoadCurvePlot (project, parameters, cache)
Bases: cea.plots.demand.DemandPlotBase

__init__ (project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature

calc_graph ()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

layout
    name = 'Peak Demand'

cea.plots.demand.peak_load.diversity_factor (data_frame_timeseries, data_frame_totals, analysis_fields, title, output_path)

cea.plots.demand.peak_load.main ()

cea.plots.demand.peak_load.peak_load_building (data_frame, analysis_fields, title, output_path)

cea.plots.demand.peak_load.peak_load_district (data_frame_totals, analysis_fields, title, output_path)

**cea.plots.demand.peak_load_supply module**

**class** cea.plots.demand.peak_load_supply.PeakLoadSupplyPlot (project, parameters, cache)
Bases: cea.plots.demand.DemandPlotBase

__init__ (project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature

calc_graph ()
    Calculate a plotly Data object as to be passed to the data attribute of Figure
layout
name = 'Peak Supply'

time_series = data_frame_timeseries,

data_frame_totals, analysis_fields, title, output_path)

cea.plots.demand.peak_load_supply.main()

cea.plots.demand.peak_load_supply.peak_load_building(data_frame, analysis_fields, title, output_path)

cea.plots.demand.peak_load_supply.peak_load_district(data_frame_totals, analysis_fields, title, output_path)

Submodules

cea.plots.life_cycle.emissions module

class cea.plots.life_cycle.emissions.EmissionsPlot (project, parameters, cache)
Bases: cea.plots.life_cycle.LifeCycleAnalysisPlotBase

Implements the Green House Gas Emissions plot

calc_graph()

Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_table()

Calculates a pandas.DataFrame to display as table.

layout
name = 'Green House Gas Emissions'

time_series = data_frame_timeseries,

data_frame_totals, analysis_fields, title, output_path)
cea.plots.life_cycle.emissions.calc_table(analysis_fields, data_frame)

cea.plots.life_cycle.emissions.calc_top_three_anchor_loads(data_frame, field)

cea.plots.life_cycle.emissions.emissions(data_frame, analysis_fields, title, output_path)

cea.plots.life_cycle.emissions.main()

Test this plot

**cea.plots.life_cycle.emissions_intensity module**

class `cea.plots.life_cycle.emissions_intensity.EmissionsIntensityPlot`(project, parameters, cache)

Bases: `cea.plots.life_cycle.LifeCycleAnalysisPlotBase`

Implements the Green House Gas Emissions Intensity plot

calc_graph()

Calculate a plotly Data object as to be passed to the data attribute of Figure

layout

name = 'Green House Gas Emissions intensity'

cea.plots.life_cycle.emissions_intensity.calc_graph(analysis_fields, data_frame)

cea.plots.life_cycle.emissions_intensity.emissions_intensity(data_frame, analysis_fields, title, output_path)

cea.plots.life_cycle.emissions_intensity.main()

Test this plot

**cea.plots.life_cycle.main module**

This is the dashboard of CEA

class `cea.plots.life_cycle.main.Plots`(locator, buildings)

Bases: object

__init__(locator, buildings)

x.__init__(...) initializes x; see help(type(x)) for signature

emissions(category)

emissions_intensity(category)

erase_zeros(data, fields)

operation_costs(category)

preprocess_buildings(buildings)

preprocess_plot_outputpath(buildings)

preprocess_plot_title(buildings)

preprocessing_building_costs()

preprocessing_building_emissions()
**primary_energy** (*category*)

**primary_energy_intensity** (*category*)

`cea.plots.life_cycle.main.main(config)`

`cea.plots.life_cycle.main.plots_main(locator, config)`

**cea.plots.life_cycle.operation_costs module**

```python
class cea.plots.life_cycle.operation_costs.OperationCostsPlot(project, parameters, cache)
Bases: cea.plots.life_cycle.LifeCycleAnalysisPlotBase
```

Implement the operation-costs plot

```python
calc_graph()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_table()
    Calculates a pandas.DataFrame to display as table.
```

```python
layout
name = 'Operation Costs'
```

```python
cea.plots.life_cycle.operation_costs.calc_graph(analysis_fields, data_frame)
```

```python
cea.plots.life_cycle.operation_costs.calc_table(analysis_fields, data_frame)
```

```python
cea.plots.life_cycle.operation_costs.calc_top_three_anchor_loads(data_frame, field)
```

```python
cea.plots.life_cycle.operation_costs.main()
Test this plot
```

```python
cea.plots.life_cycle.operation_costs.operation_costs_district(data_frame, analysis_fields, title, output_path)
```

**cea.plots.life_cycle.primary_energy module**

```python
class cea.plots.life_cycle.primary_energy.PrimaryEnergyPlot(project, parameters, cache)
Bases: cea.plots.life_cycle.LifeCycleAnalysisPlotBase
```

Implement the primary-energy plot

```python
calc_graph()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_table()
    Calculates a pandas.DataFrame to display as table.
```

```python
layout
name = 'Primary energy (non-renewable)'
```

```python
cea.plots.life_cycle.primary_energy.calc_graph(analysis_fields, data_frame)
```

```python
cea.plots.life_cycle.primary_energy.calc_table(analysis_fields, data_frame)
```
cea.plots.life_cycle.primary_energy.calc_top_three_anchor_loads (data_frame, field)

Test this plot

cea.plots.life_cycle.primary_energy.main()

cea.plots.life_cycle.primary_energy.primary_energy (data_frame, analysis_fields, title, output_path)

### cea.plots.life_cycle.primary_energy_intensity module

**class** cea.plots.life_cycle.primary_energy_intensity.PrimaryEnergyIntensityPlot (project, parameters, cache)

Bases: cea.plots.life_cycle.LifeCycleAnalysisPlotBase

Implement the primary-energy plot

```python
calc_graph()

Calculate a plotly Data object as to be passed to the data attribute of Figure

layout

name = 'Primary energy intensity (non-renewable)'
```

cea.plots.life_cycle.primary_energy_intensity.calc_graph (analysis_fields, data_frame)

Test this plot

cea.plots.life_cycle.primary_energy_intensity.main()

cea.plots.life_cycle.primary_energy_intensity.primary_energy_intensity (data_frame, analysis_fields, title, output_path)

### cea.plots.optimization package

**class** cea.plots.optimization.GenerationPlotBase (project, parameters, cache)

Bases: cea.plots.base.PlotBase

Implements properties / methods used by all plots in this category

```python
__init__ (project, parameters, cache)

Parameters

• **project** – The project to base plots on (some plots span scenarios)

• **parameters** – The plot parameters as, e.g., per the dashboard.yml file

• **cache** (cea.plots.PlotCache) – a PlotCache instance for speeding up plotting

category_name = 'optimization'
```
normalize_data(data_processed, normalization, analysis_fields)
process_generation_total_performance_halloffame()
process_generation_total_performance_pareto()

Submodules

cea.plots.optimization.Investment_costs module

class cea.plots.optimization.Investment_costs.InvestmentCostsPlot(project, parameters, cache)

Bases: cea.plots.optimization.GenerationPlotBase

Implement the “CAPEX vs. OPEX of centralized system in generation X” plot

__init__(project, parameters, cache)

Parameters
  • project – The project to base plots on (some plots span scenarios)
  • parameters – The plot parameters as, e.g., per the dashboard.yml file
  • cache (cea.plots.PlotCache) – a PlotCache instance for speeding up plotting

calc_graph()
  Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_titles()

expected_parameters = {'generation': 'plots-optimization:generation', 'normalization': 'plots-optimization:normalization',
layout
name = 'Investment costs'
output_path
title

cea.plots.optimization.Investment_costs.main()
Test this plot

cea.plots.optimization.annual_costs module

class cea.plots.optimization.annual_costs.AnnualCostsPlot(project, parameters, cache)

Bases: cea.plots.optimization.GenerationPlotBase

Implement the “CAPEX vs. OPEX of centralized system in generation X” plot

__init__(project, parameters, cache)

Parameters
  • project – The project to base plots on (some plots span scenarios)
  • parameters – The plot parameters as, e.g., per the dashboard.yml file
  • cache (cea.plots.PlotCache) – a PlotCache instance for speeding up plotting
calc_graph()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_titles()
expected_parameters = {'generation': 'plots-optimization:generation', 'normalization'
layout
name = 'Annual costs'
output_path

title
cea.plots.optimization.annual_costs.main()
    Test this plot

cea.plots.optimization.annual_emissions module

class cea.plots.optimization.annual_emissions.AnnualEmissionsPlot (project,
parameters,
cache)
    Bases: cea.plots.optimization.GenerationPlotBase
    Implement the “CAPEX vs. OPEX of centralized system in generation X” plot
__init__ (project, parameters, cache)
    Parameters
    • project – The project to base plots on (some plots span scenarios)
    • parameters – The plot parameters as, e.g., per the dashboard.yml file
    • cache (cea.plots.PlotCache) – a PlotCache instance for speeding up plotting

calc_graph()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_titles()
expected_parameters = {'generation': 'plots-optimization:generation', 'normalization'
layout
name = 'Annual emissions'
output_path

title
cea.plots.optimization.annual_emissions.main()
    Test this plot

cea.plots.optimization.annual_primary_energy module

class cea.plots.optimization.annual_primary_energy.AnnualPENPlot (project,
parameters,
cache)
    Bases: cea.plots.optimization.GenerationPlotBase
    Implement the “CAPEX vs. OPEX of centralized system in generation X” plot
__init__ \((project, parameters, cache)\)

Parameters

- **project** – The project to base plots on (some plots span scenarios)
- **parameters** – The plot parameters as, e.g., per the dashboard.yml file
- **cache** \((cea.plots.PlotCache)\) – a PlotCache instance for speeding up plotting

calc_graph() Calculate a plotly Data object as to be passed to the data attribute of Figure
calc_titles()

expected_parameters = \{'generation': 'plots-optimization:generation', 'normalization'

layout
name = 'Annual primary energy'
output_path
title

cea.plots.optimization.annual_primary_energy.main()

Test this plot

cea.plots.optimization.pareto_curve module

Show a Pareto curve plot for individuals in a given generation.

class cea.plots.optimization.pareto_curve.ParetoCurveForOneGenerationPlot \((project, parameters, cache)\)

Bases: cea.plots.optimization.GenerationPlotBase

Show a pareto curve for a single generation

__init__ \((project, parameters, cache)\)

Parameters

- **project** – The project to base plots on (some plots span scenarios)
- **parameters** – The plot parameters as, e.g., per the dashboard.yml file
- **cache** \((cea.plots.PlotCache)\) – a PlotCache instance for speeding up plotting

calc_graph() Calculate a plotly Data object as to be passed to the data attribute of Figure
calc_titles()

expected_parameters = \{'generation': 'plots-optimization:generation', 'multicriteria'

layout
name = 'Pareto curve of costs, emissions and primary energy'
output_path
title
cea.plots.optimization.pareto_curve.calc_final_dataframe(individual_data)

test this plot

cea.plots.optimization.pareto_curve_convergence module


class cea.plots.optimization.pareto_curve_convergence.OptimizationPerformance(project, parameters, cache)

    Bases: cea.plots.optimization.GenerationPlotBase

    Show a pareto curve for a single generation

    \_
    \_
    __init__(project, parameters, cache)

        Parameters

            • project – The project to base plots on (some plots span scenarios)
            • parameters – The plot parameters as, e.g., per the dashboard.yml file
            • cache(cea.plots.PlotCache) – a PlotCache instance for speeding up plotting

    calc_convergence_metrics()

    calc_graph()

        Calculate a plotly Data object as to be passed to the data attribute of Figure

    expected_parameters = {'generation': 'plots-optimization:generation', 'scenario-name'

        layout

        name = 'Performance of optimization algorithm'

    output_path

    title

    cea.plots.optimization.pareto_curve_convergence.main()

test this plot

cea.plots.solar_potential package

class cea.plots.solar_potential.SolarPotentialPlotBase(project, parameters, cache)

    Bases: cea.plots.base.PlotBase

    Implements properties / methods used by all plots in this category

    \_
    \_
    __init__(project, parameters, cache)

        x.__init__(...) initializes x; see help(type(x)) for signature

    _calculate_input_data_aggregated_kW()

        This is the data all the solar-potential plots are based on.

    add_solar_fields(df1, df2)

        Add the demand analysis fields together - use this in reduce to sum up the summable parts of the dfs
category_name = 'solar-potential'
expected_parameters = {'buildings': 'plots:buildings', 'normalization': 'plots:normalization'}

normalize_data(data_processed, buildings, analysis_fields, analysis_fields_area)
solar_hourly_aggregated_kW()
timeframe_data(data)

Submodules

cea.plots.solar_potential.main module

cea.plots.solar_potential.solar_radiation module

class cea.plots.solar_potential.solar_radiation.SolarRadiationPlot (project, parameters, cache)

    implements the solar-radiation-curve plot
calc_graph()
    Calculate a plotly Data object as to be passed to the data attribute of Figure
calc_titles()
layout
name = 'Solar radiation'
title
    Override the version in PlotBase

cea.plots.solar_potential.solar_radiation.main()
Test this plot

cea.plots.solar_potential.solar_radiation_curve module

cea.plots.solar_potential.solar_radiation_monthly module

cea.plots.solar_technology_potentials package

class cea.plots.solar_technology_potentials.SolarTechnologyPotentialsPlotBase (project, parameters, cache)

    Implements properties / methods used by all plots in this category
    PVT_hourly_aggregated_kW
    PV_hourly_aggregated_kW()
SC_ET_hourly_aggregated_kW()

SC_FP_hourly_aggregated_kW

__init__ (project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature

_calculate_PVT_hourly_aggregated_kW()

_calculate_PV_hourly_aggregated_kW()

_calculate_SC_ET_hourly_aggregated_kW()

_calculate_SC_FP_hourly_aggregated_kW()

add_pv_fields (df1, df2)
    Add the demand analysis fields together - use this in reduce to sum up the summable parts of the dfs

add_sc_et_fields (df1, df2)
    Add the demand analysis fields together - use this in reduce to sum up the summable parts of the dfs

category_name = 'solar-technology-potentials'

expected_parameters = {'buildings': 'plots:buildings', 'scenario-name': 'general:scenario-name'}

normalize_data (data_processed, buildings, analysis_fields, analysis_fields_area)

timeframe_data (data_PV)

Submodules

cea.plots.solar_technology_potentials.all_tech_hourly_curve module

cea.plots.solar_technology_potentials.all_tech_yearly module

cea.plots.solar_technology_potentials.main module

cea.plots.solar_technology_potentials.pv_monthly module

cea.plots.solar_technology_potentials.pvt_monthly module

cea.plots.solar_technology_potentials.sc_monthly module

cea.plots.supply_system package

class cea.plots.supply_system.SupplySystemPlotBase (project, parameters, cache)
    Bases: cea.plots.base.PlotBase

    Implements properties / methods used by all plots in this category

    __init__ (project, parameters, cache)
        x.__init__(...) initializes x; see help(type(x)) for signature

    category_name = 'supply-system'

    expected_parameters = {'generation': 'plots-supply-system:generation', 'individual':

    process_connected_capacities_kW()

    process_disconnected_capacities_kW()
process_individual_dispatch_curve_cooling()
process_individual_dispatch_curve_electricity()
process_individual_dispatch_curve_heating()
process_individual_ramping_capacity()
process_individual_requirements_curve_electricity()

Submodules

kea.plots.supply_system.dispatch_curve_cooling_plant module

Show a Pareto curve plot for individuals in a given generation.

```
class kea.plots.supply_system.dispatch_curve_cooling_plant.DispatchCurveDistrictCoolingPlot
```

Bases: kea.plots.supply_system.SupplySystemPlotBase

Show a pareto curve for a single generation

```
__init__(project, parameters, cache)
x.__init__(...) initializes x; see help(type(x)) for signature
```

```
calc_graph()
```

Calculate a plotly Data object as to be passed to the data attribute of Figure

```
expected_parameters = {'generation': 'plots-supply-system:generation', 'individual': 'plots-supply-system:individual', 'scenario-name': 'general:scenario-name', 'timeframe': 'plots-supply-system:timeframe'}
```

```
layout
title
```

kea.plots.supply_system.dispatch_curve_cooling_plant.main()

Test this plot

kea.plots.supply_system.dispatch_curve_electricity module

Show a Pareto curve plot for individuals in a given generation.

```
class kea.plots.supply_system.dispatch_curve_electricity.DispatchCurveDistrictElectricityPlot
```

Bases: kea.plots.supply_system.SupplySystemPlotBase

Show a pareto curve for a single generation

```
__init__(project, parameters, cache)
x.__init__(...) initializes x; see help(type(x)) for signature
```

kea.plots.supply_system.dispatch_curve_electricity.main()
calc_graph()
  Calculate a plotly Data object as to be passed to the data attribute of Figure

layout
  name = 'Dispatch curve electricity'
output_path
title
  cea.plots.supply_system.dispatch_curve_electricity.main()
  Test this plot

cea.plots.supply_system.dispatch_curve_heating_plant module

Show a Pareto curve plot for individuals in a given generation.

class cea.plots.supply_system.dispatch_curve_heating_plant.DispatchCurveDistrictHeatingPlot
  Bases: cea.plots.supply_system.SupplySystemPlotBase
  Show a pareto curve for a single generation
  __init__(project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature
  calc_graph()
    Calculate a plotly Data object as to be passed to the data attribute of Figure
  expected_parameters = {'generation': 'plots-supply-system:generation', 'individual':
    layout
      name = 'Dispatch curve heating plant'
output_path
title
  cea.plots.supply_system.dispatch_curve_heating_plant.main()
  Test this plot

cea.plots.supply_system.grid_ramping_capacity module

Show a Pareto curve plot for individuals in a given generation.

class cea.plots.supply_system.grid_ramping_capacity.RampingCapacity
  Bases: cea.plots.supply_system.SupplySystemPlotBase
  Show a pareto curve for a single generation
  __init__(project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature
calc_graph()

Calculate a plotly Data object as to be passed to the data attribute of Figure

expected_parameters = {'generation': 'plots-supply-system:generation', 'individual': 'plots-supply-system:individual'
layout
name = 'Electrical Grid Impact (ramp-rate)'
output_path

title

test this plot

test this plot

test this plot

test this plot

test this plot
____init__(project, parameters, cache)
x.__init__(...) initializes x; see help(type(x)) for signature

calc_graph()
Calculate a plotly Data object as to be passed to the data attribute of Figure

expected_parameters = {'generation': 'plots-supply-system:generation', 'individual':
layout
name = 'Requirements curve electricity'
output_path
title

Test this plot

cea.plots.supply_system.supply_system_map module

Show a Pareto curve plot for individuals in a given generation.

class cea.plots.supply_system.supply_system_map.SupplySystemMapPlot (project,
parameters,
cache)

Bases: cea.plots.supply_system.SupplySystemPlotBase

Show a pareto curve for a single generation

____init__(project, parameters, cache)
x.__init__(...) initializes x; see help(type(x)) for signature

_plot_div_producer()
Since this plot doesn’t use plotly to plot, we override _plot_div_producer to return a string containing the html div to use for this plot. The template map_div.html expects some parameters:

Here is some example data (in a YAML-like format for documentation purposes)

<table>
<thead>
<tr>
<th>data:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DH:</td>
</tr>
<tr>
<td>connected_buildings: ['B1010', 'B1017', 'B1003']</td>
</tr>
<tr>
<td>disconnected_buildings: ['B1000', 'B1009', 'B1016', ..., 'B1015']</td>
</tr>
</tbody>
</table>
| path_output_nodes: "(general:scenario)/inputs/networks/DH/gen_3_ind_1/
  nodes.shp" |
| path_output_edges: "(general:scenario)/inputs/networks/DH/gen_3_ind_1/
  edges.shp" |
| DC: () # data does not necessarily contain information for both types of |
| district networks |
| colors:           |
| dc: [63, 192, 194] |
| dh: [240, 75, 91]  |
| disconnected: [68, 76, 83] |
| district: [255, 255, 255] |
| zone: str serialization of the GeoJSON of the zone.shp |
| district: str serialization of the GeoJSON of the district.shp |
| dc: str serialization of a GeoJSON containing both the nodes.shp + edges.shp |
| of district cooling network |
| dh: str serialization of a GeoJSON containing both the nodes.shp + edges.shp |
| of district heating network |
Returns a str containing a full html `<div/>` that includes the js code to display the map.

```python
create_network_layout(connected_buildings, network_type, network_name)
```

data_processing()

```python
expected_parameters = {'generation': 'plots-supply-system:generation', 'individual':

get_network_json(edges, nodes)
```

name = 'Supply system map'

output_path
title

```python
cea.plots.supply_system.supply_system_map.get_building_connectivity(locator)
```

```python
cea.plots.supply_system.supply_system_map.main()
```

Test this plot

**cea.plots.thermal_networks package**

```python
class cea.plots.thermal_networks.ThermalNetworksMapPlotBase(project, parameters, cache)

Bases: cea.plots.thermal_networks.ThermalNetworksPlotBase
```

Some of the plots in the Thermal Networks category display their data on a map (using deck.gl)

This works by using the Jinja2 templating engine to create a html `<div/>` containing all the javascript necessary to show the plot. The template used is `network_plot.html`.

```python
__init__(project, parameters, cache)
x.__init__(...) initializes x; see help(type(x)) for signature
```

```python
_plot_div_producer()
```

Since this plot doesn’t use plotly to plot, we override _plot_div_producer to return a string containing the html div to use for this plot. The template `network_plot.html` expects some parameters:

- hash: this is used to make the html id’s in the plot unique
- edges: a GeoJson serialization of the networks edges and data
- nodes: a GeoJson serialization of the networks nodes and data,
- colors: a JSON dictionary of [R, G, B] arrays for the colors to use
- zone: a GeoJson serialization of the zone’s buildings
- district: a GeoJson serialization of the district’s buildings

Returns a str containing a full html `<div/>` that includes the js code to display the map.

**edges_df**

This property is expected to return a GeoDataFrame containing the edges of the network to display including the data.

Any columns included will be shown in the Tooltip of the map, except for those starting with an underscore.

There are some special columns that must be added here, that are used for visualization purposes:

- _LineWidth: The line width to use for the edges. This can be a computed property based on edge data

Returns
Return type: geopandas.GeoDataFrame

**nodes_df**
This property is expected to return a GeoDataFrame containing the nodes of the network to display including the data.

Any columns included will be shown in the Tooltip of the map, except for those starting with an underscore.

There are some special columns that must be added here, that are used for visualization purposes:
- 
  _Radius: The radius of the node. This can be a computed property based on node data.
- 
  _FillColor: The color ([R, G, B]) to use for the line, serialized as JSON. This can be a computed property based on node data.

Returns
Return type: geopandas.GeoDataFrame

class cea.plots.thermal_networks.ThermalNetworksPlotBase (project, parameters, cache)

Bases: cea.plots.base.PlotBase

Implements properties / methods used by all plots in this category

P_loss_kWh
__init__ (project, parameters, cache)
  x.__init__(...) initializes x; see help(type(x)) for signature

_calculate_relative_loss (absolute_loss)
  Calculate relative heat or pressure loss: 1. Sum up all plant heat produced in each time step 2. Divide absolute losses by that value

buildings_hourly
category_name = 'thermal-networks'
date
  Read in the date information from demand results of the first building in the zone

expected_parameters = {'network-name': 'plots:network-name', 'network-type': 'plots:network-type', 'scenario-name': 'general:scenario-name'}

hourly_heat_loss
hourly_loads
linear_pressure_loss_Paperm
mass_flow_kgs_nodes
mass_flow_kgs_pipes
network_pipe_length
output_path
plant_pumping_requirement_kWh
pressure_at_nodes_Pa
temperature_return_nodes_C
  Node return temperatures

temperature_supply_nodes_C
  Node supply temperatures
temperature_supply_return_plant_C
  Node supply temperatures
thermal_loss_edges_Wperm
thermal_loss_edges_kWh
title
  Override the version in PlotBase
total_thermal_losses_kWh
velocity_mps_pipes

Submodules

cea.plots.thermal_networks.1network_design module

cea.plots.thermal_networks.2demand_curve module
class cea.plots.thermal_networks.2demand_curve.LossCurvePlot (project, parameters, cache)
  Bases: cea.plots.thermal_networks.ThermalNetworksPlotBase
  Implement the heat and pressure losses plot
  __init__(project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature
calc_data_frame()
calc_graph()
  Calculate a plotly Data object as to be passed to the data attribute of Figure
layout
  name = 'Load curve of Heat and Pressure Losses'
cea.plots.thermal_networks.2demand_curve.main()
  Test this plot

cea.plots.thermal_networks.3annual_energy_consumption module
class cea.plots.thermal_networks.3annual_energy_consumption.AnnualEnergyConsumptionPlot (project, parameters, cache)
  Bases: cea.plots.thermal_networks.ThermalNetworksPlotBase
  Implement the Annual energy consumption plot
  __init__(project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature
calc_graph()
  Calculate a plotly Data object as to be passed to the data attribute of Figure
layout
name = 'Yearly energy consumption'

Test this plot

tt

class cea.plots.thermal_networks.4energy_loss_bar.EnergyLossBarPlot (project, parameters, cache)

Bases: cea.plots.thermal_networks.ThermalNetworksPlotBase
Implement the Thermal losses and pumping requirements per pipe plot
__init__ (project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature

calc_graph ()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_table ()
    Calculates a pandas.DataFrame to display as table.

layout

name = 'Thermal losses and pumping requirements per pipe'

tt

class cea.plots.thermal_networks.5heating_reset_curve.SupplyReturnAmbientCurvePlot (project, parameters, cache)

Bases: cea.plots.thermal_networks.ThermalNetworksPlotBase
Implement the heat and pressure losses plot
__init__ (project, parameters, cache)
    x.__init__(...) initializes x; see help(type(x)) for signature

ambient_temp
    Read in ambient temperature data at first building. This assumes that all buildings are relatively close to each other and have the same ambient temperature.

calc_graph ()
    Calculate a plotly Data object as to be passed to the data attribute of Figure

expected_parameters = {'network-name': 'plots:network-name', 'network-type': 'plots:network-type'}

layout

name = 'Heating Reset Curve'
output_path
plant_node
plant_temperatures
title
  Override the version in ThermalNetworksPlotBase

cea.plots.thermal_networks.5heating_reset_curve.main()
  Test this plot

cea.plots.thermal_networks.5heating_reset_curve.supply_return_ambient_temp_plot(data_frame, data_frame_2, analysis_fields, title, output_path)

cea.plots.thermal_networks.6pump_duration_curve module

class cea.plots.thermal_networks.6pump_duration_curve.LoadDurationCurvePlot(project, parameters, cache)

  Bases: cea.plots.thermal_networks.ThermalNetworksPlotBase
  Implement the load duration curve of pump plot

  __init__(project, parameters, cache)
    x.__init__(...) initializes x: see help(type(x)) for signature

calc_graph()
  Calculate a plotly Data object as to be passed to the data attribute of Figure

calc_table()
  Calculates a pandas.DataFrame to display as table.

layout
  name = 'Pumping Duration Curve'

cea.plots.thermal_networks.6pump_duration_curve.calc_graph(analysis_fields, data_frame)

cea.plots.thermal_networks.6pump_duration_curve.calc_table(analysis_fields, data_frame)

cea.plots.thermal_networks.6pump_duration_curve.evaluate_utilization(x, y)

cea.plots.thermal_networks.6pump_duration_curve.loss_duration_curve(data_frame, analysis_fields, title, output_path)
cea.plots.thermal_networks.6pump_duration_curve.main()

Test this plot

**Submodules**

**cea.plots.base module**

Implements base classes to derive plot classes from. The code in `py:mod:cea.plots.categories` uses `py:class:cea.plots.base.PlotBase` to figure out the list of plots in a category.

```python
class cea.plots.base.PlotBase (project, parameters, cache)
    Bases: object

    A base class for plots containing helper methods used by all plots.

    __init__ (project, parameters, cache)
        x.__init__(...) initializes x; see help(type(x)) for signature

    _plot_div_producer ()

    _table_div_producer ()
        Default producer for table divs (override if you need more control)

    calc_graph ()
        Calculate a plotly Data object as to be passed to the data attribute of Figure

    calc_table ()
        Calculates a pandas.DataFrame to display as table.

    category_name = None

    category_path = None

    expected_parameters = {}

    @classmethod
    get_default_parameters (config)
        Return a dictionary of parameters taken by using the values in the config file

    id ()

    layout

    locator
        Returns cea.inputlocator.InputLocator

    missing_input_files ()
        Return the list of missing input files for this plot

    name = None

    output_path
        The output path to use for the solar-potential plots

    plot (auto_open=False)
        Plots the graphs to the filename (see output_path)

    plot_div ()
        Return the plot as an html <div/> for use in the dashboard. Override this method in subclasses

    process_buildings_parameter ()
        Make sure the buildings parameter contains only buildings in the zone. Returns (and updates) the parameter.
```
**remove_unused_fields** *(data, fields)*

Helper method that, given a data frame and a list of fields in that data frame, returns the subset of those fields that actually have data.

FIXME: what about columns with negative values?

**table_div()**

Returns the html div for a table, or an empty string if no table is to be produced

**title**

Override the version in PlotBase

**totals_bar_plot()**

Creates a plot based on the totals data in percentages.

---

### `cea.plots.cache` module

Implements a cache for plot data at the project level. Cached plot data has a “path” (e.g. ‘optimization/generations_data’) and dependencies (a list of files that are used to produce that data) as well as the parameters used in that data. The cache object is passed to the `calc_graph` method and the plot is responsible for retrieving data from the cache.

**class`cea.plots.cache.MemoryPlotCache`**(project)**

Implements the `PlotCache` to also keep a copy of the cache in memory

**__init__**(project)**

Initialize the cache from disk

**load_cached_value**(data_path, parameters)**

Check memory cache before loading from disk

**store_cached_value**(data_path, parameters, producer)**

Update memory cache when storing to disk

**class`cea.plots.cache_NullPlotCache`**

A dummy cache that doesn’t cache anything - for comparing performance of `PlotCache`

**__init__**()

Initialize the cache from disk

**lookup**(data_path, plot, producer)**

**lookup_plot_div**(plot, producer)**

Lookup the cache of a plot created with `plot.plot_div()`

**lookup_table_div**(plot, producer)**

Lookup the cache of a table created with `plot.table_div()`

**class`cea.plots.cache.PlotCache`**(project)**

A cache for plot data. Use the `lookup` method to retrieve data from the cache.

**__init__**(project)**

Initialize the cache from disk

**cached_data_file**(data_path, parameters)**

**cached_div_file**(plot)**
```python
_cache_table_file(plot)
_parameter_hash(parameters)

_cache_timestamp(path)
    Return a timestamp (like `os.path.getmtime`) to compare to. Returns 0 if there is no data in the cache

_load_cached_value(data_path, parameters)
    Load a Dataframe from disk

_lookup(data_path, plot, producer)

_lookup_plot_div(plot, producer)
    Lookup the cache of a plot created with `plot.plot_div()`

_lookup_table_div(plot, producer)
    Lookup the cache of a table created with `plot.table_div()`

_newest_dependency(input_files)
    Returns the newest timestamp (`os.path.getmtime` and `time.time()`) of the input_files - the idea being, that if the cache is newer than this, then the cache is valid.

    Parameters
    input_files -- A list of tuples (locator method, args) that, when applied, produce a path

_store_cached_value(data_path, parameters, producer)
    Store the Dataframe returned from producer and return it.

c. plots.cache.cached(producer)
    Calls to a function wrapped with this decorator are cached using `self.cache.lookup`

c. plots.categories module

Lists the plots by category. See `cea/plots/__init__.py` for documentation on how plots are organized and the conventions for adding new plots.

c. plots.categories.PlotCategory(module)
    Bases: object
    Contains the data of a plot category.

    __init__(module)
    x.__init__(...) initializes x; see help(type(x)) for signature

plots
    Returns Generator[PlotBase]

c. plots.categories.is_valid_category(category)
    True, if category is the name (not the label) of a valid CEA plot category

c. plots.categories.list_categories()
    List all the categories implemented in the CEA

c. plots.categories.load_category(category_name)
    Returns a PlotsCategory object if is_valid_category(category), else None

c. plots.categories.load_plot(category_name, plot_name)

c. plots.categories.load_plot_by_id(category_name, plot_id)
    plot_id is a web-friendly way of expressing the plot’s name (which is more english friendly)
```
**cea.plots.plots_main module**

This is the dashboard of CEA

```
cea.plots.plots_main.main(config)
cea.plots.plots_main.plots_main(config)
```

**cea.plots.variable_naming module**

```
cea.plots.variable_naming.get_color_array(color)
```

**cea.resources package**

Subpackages

**cea.resources.radiation_daysim package**

Submodules

**cea.resources.radiation_daysim.daysim_main module**

```
cea.resources.radiation_daysim.daysim_main.calc_sensors_building(building_geometry_dict, settings)
cea.resources.radiation_daysim.daysim_main.calc_sensors_zone(geometry_3D_zone, locator, settings)
cea.resources.radiation_daysim.daysim_main.create_sensor_input_file(rad, chunk_n)
cea.resources.radiation_daysim.daysim_main.generate_sensor_surfaces(occface, wall_dim, roof_dim, srf_type, orientation, normal, intersection)
```

```
cea.resources.radiation_daysim.daysim_main.isolation_daysim(chunk_n, rad, geometry_3D_zone, locator, settings, max_global, weatherfile)
```

**cea.resources.radiation_daysim.geometry_generator module**

Geometry generator from Shapefiles (building footprint) and .tiff (terrain)
into 3D geometry with windows and roof equivalent to LOD3
class `cea.resources.radiation_daysim.geometry_generator.BuildingData`(
    locator,
    settings,
    geometry_terrain,
    height_col,
    nfloor_col
)

Bases: `object`

```
__init__ (locator, settings, geometry_terrain, height_col, nfloor_col)
x.__init__(...) initializes x; see help(type(x)) for signature
```

calc_surrounding_building_solids (name)
calc_zone_building_solids (name)
surroundings_building_records ()
terrain_intersection_curves (geometry_terrain)

class `cea.resources.radiation_daysim.geometry_generator.BuildingDataFinale`(
    surroundings_building_solid_list,
    architecture_wwr_df
)

Bases: `object`

```
__init__ (surroundings_building_solid_list, architecture_wwr_df)
x.__init__(...) initializes x; see help(type(x)) for signature
```

cea.resources.radiation_daysim.geometry_generator.blockPrint ()

cea.resources.radiation_daysim.geometry_generator.building_2d_to_3d (locator,
geometry_terrain,
config,
height_col,
nfloor_col)

Parameters

- **locator** (*cea.inputlocator.InputLocator*) – InputLocator - provides paths to files in a scenario
- **config** (*cea.config.Configuration*) – the configuration object to use
- **height_col** – name of the columns storing the height of buildings
- **nfloor_col** – name of the column storing the number of floors in buildings.

Returns

cea.resources.radiation_daysim.geometry_generator.burn_buildings (geometry,
terrain_intersection_curves)

cea.resources.radiation_daysim.geometry_generator.calc_building_geometry_surroundings (name,
building_solid)

cea.resources.radiation_daysim.geometry_generator.calc_building_geometry_zone (name,
building_zone,
data_preprocessed)
This script calculates the intersection of the building edges to the terrain.

```python
terra_intersections, edges_coords, edges_dir)
```

This function returns the intersecting points and intersecting faces.

```python
terra_intersections_face_solid(index, data_processed)
```

This function computes the solid for each intersecting face.

```python
terra_intersections_walls(facade_list, wwr, data_processed)
```

This function calculates the windows and walls based on the facade list, WWR, and data processed.

```python
create_hollowed_facade(surface_facade, window)
```

This function creates a hollowed facade.

```python
create_windows(surface, wwr, ref_pypt)
```

This function creates windows based on the surface, WWR, and reference point.

```python
enablePrint()
```

This function enables printing.

```python
geometry_main(locator, config)
```

This function runs the geometry main process.

```python
identify_surfaces_type(occface_list)
```

This function identifies the surfaces type.

```python
print_progress(i, n, args, result)
```

This function prints progress updates.

```python
raster_to_tin(input_terrain_raster)
```

This function converts raster to tin.

### `cea.resources.radiation_daysim.radiation_main` module

Radiation engine and geometry handler for CEA

```python
add_rad_mat(daysim_mat_file, geometry_table)
```

This function adds radiation materials to the geometry table.

```python
buildings_to_radiance(rad, building_surface_properties, geometry_3D_zone, geometry_3D_surroundings)
```

This function builds buildings for radiation analysis.

```python
calc_transmissivity(G_value)
```

This function calculates window transmissivity from its transmittance using an empirical equation from Radiance.

**Parameters**

- **G_value** – Solar energy transmittance of windows (dimensionless)

**Returns**

Transmissivity


```python
check_daysim_bin_directory(path_hint)
```

This function checks for the Daysim bin directory based on path_hint and returns it on success.
If the binaries could not be found there, check in a folder Dependencies/Daysim of the installation - this will catch installations on Windows that used the official CEA installer.

Check the RAYPATH environment variable. Return that.

Check for C:\Daysim - it might be there?

If the binaries can’t be found anywhere, raise an exception.

**Parameters** path_hint (**str**) – The path to check first, according to the cea.config file.

**Returns** path_hint, contains the Daysim binaries - otherwise an exception occurs.

```python
c.cea.resources.radiation_daysim.radiation_main.create_radiance_srf(occface, srfname, srfmat, rad)
```

```python
c.cea.resources.radiation_daysim.radiation_main.main(config)
```

This function makes the calculation of solar insolation in X sensor points for every building in the zone of interest. The number of sensor points depends on the size of the grid selected in the config file and are generated automatically.

**Parameters** config (**cea.config.Configuration**) – Configuration object with the settings (genera and radiation)

**Returns**

```python
c.cea.resources.radiation_daysim.radiation_main.radiation_singleprocessing(rad, geometry_3D_zone, locator, settings)
```

```python
c.cea.resources.radiation_daysim.radiation_main.reader_surface_properties(locator, input_shp)
```

This function returns a dataframe with the emissivity values of walls, roof, and windows of every building in the scene.

```python
c.cea.resources.radiation_daysim.radiation_main.terrain_to_radiance(rad, tin_occfaceTerrain)
```

**cea.resources.radiation_daysim.visualization module**

**Submodules**

**cea.resources.geothermal module**

```python
c.cea.resources.geothermal.calc_area_buildings(locator, buildings_list)
```

```python
c.cea.resources.geothermal.calc_geothermal_potential(locator, config)
```

A very simplified calculation based on the area available
cea.resources.geothermal.calc_ground_temperature(locator, T_ambient_C, depth_m)
Calculates hourly ground temperature fluctuation over a year following [Kusuda, T. et al., 1965].

Parameters
- T_ambient_C (np array) – vector with outdoor temperature
- depth_m – depth

Return T_ground_K vector with ground temperatures in [K]
Rtype T_ground_K np array


cea.resources.geothermal.calc_temperature_underground(T_amplitude_K, T_avg, conductivity_soil, density_soil, depth_m, heat_capacity_soil)

cea.resources.geothermal.main(config)

cea.resources.natural_gas module

natural gas

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

Parameters PnomGas (float) – peak natural gas supply in [W]

Returns InvCa
Rtype InvCa

cea.resources.sewage_heat_exchanger module

Sewage source heat exchanger

cea.resources.sewage_heat_exchanger.calc_Sewagetemperature(Qwwf, Qww, tsww, trww, mcp_tw, mcpww, 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
- SW_ratio (float) – ratio of decrease/increase in sewage water due to solids and also water intakes.

Returns mcp_combi sewage water heat capacity [kW_K]
Rtype mcp_combi float
Returns \( t_{\text{to\_sewage}} \) sewage water temperature

Rtype \( t_{\text{to\_sewage}} \) float

c.cea.resources.sewage_heat_exchanger.calc_sewage_heat_exchanger(locator, config)
Calaculate the heat extracted from the sewage HEX.

Parameters

- \( \text{locator} \) – an InputLocator instance set to the scenario to work on
- \( \text{Length\_HEX\_available} (\text{float}) \) – HEX length available

Save the results to \( \text{SWP}\_\text{csv} \)

c.cea.resources.sewage_heat_exchanger.calc_sewageheat(mcp\_kWC\_zone, \ t_{\text{in}}\_\text{C}, \ w\_\text{HEX}\_m, \ Vf\_ms, \ h0, \ \text{min\_lps}, \ L\_\text{HEX}\_m, \ \ t_{\text{min}}\_\text{C}, \ \text{ATmin}, \ V\_lps\_external)
Calculates the operation of sewage heat exchanger.

Parameters

- \( \text{mcp\_kWC\_total} (\text{float}) \) – heat capacity of total sewage in a zone
- \( \ t_{\text{in}}\_\text{C} (\text{float}) \) – sewage inlet temperature of a zone
- \( \ w\_\text{HEX}\_m (\text{float}) \) – width of the sewage HEX
- \( \ Vf\_ms (\text{float}) \) – sewage flow rate [m/s]
- \( \ cp (\text{float}) \) – water specific heat capacity
- \( \ h0 (\text{float}) \) – sewage heat transfer coefficient
- \( \ \text{min\_lps} (\text{float}) \) – sewage minimum flow rate in [lps]
- \( \ L\_\text{HEX}\_m (\text{float}) \) – HEX length available
- \( \ \ t_{\text{min}}\_\text{C} (\text{float}) \) – minimum temperature of extraction
- \( \ \text{ATmin} (\text{float}) \) – minimum area of heat exchange

Returns \( Q_{\text{source}} \) heat supplied by sewage

Return type \( \text{float} \)

Returns \( t_{\text{source}} \) sewage heat supply temperature

Rtype \( t_{\text{source}} \) float

Returns \( t_{\text{b2}} \) sewage return temperature

Rtype \( t_{\text{bs}} \) float

Returns \( t_{\text{a1}} \) temperature inlet of the cold stream (from the HP)

Rtype \( t_{\text{a1}} \) float

Returns \( t_{\text{a2}} \) temperature outlet of the cold stream (to the HP)

Rtype \( t_{\text{a2}} \) float

cea.resources.water_body_potential module

Sewage source heat exchanger

```python
cea.resources.water_body_potential.calc_lake_potential(locator, config)
```

Quick calculation of lake potential. This does not refer to CEA original publication. In that case, the implementation of the Lake potential algorithm was carried out with another tool and then the results were implemented in CEA for a specific case study # TODO: create proper lake potential model

```python
cea.resources.water_body_potential.main(config)
```

cea.technologies package

Subpackages

cea.technologies.network_layout package

Submodules

cea.technologies.network_layout.connectivity_potential module

This script uses libraries in shapely to create connections from a series of points (buildings) to the closest street

```python
cea.technologies.network_layout.connectivity_potential.bend_towards(line, where, to)
```

Move the point where along a line to the point at location to.

- **Args:** line: a LineString where: a point ON the line (not necessarily a vertex) to: a point NOT on the line where the nearest vertex will be moved to
- **Returns:** the modified (bent) line

```python
cea.technologies.network_layout.connectivity_potential.calc_connectivity_network(path_streets_shp, path_connection_point_buildings_shp, path_potential_network)
```

This script outputs a potential network connecting a series of building points to the closest street network the street network is assumed to be a good path to the district heating or cooling network

- **Parameters**
  - `path_streets_shp` – path to street shapefile
  - `path_connection_point_buildings_shp` – path to substations in buildings (or close by)
  - `path_potential_network` – output path shapefile
- **Returns**

```python
cea.technologies.network_layout.connectivity_potential.calculate_end_points_intersections(prototype_network, crs)
```

```python
cea.technologies.network_layout.connectivity_potential.compute_intersections(lines, crs)
```

```python
cea.technologies.network_layout.connectivity_potential.compute_end_points(lines, crs)
```
cea.technologies.network_layout.connectivity_potential.create_terminals(building_centroids, crs, street_network)

Find endpoints of lines that don’t touch another line.

Args: lines: a list of LineStrings or a MultiLineString

Returns: A list of line end Points that don’t touch any other line of lines

cea.technologies.network_layout.connectivity_potential.find_isolated_endpoints(lines)

Find endpoints of lines that don’t touch another line.

Args: lines: a list of LineStrings or a MultiLineString

Returns: A list of line end Points that don’t touch any other line of lines

cea.technologies.network_layout.connectivity_potential.main(config)

cea.technologies.network_layout.connectivity_potential.near_analysis(building_centroids, street_network, crs)

cea.technologies.network_layout.connectivity_potential.nearest_neighbor_within(others, point, max_distance)

Find nearest point among others up to a maximum distance.

Args: others: a list of Points or a MultiPoint point: a Point max_distance: maximum distance to search for the nearest neighbor

Returns: A shapely Point if one is within max_distance, None otherwise

cea.technologies.network_layout.connectivity_potential.one_linestring_per_intersection(lines, crs)

Move line endpoints to intersections of line segments.

Given a list of touching or possibly intersecting LineStrings, return a list LineStrings that have their endpoints at all crossings and intersecting points and ONLY there.

Args: a list of LineStrings or a MultiLineString

Returns: a list of LineStrings

cea.technologies.network_layout.connectivity_potential.simplify_liness_accuracy(lines, decimals, crs)

cea.technologies.network_layout.connectivity_potential.simplify_points_accuracy(building_centroids, decimals, crs)

cea.technologies.network_layout.connectivity_potential.snap_points(points, lines, crs)

cea.technologies.network_layout.connectivity_potential.snappy_endings(lines, max_distance, crs)

Snap endpoints of lines together if they are at most max_length apart.

Args: lines: a list of LineStrings or a MultiLineString max_distance: maximum distance two endpoints may be joined together
cea.technologies.network_layout.connectivity_potential.split_line_by_nearest_points(gdf_line, gdf_points, tolerance, crs)

Split the union of lines with the union of points resulting

Parameters

• gdf_line (GeoDataFrame) – GeoDataFrame with multiple rows of connecting line segments
• gdf_points (GeoDataFrame) – geodataframe with multiple rows of single points

Returns gdf_segments (GeoDataFrame of segments)

Return type GeoDataFrame

https://github.com/ojdo/python-tools/blob/master/shapelytools.py#L144

cea.technologies.network_layout.connectivity_potential.vertices_from_lines(lines)

Return list of unique vertices from list of LineStrings.

gdf_line (GeoDataFrame) – GeoDataFrame with multiple rows of connecting line segments

gdf_points (GeoDataFrame) – geodataframe with multiple rows of single points

Returns gdf_segments (GeoDataFrame of segments)

Return type GeoDataFrame

class cea.technologies.network_layout.main.NetworkLayout(network_layout=None)

Bases: object

Capture network layout information

__init__(network_layout=None)

x.__init__(...) initializes x; see help(type(x)) for signature

cea.technologies.network_layout.main.layout_network(network_layout, locator, plant_building_names=[], input_path_name='streets', output_name_network='', optimization_flag=False)

This script calculates the minimum spanning tree of a shapefile network

calc_minimum_spanning_tree(input_network, output_network, building_nodes_shp, output_edges, output_nodes, weight_field, type_mat_default, pipe_diameter)
cea.technologies.network_layout.minimum_spanning_tree.main(config)

**cea.technologies.network_layout.steiner_spanning_tree module**

This script calculates the minimum spanning tree of a shapefile network:

- `add_loops_to_network(G, mst_non_directed, new_mst_nodes, mst_edges, type_mat, pipe_dn)`
- `add_plant_close_to_anchor(building_anchor, new_mst_nodes, mst_edges, type_mat, pipe_dn)`
- `building_node_from_name(building_name, nodes_df)`
- `calc_coord_anchor(total_demand_location, nodes_df, type_network)`
- `calc_steiner_spanning_tree(crs_projected, input_network_shp, output_network_folder, building_nodes_shp, output_edges, output_nodes, weight_field, type_mat_default, pipe_diameter_default, type_network, total_demand_loc, create_plant, allow_looped_network, optimization_flag, plant_building_names, disconnected_building_names)`

Calculate the minimum spanning tree of the network. Note that this function can’t be run in parallel in it's
City Energy Analyst Documentation, Release 2.29.0

present form.

Parameters

- **crs_projected** (*str*) – e.g. “+proj=utm +zone=48N +ellps=WGS84 +datum=WGS84 +units=m +no_defs”
- **input_network_shp** (*str*) – e.g. “TEMP/potential_network.shp”
- **output_network_folder** (*str*) – “{general:scenario}/inputs/networks/DC”
- **building_nodes_shp** (*str*) – e.g. “%TEMP%/nodes_buildings.shp”
- **output_edges** (*str*) – “{general:scenario}/inputs/networks/DC/edges.shp”
- **output_nodes** (*str*) – “{general:scenario}/inputs/networks/DC/nodes.shp”
- **weight_field** (*str*) – e.g. “Shape_Leng”
- **type_mat_default** (*str*) – e.g. “T1”
- **pipe_diameter_default** (*float*) – e.g. 150
- **type_network** (*str*) – “DC” or “DH”
- **total_demand_location** (*str*) – “{general:scenario}/outputs/data/demand/Total_demand.csv”
- **create_plant** (*bool*) – e.g. True
- **allow_looped_networks** (*bool*) –
- **optimization_flag** (*bool*) –
- **plant_building_names** (*List[str]*) – e.g. ['B001']
- **disconnected_building_names** (*List[str]*) – e.g. ['B002', 'B010', 'B004', 'B005', 'B009']

Returns (mst_edges, new_mst_nodes)

```python
cea.technologies.network_layout.steiner_spanning_tree.main(config)
```

**cea.technologies.network_layout.substations_location module**

This script calculates the location of substations in case we do not have it. It is estimated as the centroid of buildings.

```python
cea.technologies.network_layout.substations_location.calc_substation_location(input_buildings_shp, output_substations_shp, connected_buildings, consider_only_building, type_network='DH', total_demand=False)
```

**cea.technologies.solar package**

**Submodules**
**cea.technologies.solar.constants module**

Parameters used for solar technologies

**cea.technologies.solar.photovoltaic module**

Photovoltaic

**cea.technologies.solar.photovoltaic.calc_Cinv_pv** *(total_module_area_m2, locator, technology=0)*

To calculate capital cost of PV modules, assuming 20 year system lifetime. 
:param \( P_{\text{peak}} \): installed capacity of PV module [kW]
:return \( \text{InvCa} \): capital cost of the installed PV module [CHF/Y]

**cea.technologies.solar.photovoltaic.calc_Crem_pv** *(\( E_{\text{nom}} \))*

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** *(float)* – Nominal Capacity of solar panels (PV or PVT) [Wh]

**Return**
- **KEV_obtained_in_RpPerkWh** – KEV remuneration [Rp/kWh]

**Rtype**
- **KEV_obtained_in_RpPerkWh** – float

**cea.technologies.solar.photovoltaic.calc_PV** *(locator, config, latitude, longitude, weather_data, datetime_local, building_name)*

This function first determines the surface area with sufficient solar radiation, and then calculates the optimal tilt angles of panels at each surface location. The panels are categorized into groups by their surface azimuths, tilt angles, and global irradiation. In the last, electricity generation from PV panels of each group is calculated.

**Parameters**
- **locator** *(cea.inputlocator.InputLocator)* – An InputLocator to locate input files
- **radiation_path** *(String)* – solar insulation data on all surfaces of each building (path
- **metadata_csv** *(csv)* – data of sensor points measuring solar insulation of each building
- **latitude** *(float)* – latitude of the case study location
- **longitude** *(float)* – longitude of the case study location
- **weather_path** *(epw)* – path to the weather data file of the case study location
- **building_name** *(Series)* – list of building names in the case study

**Returns**
- Building_PV.csv with PV generation potential of each building, Building_sensors.csv with sensor data of each PV panel.

**cea.technologies.solar.photovoltaic.calc_PV_power** *(absorbed_radiation_Wperm2, T_cell_C, eff_nom, tot_module_area_m2, Bref_perC, misc_losses)*

To calculate the power production of PV panels.

**Parameters**
- **absorbed_radiation_Wperm2** *(float)* – absorbed radiation [W/m2]
- **T_cell_C** – cell temperature [degree]
- **eff_nom** *(float)* – nominal efficiency of PV module [-]
• **tot_module_area_m2** *(float)* – total PV module area [m²]
• **Bref_perC** *(float)* – cell maximum power temperature coefficient [degree C⁻¹]
• **misc_losses** *(float)* – expected system loss [-]

**Return el_output_PV_kW**  
Power production [kW]

Rtype el_output_PV_kW  float


**kea.technologies.solar.photovoltaic.calc_absorbed_radiation_PV** *(I_sol, I_direct, I_diffuse, tilt, Sz, teta, tetaed, tetaeg, panel_properties_PV)*

**Parameters**

• **I_sol** *(float)* – total solar radiation [Wh/m²]
• **I_direct** *(float)* – direct solar radiation [Wh/m²]
• **I_diffuse** *(float)* – diffuse solar radiation [Wh/m²]
• **tilt** *(float)* – solar panel tilt angle [rad]
• **Sz** *(float)* – solar zenith angle [rad]
• **teta** *(float)* – angle of incidence [rad]
• **tetaed** *(float)* – effective incidence angle from diffuse radiation [rad]
• **tetaeg** *(float)* – effective incidence angle from ground-reflected radiation [rad]
• **panel_properties_PV** *(dataframe)* – properties of the PV panel

**Returns**


**kea.technologies.solar.photovoltaic.calc_angle_of_incidence** *(g, lat, ha, tilt, teta_z)*

To calculate angle of incidence from solar vector and surface normal vector. (Validated with Sandia pvlib.irrandiance.aoi)

**Parameters**

• **lat** *(float)* – latitude of the location of case study [radians]
• **g** *(float)* – declination of the solar position [radians]
• **ha** *(float)* – hour angle [radians]
• **tilt** *(float)* – panel surface tilt angle [radians]
• **teta_z** *(float)* – panel surface azimuth angle [radians]

**Return teta_B**  angle of incidence [radians]

Rtype teta_B  float
cea.technologies.solar.photovoltaic.calc_cell_temperature(absorbed_radiation_Wperm2, T_external_C, panel_properties_PV)

Calculates cell temperatures based on the absorbed radiation.

**Parameters**
- **absorbed_radiation_Wperm2 (np.array)** – absorbed radiation on panel
- **T_external_C (series)** – drybulb temperature from the weather file
- **panel_properties_PV (dataframe)** – panel property from the supply system database

**Return**
T_cell_C cell temperature of PV panels

**Rtype**
T_cell_C series

cea.technologies.solar.photovoltaic.calc_diffuseground_comp(tilt_radians)

To calculate reflected radiation and diffuse radiation.

- **param**
  - tilt_radians: surface tilt angle [rad]
- **type**
  - tilt_radians: float
  - return teta_ed: effective incidence angle from diffuse radiation [rad]
  - return teta_eg: effective incidence angle from ground-reflected radiation [rad]
  - rtype teta_ed: float
  - rtype teta_eg: float

**References**

cea.technologies.solar.photovoltaic.calc_optimal_angle(teta_z, latitude, transmissivity)

To calculate the optimal tilt angle of the solar panels.

**Parameters**
- **teta_z (float)** – surface azimuth, 0 degree south (east negative) or 0 degree north (east positive)
- **latitude (float)** – latitude of the case study site
- **transmissivity (float)** – clearness index [-]

**Return**
abs(b) optimal tilt angle [radians]

**Rtype**
abs(b) float


cea.technologies.solar.photovoltaic.calc_optimal_spacing(Sh, Az, tilt_angle, module_length)

To calculate the optimal spacing between each panel to avoid shading.

**Parameters**
- **Sh (float)** – Solar elevation at the worst hour [degree]
- **Az (float)** – Solar Azimuth [degree]
- **tilt_angle (float)** – optimal tilt angle for panels on flat surfaces [degree]
- **module_length (float)** – [m]

**Return**
D optimal distance in [m]

**Rtype**
D float

cea.technologies.solar.photovoltaic.calc_properties_PV_db(database_path, config)

To assign PV module properties according to panel types.
Parameters **type**

**type_PVpanel** *(string)* – type of PV panel used

Returns dict with Properties of the panel taken form the database

```python
ccea.technologies.solar.photovoltaic.calc_pv_generation(sensor_groups, weather_data, date_local, solar_properties, latitude, panel_properties_PV)
```

To calculate the electricity generated from PV panels.

Parameters

- **hourly_radiation** *(dataframe)* – mean hourly radiation of sensors in each group [Wh/m²]
- **number_groups** *(float)* – number of groups of sensor points
- **number_points** *(float)* – number of sensor points in each group
- **prop_observers** *(dataframe)* – mean values of sensor properties of each group of sensors
- **weather_data** *(dataframe)* – weather data read from the epw file
- **g** *(float)* – declination
- **Sz** *(float)* – zenith angle
- **Az** – solar azimuth
- **ha** – hour angle
- **latitude** – latitude of the case study location

Returns

```python
ccea.technologies.solar.photovoltaic.calc_surface_azimuth(xdir, ydir, B)
```

Calculate surface azimuth from the surface normal vector (x,y,z) and tilt angle (B). Following the geological sign convention, an azimuth of 0 and 360 degree represents north, 90 degree is east.

Parameters

- **xdir** *(float)* – surface normal vector x in (x,y,z) representing east-west direction
- **ydir** *(float)* – surface normal vector y in (x,y,z) representing north-south direction
- **B** *(float)* – surface tilt angle in degree

Returns **surface_azimuth** the azimuth of the surface of a solar panel in degree

Rtype **surface_azimuth** float

```python
ccea.technologies.solar.photovoltaic.main(config)
```

```python
ccea.technologies.solar.photovoltaic.optimal_angle_and_tilt(sensors_metadata_clean, latitude, worst_sh, worst_Az, transmissivity, Max_Isol, module_length)
```

This function first determines the optimal tilt angle, row spacing and surface azimuth of panels installed at each sensor point. Secondly, the installed PV module areas at each sensor point are calculated. Lastly, all the modules are categorized with its surface azimuth, tilt angle, and yearly radiation. The output will then be used to calculate the absorbed radiation.

Parameters
- `sensors_metadata_clean (dataframe)` – data of filtered sensor points measuring solar insulation of each building
- `latitude (float)` – latitude of the case study location
- `worst_sh (float)` – solar elevation at the worst hour [degree]
- `worst_Az (float)` – solar azimuth at the worst hour [degree]
- `transmissivity (float)` – transmissivity: clearness index [-]
- `module_length (float)` – length of the PV module [m]
- `Max_Isol (float)` – max radiation potential (equals to global horizontal radiation) [Wh/m2/year]

**Returns sensors_metadata_clean** data of filtered sensor points categorized with module tilt angle, array spacing, surface azimuth, installed PV module area of each sensor point and the categories

**Rtype sensors_metadata_clean** dataframe

**Assumptions**

1) Tilt angle: If the sensor is on tilted roof, the panel will have the same tilt as the roof. If the sensor is on a wall, the tilt angle is 90 degree. Tilt angles for flat roof is determined using the method from Quinn et al.

2) Row spacing: Determine the row spacing by minimizing the shadow according to the solar elevation and azimuth at the worst hour of the year. The worst hour is a global variable defined by users.

3) Surface azimuth (orientation) of panels: If the sensor is on a tilted roof, the orientation of the panel is the same as the roof. Sensors on flat roofs are all south facing.

**cea.technologies.solar.photovoltaic_thermal module**

Photovoltaic thermal panels

`cea.technologies.solar.photovoltaic_thermal.calc_Cinv_PVT (PVT_peak_W, locator, technology=0)`

P_peak in kW result in CHF technology = 0 represents the first technology when there are multiple technologies. FIXME: handle multiple technologies when cost calculations are done

`cea.technologies.solar.photovoltaic_thermal.calc_PVT (locator, config, latitude, longitude, weather_data, date_local, building_name)`

This function first determines the surface area with sufficient solar radiation, and then calculates the optimal tilt angles of panels at each surface location. The panels are categorized into groups by their surface azimuths, tilt angles, and global irradiation. In the last, electricity and heat generation from PVT panels of each group are calculated.

**Parameters**

- `locator (cea.inputlocator.InputLocator)` – An InputLocator to locate input files
- `radiation_json_path (string)` – path to solar insulation data on all surfaces of each building
- `metadata_csv_path (string)` – path to data of sensor points measuring solar insulation of each building
- `latitude (float)` – latitude of the case study location
• **longitude** (*float*) – longitude of the case study location

• **weather_path** (*epw*) – path to the weather data file of the case study location

• **building_name** (*Series*) – list of building names in the case study

• **T_in** – inlet temperature to the solar collectors [C]

**Returns** Building_PVT.csv with solar collectors heat generation potential of each building, Building_PVT_sensors.csv with sensor data of each PVT panel.

c.technologies.solar.photovoltaic_thermal.calc_PVT_generation(**sensor_groups**, **weather_data**, **date_local**, **solar_properties**, **latitude**, **tot_bui_height_m**, **panel_properties_SC**, **panel_properties_PV**, **config**)

To calculate the heat and electricity generated from PVT panels.

**Parameters**

- **sensor_groups** (*dict*) – properties of sensors in each group

- **weather_data** (*dataframe*) – weather data read from .epw

- **solar_properties** –

- **latitude** – latitude of the case study location

- **tot_bui_height_m** – total height of all buildings [m]

- **panel_properties_SC** – properties of solar thermal collectors

- **panel_properties_PV** – properties of photovoltaic panels

- **config** – user settings from c.cea.config

**Returns**


This function calculates the heat & electricity production from PVT collectors. The heat production calculation is adapted from calc_SC_module and then the updated cell temperature is used to calculate PV electricity production.

**Parameters**

- **tilt_angle_deg** – solar panel tilt angle [rad]

- **IAM_b_vector** – incident angle modifier for beam radiation [-]

- **I_direct_vector** – direct radiation [W/m2]
• **I_diffuse_vector** – diffuse radiation [W/m²]
• **Tamb_vector_C** – dry bulb temperature [°C]
• **IAM_d_vector** – incident angle modifier for diffuse radiation [-]
• **Leq** – equivalent length of pipes per aperture area [m/m² aperture]
• **Le** – equivalent length of collector pipes per aperture area [m/m² aperture]
• **absorbed_radiation_PV_Wperm2** – absorbed solar radiation of PV module [Wh/m²]
• **Tcell_PV_C** – PV cell temperature [°C]
• **module_area_per_group_m2** – PV module area [m²]

**Returns**

---


```python
cea.technologies.solar.photovoltaic_thermal.calc_pipe_equivalent_length(panel_properties_PV, panel_properties_SC, tot_bui_height_m, total_area_module_m2)
```

```python
cea.technologies.solar.photovoltaic_thermal.get_t_in_pvt(config)
```

```python
cea.technologies.solar.photovoltaic_thermal.main(config)
```

**cea.technologies.solar.solar_collector module**

class solar_collectors

```python
cea.technologies.solar.solar_collector.calc_pipe_equivalent_length(tot_bui_height_m, panel_prop, total_area_module)
```

To calculate the equivalent length of pipings in buildings:

```python
cea.technologies.solar.solar_collector.calc_Cinv_SC(Area_m2, locator, panel_type)
```

Lifetime 35 years

```python
cea.technologies.solar.solar_collector.calc_Eaux_SC(specific_flow_kgpers, dP_collector_Pa, Leq_mperm2, l_int_mperm2, Aa_m2)
```

Calculate auxiliary electricity for pumping heat transfer fluid through solar collectors to downstream equipment (absorption chiller, district heating network...). This include pressure losses from pipe friction, collector, and the building head.

```python
cea.technologies.solar.solar_collector.calc_Eaux_panels(specific_flow_kgpers, dP_collector_Pa, pipe_lengths, Aa_m2)
```

Calculate auxiliary electricity for pumping heat transfer fluid in solar collectors. This include pressure losses from pipe friction, collector.

---

9.1. cea package
loss per module [Pa] :param Leq_mperm2: total pipe length per aperture area [m] :param Aa_m2: aperture area [m2] :return:

```python
ccea.technologies.solar.solar_collector.calc_IAM_beam_SC(solar_properties, teta_z_deg, tilt_angle_deg, type_SCpanel, latitude_deg)
```


```python
ccea.technologies.solar.solar_collector.calc_SC(locator, config, latitude, longitude, weather_data, date_local, building_name)
```

This function first determines the surface area with sufficient solar radiation, and then calculates the optimal tilt angles of panels at each surface location. The panels are categorized into groups by their surface azimuths, tilt angles, and global irradiation. In the last, heat generation from SC panels of each group is calculated.


```python
ccea.technologies.solar.solar_collector.calc_SC_generation(sensor_groups, weather_data, date_local, solar_properties, tot_bui_height, panel_properties_SC, latitude_deg, config)
```


```python
ccea.technologies.solar.solar_collector.calc_SC_module(config, radiation_Wperm2, panel_properties, Tamb_vector_C, IAM_b, tilt_angle_deg, pipe_lengths)
```

This function calculates the heat production from a solar collector. The method is adapted from TRNSYS Type 832. Assume no no condensation gains, no wind or long-wave dependency, sky factor set to zero.

This function determines the optimal mass flow rate and the corresponding pressure drop that maximize the total heat production in every time-step. It is done by maximizing the energy generation function (balance equation) assuming the electricity requirement is twice as valuable as the thermal output of the solar collector.

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

This function takes the following parameters:
- `q1`: qout [kW] at zero flow rate
- `q2`: qout [kW] at nominal flow rate (mB0)
- `q3`: qout [kW] at maximum flow rate (mB_max)
- `q4`: qout [kW] at minimum flow rate (mB_min)
- `E1`: auxiliary electricity used at zero flow rate [kW]
- `E2`: auxiliary electricity used at nominal flow rate [kW]
- `E3`: auxiliary electricity used at max flow rate [kW]
- `E4`: auxiliary electricity used at min flow rate [kW]
- `m1`: zero flow rate [kg/hr/m2 aperture]
- `m2`: nominal flow rate (mB0) [kg/hr/m2 aperture]
- `m3`: maximum flow rate (mB_max) [kg/hr/m2 aperture]
- `m4`: minimum flow rate (mB_min) [kg/hr/m2 aperture]
- `dP1`: pressure drop [Pa/m2] at zero flow rate
- `dP2`: pressure drop [Pa/m2] at nominal flow rate
- `dP3`: pressure drop [Pa/m2] at maximum flow rate
- `dP4`: pressure drop [Pa/m2] at minimum flow rate
- `Area_a`: aperture area [m2]

The function returns:
- `mass_flow_opt`: optimal mass flow at each hour [kg/s]
- `dP_opt`: pressure drop at optimal mass flow at each hour [Pa]

Fonseca et al., 2016


```python
cea.technologies.solar.solar_collector.calc_optimal_mass_flow_2(m, q, dp)
```

Set mass flow and pressure drop to zero if the heat balance is negative.

```python
cea.technologies.solar.solar_collector.calc_properties_SC_db(database_path, config)
```

To assign SC module properties according to panel types.

```python
cea.technologies.solar.solar_collector.calc_qloss_network(Mfl, Le, Area_a, Tm, Te, maxmsc)
```

Calculate non-recoverable losses.

```python
cea.technologies.solar.solar_collector.get_t_in_sc(config)
```

```python
cea.technologies.solar.solar_collector.main(config)
```

```python
cea.technologies.solar.solar_collector.update_negative_total_supply(aperture_area_m2, auxiliary_electricity_kW, flow, mcp_kWperK, pipe_lengths, specific_flows_kgps, specific_pressure_losses_Pa, supply_losses_kW, supply_out_total_kW)
```

9.1. cea package
This function updates the hot water production when losses are too high. When supply losses are higher than supply out (supply_out_total < 0), the hot water is re-circulated back to panels instead of sending it to downstream equipment (DH or absorption chiller):

:param aperture_area_m2: aperture area per panel
:param auxiliary_electricity_kW: electricity required to pump hot water in the transmission pipelines
:param flow: index for the iteration number
:param mcp_kWperK: specific mcp_kWperK
:param pipe_lengths: lengths of transmission pipes
:param specific_flows_kgpers: specific mass flow of hot water in panels
:param specific_pressure_losses_Pa: specific pressure drop per panel
:param supply_losses_kW: heat loss through transmission pipelines
:param supply_out_total_kW: total heat supply
:return:

`cea.technologies.solar.solar_collector.vectorize_calc_Eaux_SC(specific_flows_kgpers, dP_collector_Pa, pipe_lengths, Aa_m2)`

**cea.technologies.thermal_network package**

**Submodules**

**cea.technologies.thermal_network.simplified_thermal_network module**

`cea.technologies.thermal_network.simplified_thermal_network.calc_head_loss_m(diameter_m, max_volume_flow_rate_m3s, coefficient_friction, length_m)`

`cea.technologies.thermal_network.simplified_thermal_network.calc_linear_thermal_loss_coefficient(diameter_ext_m, diameter_int_m, diameter_insulation_m)`

`cea.technologies.thermal_network.simplified_thermal_network.calc_max_diameter(volume_flow_m3s, pipe_catalog, velocity_ms, peak_load_percentage)`

`cea.technologies.thermal_network.simplified_thermal_network.calc_thermal_loss_per_pipe(T_in_K, m_kgpers, T_ground_K, k_kWperK)`

`cea.technologies.thermal_network.simplified_thermal_network.calculate_ground_temperature(locator)`

calculate ground temperatures.

**Parameters** locator –

**Returns** list of ground temperatures, one for each hour of the year

**Return type** list[np.float64]
Extracts network data into DataFrames for pipes and nodes in the network

**Parameters**

- `edge_shapefile_df (DataFrame)` – DataFrame containing all data imported from the edge shapefile
- `node_shapefile_df (DataFrame)` – DataFrame containing all data imported from the node shapefile

**Return**

- `node_df` DataFrame containing all nodes and their corresponding coordinates
- `edge_df` list of edges and their corresponding lengths and start and end nodes

This function reads the existing node and pipe network from a shapefile and produces an edge-node incidence matrix (as defined by Oppelt et al., 2016) as well as the edge properties (length, start node, and end node) and node coordinates.

**cea.technologies.thermal_network.simplified_thermal_network.main(config)**

**run the whole network summary routine**

This module implements the substation model.

**cea.technologies.thermal_network.substation_matrix.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. 

- `t_0`: last minimum temperature
- `t_1`: current minimum temperature
to evaluate

**Return**: new minimum temperature

**cea.technologies.thermal_network.substation_matrix.calc_DH_supply(t_0, t_1)**

This function calculates the heating temperature requirement of the building side according to the maximum temperature requirement at that time-step. 

- `t_0`: temperature requirement from one heating application
- `t_1`: temperature requirement from another heating application

**Return**: maximum temperature requirement

**cea.technologies.thermal_network.substation_matrix.calc_HEX_cooling(building, type, name, tci, UA, cc_old, delta_cap_mass_flow)**

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

```python
cea.technologies.thermal_network.substation_matrix.calc_HEX_heating(building, type, name, thi, UA, ch_old, delta_cap_mass_flow)
```

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

```python
cea.technologies.thermal_network.substation_matrix.calc_HEX_mix(heat, temperatures, mass_flows)
```

This function computes the average temperature between two vectors of heating demand. In this case, domestic hotwater and space heating.

**Parameters**

- \( heat \) – load heating
- \( temperatures \) – out temperature of heat exchanger for different heating modes
- \( mass_flows \) – mass flows for each heating mode

**Returns** \(tavg\): average out temperature.
This function calculates the area of a heat exchanger at nominal conditions.

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

**Returns** (area, UA): area: area of heat exchange, UA: coefficient representing the area of heat exchanger times the coefficient of transmittance of the heat exchanger

This function calculates the state of the heat exchanger at the substation of every customer with cooling needs. Cold/primary side: network; hot/secondary side: building.

- **Qnom**: nominal cooling load
- **thi_0**: inflow temperature of secondary/building side
- **tho_0**: outflow temperature of secondary/building side
- **tci_0**: inflow temperature of primary/network side
- **ch_0**: capacity mass flow rate on secondary/building side

**Returns** (Area_HEX_cooling, UA_cooling)

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 = logarithmic temperature difference

This function calculates the Area and UA of each substation heat exchanger. Primary side = network, Secondary side = Building.

- **cc_0**: nominal capacity mass flow rate primary side
- **Qnom**: nominal heating load
- **thi_0**: nominal inflow temperature of primary/network side
- **tci_0**: nominal inflow temperature of secondary/building side
- **tco_0**: nominal outflow temperature of secondary/building side

**Returns**
- **Area_HEX_heating**: Heat exchanger area in [m^2]
- **UA_heating**: UA
This function returns the heat exchanger specifications for given building demand, HEX type and supply temperature. primary side: network; secondary side: building:

- **param** building_demand: DataFrame with demand values
- **param** load_type: ‘cs_sys’ or ‘hs_sys’ for cooling or heating, ‘cdata_sys’, ‘cre_sys’
- **param** building_system: ‘aru’, ‘ahu’, ‘scu’
- **param** T_supply_C: Supply temperature
- **return**: HEX area and UA

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

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

This function calculates individual substation return temperature and required heat capacity (mcp) of the supply stream at each time step.

- **param** building: list of building informations
- **param** T_DC_supply_K: matrix of the substation supply temperatures in K
- **param** substation_HEX_specs: substation heat exchanger properties
- **param** thermal_network

**Return**
- **t_return_DC**: the substation return temperature
- **mcp_DC**: the required heat capacity (mcp) from the DC

This function calculates the efficiency of exchange for a tube-shell heat exchanger according to the NTU method of ASHRAE 90.1
calculate individual substation return temperature and required heat capacity (mcp) of the supply stream at each
time step.  

:param building: list of building informations 
:param T_DH_supply_K: matrix of the substation supply temperatures in K 
:param substation_HEX_specs: substation heat exchanger properties

Return t_return_DH  the substation return temperature

Return mcp_DH  the required heat capacity (mcp) from the DH

determines thermal network target temperatures (T_supply_DH_C,T_supply_DC) on the network side at each
substation.  

:param building_names: 
:param locator: 
:return:

determines thermal network target temperatures (T_supply_DH_C,T_supply_DC) on the network side at each
substation.  

:param building_names: 
:param locator: 
:return:

determines thermal network target temperatures (T_supply_DH_C,T_supply_DC) on the network side at each
substation.  

:param building_names: 
:param locator: 
:return:

determines thermal network target temperatures (T_supply_DH_C,T_supply_DC) on the network side at each
substation.  

:param building_names: 
:param locator: 
:return:
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 buildings_demands – Dictionary of DataFrames with all buildings_demands in the area

Returns (substations_HEX_specs, buildings_demands) - substations_HEX_specs: dataframe with substation heat exchanger specs at each building, buildings_demands: lists of heating demand/flowrate/supply temperature of all buildings connected to the network.

This function size the substation heat exchanger area and the UA values.

Parameters building_demand – dataframe with building demand properties

Returns A list of substation heat exchanger properties (Area & UA) for heating, cooling and DHW

Calculate all substation return temperature and required flow rate at each time-step.
Parameters

- **locator** – an InputLocator instance set to the scenario to work on
- **buildings_demands** – dictionary of building demands
- **substations_HEX_specs** – list of dataframes for substation heat exchanger Area and UA for heating, cooling and DHW
- **T_substation_supply** – supply temperature at each substation in [K]
- **t** – time-step
- **network_type** – a string that defines whether the network is a district heating (‘DH’) or cooling (‘DC’) network
- **use_same_temperature_for_all_nodes** – flag for calculating nominal flow rate, using one target temperature
- **thermal_network** (cea.technologies.thermal_network.thermal_network.ThermalNetwork) – container for all the thermal network data.

Returns

tcea.technologies.thermal_network.thermal_network module

Hydraulic - thermal network
class cea.technologies.thermal_network.thermal_network.HourlyThermalResults(T_supply_nodes, 
T_return_nodes, 
temperatures_at_plant_K, 
q_loss_supply_edges_kW, 
linear_thermal_loss_supply_edges_Wperm, 
thermal_losses_system_kW, 
plant_heat_requirement, 
pressure_at_supply_nodes_Pa, 
pressure_loss_system_Pa, 
pressure_loss_system_kW, 
pressure_loss_substations_kW, 
linear_pressure_loss_supply_Paperm, 
edge_mass_flows, 
node_mass_flows, 
velocities_in_supply_edges_mpers, 
pressure_loss_supply_edge_kW)

Bases: tuple

T_return_nodes
    Alias for field number 1

T_supply_nodes
    Alias for field number 0

__getnewargs__()
    Return self as a plain tuple. Used by copy and pickle.

__getstate__()
    Exclude the OrderedDict from pickling

static __new__(_cls, T_supply_nodes, T_return_nodes, temperatures_at_plant_K, 
q_loss_supply_edges_kW, linear_thermal_loss_supply_edges_Wperm, 
thermal_losses_system_kW, plant_heat_requirement, pressure_at_supply_nodes_Pa, 
pressure_loss_system_Pa, pressure_loss_system_kW, 
pressure_loss_substations_kW, linear_pressure_loss_supply_Paperm, 
edge_mass_flows, node_mass_flows, velocities_in_supply_edges_mpers, pressure_loss_supply_edge_kW)

Create new instance of HourlyThermalResults(T_supply_nodes, T_return_nodes, temperatures_at_plant_K, 
q_loss_supply_edges_kW, linear_thermal_loss_supply_edges_Wperm, 
thermal_losses_system_kW, plant_heat_requirement, pressure_at_supply_nodes_Pa, 
pressure_loss_system_Pa, pressure_loss_system_kW, pressure_loss_substations_kW, 
linear_pressure_loss_supply_Paperm, edge_mass_flows, node_mass_flows, velocities_in_supply_edges_mpers, pressure_loss_supply_edge_kW)


__repr__

Return a nicely formatted representation string

__slots__ = ()

_asdict()

Return a new OrderedDict which maps field names to their values

_fields = ('T_supply_nodes', 'T_return_nodes', 'temperatures_at_plant_K', 'q_loss_supply_edges_kW', ...

classmethod _make(iterable, new=<built-in method __new__ of type object>, len=<built-in function len>)

Make a new HourlyThermalResults object from a sequence or iterable

_replace(**kwds)

Return a new HourlyThermalResults object replacing specified fields with new values

downstream

Alias for field number 12

linear_pressure_loss_supply_Paperm

Alias for field number 11

linear_thermal_loss_supply_edges_Wperm

Alias for field number 4

node_mass_flows

Alias for field number 13

plant_heat_requirement

Alias for field number 6

pressure_at_supply_nodes_Pa

Alias for field number 7

pressure_loss_substations_kW

Alias for field number 10

pressure_loss_supply_edge_kW

Alias for field number 15

pressure_loss_system_Pa

Alias for field number 8

pressure_loss_system_kW

Alias for field number 9

q_loss_supply_edges_kW

Alias for field number 3

temperatures_at_plant_K

Alias for field number 2

thermal_losses_system_kW

Alias for field number 5

velocities_in_supply_edges_mpers

Alias for field number 14

class cea.technologies.thermal_network.thermal_network.ThermalNetwork(locator, network_name, thermal_network_section=None)
Bases: object

A thermal network instance contains information about the edges, nodes and buildings of a thermal network as produced by `get_thermal_network_from_csv()` or `get_thermal_network_from_shapefile()`.

Variables

- **edge_node_df** (DataFrame) – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values (n x e)

- **all_nodes_df** (DataFrame) – DataFrame that contains all nodes, whether a node is a consumer, plant, or neither, and, if it is a consumer or plant, the name of the corresponding building (2 x n)

- **edge_df** (DataFrame) –

__init__ (locator, network_name, thermal_network_section=None)

x.__init__(...) initializes x; see help(type(x)) for signature

clone ()

Create a copy of the thermal network. Assumes the fields have all been set.

copy_config_section (thermal_network_section)

get_thermal_network_from_csv ()

This function reads the existing node and pipe network from csv files (as provided for the Zug reference case) and produces an edge-node incidence matrix (as defined by Oppelt et al., 2016) as well as the length of each edge.

Parameters

- **locator** (InputLocator) – an InputLocator instance set to the scenario to work on

- **network_type** (str) – a string that defines whether the network is a district heating (’DH’) or cooling (’DC’) network

Return edge_node_df DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values (n x e)

Return all_nodes_df DataFrame that contains all nodes, whether a node is a consumer, plant, or neither, and, if it is a consumer or plant, the name of the corresponding building (2 x n)

Return pipe_data_df[‘LENGTH’] vector containing the length of each edge in the network (1 x e)

Rtype edge_node_df DataFrame

Rtype all_nodes_df DataFrame

Rtype pipe_data_df[‘LENGTH’] array

The following files are created by this script:

- **DH_EdgeNode**: csv file containing edge_node_df stored in locator.get_optimization_network_layout_folder()

- **DH_AllNodes**: csv file containing all_nodes_df stored in locator.get_optimization_network_layout_folder()
get_thermal_network_from_shapefile()
This function reads the existing node and pipe network from a shapefile and produces an edge-node incidence matrix (as defined by Oppelt et al., 2016) as well as the edge properties (length, start node, and end node) and node coordinates.

Parameters
- locator (InputLocator) – an InputLocator instance set to the scenario to work on
- network_type (str) – a string that defines whether the network is a district heating (‘DH’) or cooling (‘DC’) network

Return
- edge_node_df DataFrame consisting of \(n \times e\) rows (number of nodes) and \(e\) columns (number of edges) and indicating the direction of flow of each edge \(e\) at node \(n\): if \(e\) points to \(n\), value is \(1\); if \(e\) leaves node \(n\), \(-1\); else, \(0\). E.g. a plant will only have exiting flows, so only negative values (\(n \times e\))
- all_nodes_df DataFrame that contains all nodes, whether a node is a consumer, plant, or neither, and, if it is a consumer or plant, the name of the corresponding building (\(2 \times n\))
- edge_df[‘pipe length’] vector containing the length of each edge in the network (\(1 \times e\))

The following files are created by this script:
- DH_EdgeNode: csv file containing \(n \times e\) edges of all nodes in the network
- DH_Node_DF: csv file containing \(n \times 2\) nodes
- DH_Pipe_DF: csv file containing \(e \times 1\) pipe length

assign_pipes_to_edges(thermal_network)
This function assigns pipes from the catalog to the network for a network with unspecified pipe properties. Pipes are assigned based on each edge’s minimum and maximum required flow rate. Assuming max velocity for pipe DN450-550 is 3 m/s; for DN600 is 3.5 m/s. min velocity for all pipes are 0.3 m/s.

Parameters
- thermal_network (ThermalNetwork) – thermal network object
- mass_flow_df (DataFrame) – DataFrame containing the mass flow rate for each edge \(e\) at each time of the year \(t\)
- locator (InputLocator) – an InputLocator instance set to the scenario to work on

Return pipe_properties_df DataFrame containing the pipe properties for each edge in the network
This function calculates the aggregated heat conduction coefficients of all the pipes. Following the reference from [Wang et al., 2016]. The pipe material properties are referenced from [A. Kecabas et al., 2011], and the pipe catalogs are referenced from [J.A. Fonseca et al., 2016] and [isoplus].

Parameters

- **mass_flow** (*DataFrame*) – Vector with mass flows of each edge \((e \times 1)\)
- **locator** (*InputLocator*) – an InputLocator instance set to the scenario to work on
- **pipe_properties_df** (*DataFrame*) – DataFrame containing the pipe properties for each edge in the network
- **temperature__k** (*list*) – matrix containing the temperature of the water in each edge \(e\) at time \(t\) \((t \times e)\)
- **network_type** (*str*) – a string that defines whether the network is a district heating (‘DH’) or cooling (‘DC’) network
- **edge_df** (*DataFrame*) – list of edges and their corresponding lengths and start and end nodes

Return **k_all** DataFrame of aggregated heat conduction coefficients \((1 \times e)\) for all edges


**Calculates the Darcy friction factor** [Oppelt et al., 2016].

Parameters

- **pipe_diameter_m** (*ndarray*) – vector containing the pipe diameter in m for each edge \(e\) in the network \((e \times 1)\)
• **reynolds** (*ndarray*) – vector containing the reynolds number of flows in each edge in that timestep (e x 1)

• **roughness_m** (*pipe*) – float with pipe roughness

**Return**

**darcy**  calculated darcy friction factor for flow in each edge (ex1)

**Rtype**  ndarray


```python
cea.technologies.thermal_network.thermal_network.calc_edge_temperatures(temperature_node, edge_node)
```

Calculates the temperature at each edge assuming the average temperature in the edge is equal to the average of the temperatures at its start and end node as done, for example, by Wang et al. (2016), that is:

\[
T_{\text{edge}} = \frac{(T_{\text{node}_1} + T_{\text{node}_2})}{2}
\]

**Parameters**

• **temperature_node** – array containing the temperature in each node n (1 x n)

• **edge_node** – matrix consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. (n x e)

**Return**

**temperature_edge**  array containing the temperature in each edge e (1 x n)


```python
cea.technologies.thermal_network.thermal_network.calc_kinematic_viscosity(temperature)
```

Calculates the kinematic viscosity of water as a function of temperature based on a simple fit from data from the engineering toolbox.

**Parameters**

• **temperature** – in K

**Returns**  kinematic viscosity in m²/s

```python
cea.technologies.thermal_network.thermal_network.calc_mass_flow_edges(edge_node_df, mass_flow_substation_df, all_nodes_df, pipe_diameter_m, pipe_length_m, T_edge_K)
```

This function carries out the steady-state mass flow rate calculation for a predefined network with predefined mass flow rates at each substation based on the method from Todini et al. (1987), Ikonen et al. (2016), Oppelt et al. (2016), etc.

**Parameters**

• **all_nodes_df** (*DataFrame(t x n]*) – DataFrame containing all nodes and whether a node n is a consumer or plant node (and if so, which building that node corresponds to), or neither.

• **edge_node_df** (*DataFrame*) – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values (n x e)
• **mass_flow_substation_df** *(DataFrame)* – DataFrame containing the mass flow rate at each node \( n \) at each time of the year \( t \)

• **pipe_diameter_m** *(ndarray)* – vector containing the pipe diameter in m for each edge \( e \) in the network \((e \times 1)\)

• **pipe_length_m** *(ndarray)* – vector containing the length in m of each edge \( e \) in the network \((e \times 1)\)

• **\( T_{\text{edge}} \)** *(ndarray)* – matrix containing the temperature of the water in each edge \( e \) at time \( t \) \((t \times e)\)

Return **mass_flow_edge** matrix specifying the mass flow rate at each edge \( e \) at the given time step \( t \)

Rtype **mass_flow_edge** numpy.ndarray

```python
cea.technologies.thermal_network.thermal_network.calc_max_diameter(volume_flow_m3s, pipe_catalog, velocity_ms)
```

```python
cea.technologies.thermal_network.thermal_network.calc_max_edge_flowrate(thermal_network, processes=1)
```

Calculates the maximum flow rate in the network in order to assign the pipe diameter required at each edge. This is done by calculating the mass flow rate required at each substation to supply the calculated demand at the target supply temperature for each time step, finding the maximum for each node throughout the year and calculating the resulting necessary mass flow rate at each edge to satisfy this demand.

Parameters

• **thermal_network** *(ThermalNetwork)* – contains information about the thermal network

• **all_nodes_df** *(DataFrame)* – DataFrame containing all nodes and whether a node \( n \) is a consumer or plant node (and if so, which building that node corresponds to), or neither. \((2 \times n)\)

• **buildings_demands** – demand of each building in the scenario

• **edge_node_df** – DataFrame consisting of \( n \) rows (number of nodes) and \( e \) columns (number of edges) and indicating the direction of flow of each edge \( e \) at node \( n \): if \( e \) points to \( n \), value is 1; if \( e \) leaves node \( n \), -1; else, 0. E.g. a plant will only have exiting flows, so only negative values \((n \times e)\)

• **locator** *(InputLocator)* – an InputLocator instance set to the scenario to work on

• **substations_hex_specs** *(DataFrame)* – DataFrame with substation heat exchanger specs at each building.

• **t_target_supply_C** – target supply temperature at each substation

• **network_type** *(str)* – a string that defines whether the network is a district heating (‘DH’) or cooling (‘DC’) network

• **pipe_length** *(array)* – vector containing the length of each edge in the network

Return **edge_mass_flow_df** mass flow rate at each edge throughout the year

Return **max_edge_mass_flow_df** maximum mass flow at each edge to be used for pipe sizing

Rtype **edge_mass_flow_df** DataFrame

Rtype **max_edge_mass_flow_df** DataFrame
Calculates the nusselt number of the internal flow inside the pipes.

**Parameters**

- **pipe_diameter_m** (ndarray) – vector containing the pipe diameter in m for each edge e in the network (e x 1)
- **mass_flow_rate_kgs** (ndarray) – matrix containing the mass flow rate in each edge e at time t (t x e)
- **temperature_K** (list) – matrix containing the temperature of the water in each edge e at time t (t x e)
- **network_type** (str) – a string that defines whether the network is a district heating ('DH') or cooling ('DC') network

**Return** nusselt calculated nusselt number for flow in each edge (ex1)

**Rtype** nusselt ndarray

---

calculate plant heat requirements according to plant supply/return temperatures and flow rate:

- **plant_node**: list of plant nodes
- **t_supply_nodes**: node temperatures on the supply network
- **t_return_nodes**: node temperatures on the return network
- **mass_flow_substations_nodes_df**: substation mass flows

**Parameters**

- **plant_node**: ndarray
- **t_supply_nodes**: ndarray
- **t_return_nodes**: ndarray
- **mass_flow_substations_nodes_df**: pandas dataframe

**Return**

---

calculates the prandtl number of the internal flow inside the pipes.

**Parameters**

- **temperature__k** (list) – matrix containing the temperature of the water in each edge e at time t (t x e)

**Return**

---

Calculates the pressure losses throughout a pipe based on the Darcy-Weisbach equation and the Swamee-Jain solution for the Darcy friction factor [Oppelt et al., 2016].

**Parameters**

- **pipe_diameter_m** (ndarray) – vector containing the pipe diameter in m for each edge e in the network (e x 1)
- **pipe_length_m** (ndarray) – vector containing the length in m of each edge e in the network (e x 1)
- **mass_flow_rate_kgs** (ndarray) – matrix containing the mass flow rate in each edge e at time t (t x e)
- **t_edge__k** (list) – matrix containing the temperature of the water in each edge e at time t (t x e)
• **loop_type** *(binary)* – int indicating if function is called from loop calculation or not, or is derivate is necessary (1 = derivative of Loop, 2 = branch)

**Return** *pressure_loss_edge*  pressure loss through each edge e at each time t (t x e)

**Rtype** *pressure_loss_edge*  ndarray


```python
cea.technologies.thermal_network.thermal_network.calc_pressure_loss_substations(thermal_network, supply_temperature, t)
```

This function calculates the pressure losses in substations assuming each substation to be modeled by a valve and HEX for each supplied heating or cooling load.


```python
cea.technologies.thermal_network.thermal_network.calc_pressure_loss_system(pressure_loss_pipe_supply, pressure_loss_pipe_return, pressure_loss_substation)
```

```python
cea.technologies.thermal_network.thermal_network.calc_pressure_nodes(t_supply_node__k, t_return_node__k, thermal_network, t)
```

Calculates the pressure at each node based on Eq. 1 in Todini & Pilati (1987). For the pressure drop through a pipe, the Darcy-Weisbach equation was used as in Oppelt et al. (2016) instead of the Hazen-Williams method used by Todini & Pilati. Since the pressure is calculated after the mass flow rate (rather than concurrently) this is only a first step towards implementing the Gradient Method from Todini & Pilati used by EPANET et al.

**Parameters**

• **edge_node_df** *(DataFrame)* – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values (n x e)

• **pipe_diameter** *(ndarray)* – vector containing the pipe diameter in m for each edge e in the network (e x 1)

• **pipe_length** *(ndarray)* – vector containing the length in m of each edge e in the network (e x 1)

• **edge_mass_flow** *(ndarray)* – matrix containing the mass flow rate in each edge e at time t (1 x e)

• **t_supply_node__k** *(list)* – array containing the temperature in each supply node n (1 x n)

• **t_return_node__k** *(list)* – array containing the temperature in each return node n (1 x n)

**Return** *pressure_loss_nodes_supply*  array containing the pressure loss at each supply node (1 x n)
Return `pressure_loss_nodes_return` array containing the pressure loss at each return node (1 x n)

Return `pressure_loss_system` pressure loss over the entire network

Rtype `pressure_loss_nodes_supply` ndarray

Rtype `pressure_loss_nodes_return` ndarray

Rtype `pressure_loss_system` float

`cea.technologies.thermal_network.thermal_network.calc_return_node_temperature(index, m_d, t_e_out, t_return, z_pipe_out, m_sub)`

The function calculates the node temperature with merging flows from pipes in the return line.

Parameters

- **index** (float `T_return_all_2`) – node index
- **m_d** (DataFrame) – pipe mass flow matrix (exe)
- **t_e_out** (DataFrame) – pipe outlet temperatures in edge node matrix (nxe)
- **t_return** (list) – list of substation return temperatures
- **z_pipe_out** (DataFrame) – pipe outlet matrix (nxe)
- **m_sub** (DataFrame) – DataFrame substation flow rate

Returns **t_node** node temperature with merging flows in the return line

Rtype `t_node` float

`cea.technologies.thermal_network.thermal_network.calc_return_temperatures(t_ground, edge_node_df, mass_flow_df, mass_flow_substation_df, k, t_return, thermal_network)`

This function calculates the node temperatures considering heat losses in the return line. Starting from the substations at the end branches, the function goes through the edge-node index to search for the outlet node, and calculates the outlet node temperature after heat loss. Starting from that outlet node, the function calculates the node temperature at the corresponding pipe outlet, and the calculation goes on until all the node temperatures are solved. At nodes connecting to multiple pipes, the mixing temperature is calculated.

Parameters

- **t_ground** – vector with ground temperatures in K
- **edge_node_df** – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values
- **mass_flow_df** – DataFrame containing the mass flow rate for each edge e at each t
- **mass_flow_substation_df** – DataFrame containing the mass flow rate for each substation at each t
- **k** – aggregated heat conduction coefficient for each pipe
• t_return – return temperatures at the substations

```
Return t_node.T  list of node temperatures (nx1)
Rtype t_node.T  list
```

**cea.technologies.thermal_network.thermal_network.calc_reynolds**(mass_flow_rate_kgs, temperature__k, pipe_diameter_m)

Calculates the reynolds number of the internal flow inside the pipes.

**Parameters**

- **pipe_diameter_m** *(ndarray)* – vector containing the pipe diameter in m for each edge e in the network (e x 1)
- **mass_flow_rate_kgs** *(ndarray)* – matrix containing the mass flow rate in each edge e at time t (t x e)
- **temperature__k** *(list)* – matrix containing the temperature of the water in each edge e at time t (t x e)

**cea.technologies.thermal_network.thermal_network.calc_supply_temperatures**(t, edge_node_df, mass_flow_df, k, thermal_network)

This function calculate the node temperatures considering heat losses in the supply network. Starting from the plant supply node, the function go through the edge-node index to search for the outlet node, and calculate the outlet node temperature after heat loss. And starting from the outlet node, the function calculates the node temperature at the corresponding pipe outlet, and the calculation goes on until all the node temperatures are solved. At nodes connecting to multiple pipes, the mixing temperature is calculated.

**Parameters**

- **edge_node_df** *(DataFrame)* – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values (n x e)
- **mass_flow_df** *(DataFrame)* – DataFrame containing the mass flow rate for each edge e at each time of the year t (1 x e)
- **k** – aggregated heat conduction coefficient for each pipe (1 x e)

```
Return t_node.T  list of node temperatures (nx1)
Return plant_node the index of the plant node
Rtype t_node.T  list
Rtype plant_node  numpy array
```

**cea.technologies.thermal_network.thermal_network.calc_t_out**(node, edge, k_old, m_d, z, t_e_in, t_e_out, t_ground, z_note, thermal_network)

Given the pipe inlet temperature, this function calculate the outlet temperature of the pipe. Following the reference of [Wang et al., 2016]_.

**Parameters**

```
• **node** (*float*) – node index
• **edge** (*np array*) – edge indices
• **k** (*[kW/K]*) – DataFrame of aggregated heat conduction coefficient for each pipe (exe)
• **m_d** (*DataFrame*) – DataFrame of pipe flow rate (exe)
• **z** (*DataFrame*) – DataFrame of edge_node_matrix (nxe)
• **t_e_in** (*DataFrame*) – DataFrame of pipe inlet temperatures [K] in edge_node_matrix (nxe)
• **t_e_out** (*DataFrame*) – DataFrame of pipe outlet temperatures [K] in edge_node_matrix (nxe)
• **t_ground** (*list*) – vector with ground temperatures in [K]
• **z_note** (*DataFrame*) – DataFrame of the matrix to store information of solved nodes

:returns The calculated pipe outlet temperatures are directly written to T_e_out.

district heating systems and model parameters calibration. Energy Conversion and Management, 120, 294-305.

```python
cea.technologies.thermal_network.thermal_network.calc_thermal_conductivity(temperature)
```
Calculates the thermal conductivity of water as a function of temperature based on a fit proposed in:

Parameters **temperature** – in K

Returns thermal conductivity in W/(m*K)

```python
cea.technologies.thermal_network.thermal_network.calc_thermal_loss_system(thermal_loss_pipe_supply, thermal_loss_pipe_return)
```
calculate ground temperatures.

Parameters **locator** –

Returns list of ground temperatures, one for each hour of the year

Return type list[np.float64]

```python
cea.technologies.thermal_network.thermal_network.calculate_outflow_temp(z, z_note, m_d, t_e_out, z_pipe_out, t_node, t_e_in, t_ground_k, not_stuck, k, thermal_network)
```
calculates outflow temperature of nodes based on incoming mass flows and temperatures.

Parameters

• **z** (*dataframe (n x e)) – copy of edge-node matrix (n x e)
• **z_note** (*dataframe (n x e)) – copy of z matrix (n x e)
The function changes the flow directions in edge_node_df to align with flow directions at each time-step, this way all the mass flows are positive.

Parameters

- **edge_mass_flow** – Current mass flows on each edge
- **edge_node_df** – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values

Return edge_mass_flow
Return `edge_node_df` Updated `edge_node_df` matrix set to match positive flow directions of `edge_mass_flows`

```python
c ea.technologies.thermal_network.thermal_network.edge_mass_flow_iteration(thermal_network, edge_mass_flow_df, iteration_counter, t)
```

**Parameters**

- `network_type` – string with network type, DH or DC
- `edge_mass_flow_df` – edge mass flows (1 x e)
- `iteration_counter` – iteration counter
- `cc_value_sh` – capacity mass flow for space heating (1 x e)
- `ch_value` – capacity mass flow for cooling (1 x e)
- `cc_value_dhw` – capacity mass flow for warm water (1 x e)

**Returns**

```python
c ea.technologies.thermal_network.thermal_network.extract_network_from_shapefile(edge_shapefile_df, node_shapefile_df)
```

Extracts network data into DataFrames for pipes and nodes in the network

**Parameters**

- `edge_shapefile_df` (*DataFrame*) – DataFrame containing all data imported from the edge shapefile
- `node_shapefile_df` (*DataFrame*) – DataFrame containing all data imported from the node shapefile

**Return**

- `node_df` DataFrame containing all nodes and their corresponding coordinates
- `edge_df` list of edges and their corresponding lengths and start and end nodes

**Return type**

- `node_df` *DataFrame*
- `edge_df` *DataFrame*

```python
c ea.technologies.thermal_network.thermal_network.extrapolate_datapoints_for_representative_weeks(representative_week_data)
```

```python
c ea.technologies.thermal_network.thermal_network.find_loops(edge_node_df)
```

This function converts the input matrix into a `networkx` type graph and identifies all fundamental loops of the network. The group of fundamental loops is defined as the series of linear independent loops which can be combined to form all other loops.

**Parameters**

- `edge_node_df` (*DataFrame*) – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values (n x e)

**Returns**

- `loops` list of all fundamental loops in the network
- `graph` `networkx` dictionary type graph of network

**Return type**

- `loops` *list*
- `graph` *dictionary*
This function calculates the edge mass flows and node mass flows of each hour of the year.

Parameters

- **thermal_network** (ThermalNetwork) – object holding all the information about the thermal network
- **t** – timestep
- **t_target_supply_C** – target temperature of nodes
- **network_type** – ‘DH’ or ‘DC’
- **locator** – InputLocator
- **buildings_demands** – DataFrame of Building demands
- **substations_hex_specs** – DataFrame with substation heat exchanger specs at each building.
- **all_nodes_df** – DataFrame containing all nodes and whether a node n is a consumer or plant node (and if so, which building that node corresponds to), or neither. (2 x n)
- **edge_node_df** – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values (n x e)
- **edge_mass_flow_df** – Storage for edge mass flows of all hours of the year
- **diameter_guess** – Pipe diameter values
- **pipe_length** – Length of each edge
- **node_mass_flow_df** – Storage for node mass flows of all hours of the year

Return **edge_mass_flow_df** Storage for edge mass flows of all hours of the year

Return **node_mass_flow_df** Storage for node mass flows of all hours of the year
to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values.

- **all_nodes_df** – list of plant nodes and consumer nodes and their corresponding building names
- **edge_mass_flow_df_kgs** – Mass flow over every edge
- **t_target_supply_df** – Target supply temperature of each node
- **buildings_demands** – DataFrame of building demands
- **substations_HEX_specs** – DataFrame with substation heat exchanger specs at each building
- **edge_df** – list of edges and their corresponding lengths and start and end nodes
- **pipe_properties_df** – DataFrame containing the pipe properties for each edge in the network
- **csv_outputs** – Dictionary collecting all variables which are stored for all 8760 timesteps and later written to csv files

**Return csv_outputs** DataFrame with calculated values

**Return edge_mass_flow_df_kgs** updated edge mass flows

```python
ccea.technologies.thermal_network.thermal_network.initial_diameter_guess(thermal_network)
```

This function calculates an initial guess for the pipe diameter in looped networks based on the time steps with the 50 highest demands of the year. These pipe diameters are iterated until they converge, and this result is passed as an initial guess for the iteration over all time steps in an attempt to reduce total runtime.

**Parameters**

- **thermal_network** (ThermalNetwork) – object containing all the data of the thermal network.
- **all_nodes_df** (DataFrame) – DataFrame containing all nodes and whether a node n is a consumer or plant node (and if so, which building that node corresponds to), or neither. (2 x n)
- **buildings_demands** (dict) – demand of each building in the scenario
- **edge_node_df** (DataFrame) – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values
- **locator** (InputLocator) – an InputLocator instance set to the scenario to work on
- **substations_HEX_specs** (DataFrame) – DataFrame with substation heat exchanger specs at each building.
- **t_target_supply** (list) – target supply temperature at each substation
- **network_type** (str) – a string that defines whether the network is a district heating (‘DH’) or cooling (‘DC’) network
- **network_name** (str) – string with name of network
- **edge_df** (DataFrame) – list of edges and their corresponding lengths and start and end nodes
- **set_diameter** (bool) – boolean if diameter needs to be set
Return pipe_properties_df[['D_int_m': 'D_int_m']].values 
initial guess pipe diameters for all edges

Rtype pipe_properties_df[['D_int_m': 'D_int_m']].values array

cea.technologies.thermal_network.thermal_network.load_max_edge_flowrate_from_previous_run(thermal_network)
Bypass the calculation of calc_max_edge_flowrate and use the results form the previous run

cea.technologies.thermal_network.thermal_network.load_node_flowrate_from_previous_run(thermal_network)
Bypass the calculation of calc_max_edge_flowrate and use the results form the previous run

cea.technologies.thermal_network.thermal_network.main(config)
run the whole network summary routine

cea.technologies.thermal_network.thermal_network.output_hex_specs_at_nodes(substation_HEX_Q, thermal_network)

cea.technologies.thermal_network.thermal_network.prepare_inputs_of_representative_weeks(thermal_network)

cea.technologies.thermal_network.thermal_network.read_in_diameters_from_shapefile(thermal_network)

cea.technologies.thermal_network.thermal_network.read_properties_from_buildings(buildings_demands, property)
The function reads certain property from each building and output as a DataFrame.

Parameters

• buildings_demands – demand of each building in the scenario

• property – certain property from the building demand file. e.g. T_supply_target

Return property_df DataFrame of the particular property at each building.

Rtype property_df DataFrame

cea.technologies.thermal_network.thermal_network.reset_min_mass_flow_variables(thermal_network, t)
This function resets the parameters used for data storage for the minimum mass flow iteration :param thermal_network: :return:

cea.technologies.thermal_network.thermal_network.save_all_results_to_csv(csv_outputs, thermal_network)

cea.technologies.thermal_network.thermal_network.solve_network_temperatures(thermal_network, t)
This function calculates the node temperatures at time-step t accounting for heat losses throughout the network. There is one iteration to determine weather the substation supply temperature and the substation mass flow are cohesive. It is done as follow: The substation supply temperatures (T_substation_supply) are calculated based on the nominal edge flow rate (see calc_max_edge_flowrate), and then the substation mass flow requirements (mass_flow_substation_nodes_df) and pipe mass flows (edge_mass_flow_df_2) are updated accordingly. Following, the substation supply temperatures(T_substation_supply_2) are recalculated with the updated pipe mass flow.

The iteration continues until the substation supply temperatures converged.
Lastly, the plant heat requirements are calculated base on the plant supply/return temperatures and flow rates.

Parameters

• locator (InputLocator) – an InputLocator instance set to the scenario to work on

• t_ground – vector with ground temperatures in K

9.1. cea package 273
- `edge_node_df (DataFrame)` – DataFrame consisting of n rows (number of nodes) and e columns (number of edges) and indicating the direction of flow of each edge e at node n: if e points to n, value is 1; if e leaves node n, -1; else, 0. E.g. a plant will only have exiting flows, so only negative values (n x e)
- `all_nodes_df (DataFrame)` – DataFrame containing all nodes and whether a node n is a consumer or plant node (and if so, which building that node corresponds to), or neither. (2 x n)
- `edge_mass_flow_df (DataFrame)` – mass flow rate at each edge throughout the year
- `t_target_supply_df (DataFrame)` – target supply temperature at each substation
- `buildings_demands` – demand of each building in the scenario
- `substations_hex_specs (DataFrame)` – DataFrame with substation heat exchanger specs at each building.
- `t` – current time step
- `network_type (str)` – a string that defines whether the network is a district heating (‘DH’) or cooling (‘DC’) network
- `edge_df (DataFrame)` – list of edges and their corresponding lengths and start and end nodes
- `pipe_properties_df (DataFrame)` – DataFrame containing the pipe properties for each edge in the network
- `thermal_network (ThermalNetwork)` – A container for all the thermal network data

Returns
- `T_supply_nodes` list of supply line node temperatures (nx1)
- `T_return_nodes` list of return line node temperatures (nx1)
- `plant_heat_requirement` list of plant heat requirement

This function performs thermal and hydraulic calculation of a “well-defined” network, namely, the plant/consumer substations, piping routes and the pipe properties (length/diameter/heat transfer coefficient) are already specified.

The hydraulic calculation is based on Oppelt, T., et al., 2016 for the case with no loops. Firstly, the consumer substation heat exchanger designs are calculated according to the consumer demands at each substation. Secondly, the piping network is imported as a node-edge matrix (NxE), which indicates the connections of all nodes and edges and the direction of flow between them following graph theory. Nodes represent points in the network, which could be the consumers, plants or joint points. Edges represent the pipes in the network. For example, (n1,e1) = 1 denotes the flow enters edge “e1” at node “n1”, while when (n2,e2) = -1 denotes the flow leaves edge “e2” at node “n2”. Following, a steady-state hydraulic calculation is carried out at each time-step to solve for the edge mass flow rates according to mass conservation equations. With the maximum mass flow calculated from each edge, the property of each pipe is assigned.
Thirdly, the hydraulic thermal calculation for each time-steps over a year is based on a heat balance for each edge (heat at the pipe inlet equals heat at the outlet minus heat losses through the pipe). Finally, the pressure loss calculation is carried out based on Todini et al. (1987)

**Parameters**

- `temperature_control` – the control strategy of supply temperatures at plants
- `locator` (InputLocator) – an InputLocator instance set to the scenario to work on
- `network_type` (str) – a string that defines whether the network is a district heating (‘DH’) or cooling (‘DC’) network
- `file_type` (str) – string that defines the type of source file for the network to be imported (‘csv’ or shapefile ‘shp’)

The following files are created by this script, depending on the network type defined in the inputs:

- DH_EdgeNode or DC_EdgeNode: .csv, edge-node matrix for the defined network
- DH_AllNodes or DC_AllNodes: .csv, list of plant nodes and consumer nodes and their corresponding building names
- DH_MassFlow or DC_MassFlow: .csv, mass flow rates at each edge for each time step
- DH_T_Supply or DC_T_Supply: .csv, describes the supply temperatures at each node at each type step
- DH_T_Return or DC_T_Return: .csv, describes the return temperatures at each node at each type step
- DH_Plant_heat_requirement or DC_Plant_heat_requirement: .csv, heat requirement from the plants in a district heating or cooling network
- DH_P_Supply or DC_P_Supply: .csv, supply side pressure for each node in a district heating or cooling network at each time step
- DH_P_Return or DC_P_Return: .csv, return side pressure for each node in a district heating or cooling network at each time step
- DH_P_DeltaP or DC_P_DeltaP.csv, pressure drop over an entire district heating or cooling network at each time step

This function writes node values to the corresponding building substations.

**Parameters**

- `t_supply_nodes` (DataFrame) – DataFrame of supply line node temperatures (nx1)
- `all_nodes_df` (DataFrame) – DataFrame that contains all nodes, whether a node n is a consumer, plant, or neither, and, if it is a consumer or plant, the name of the corresponding building (2 x n)

**Return** T_substation_supply dataframe with node values matched to building substations

**Rtype** T_substation_supply DataFrame

The function writes values (temperatures or mass flows) from each substations to the corresponding nodes in the edge node matrix.

**Parameters**

- `all_nodes_df` – DataFrame containing all nodes and whether a node n is a consumer or plant node (and if so, which building that node corresponds to), or neither. (2 x n)
• **df_value** – DataFrame of value of each substation

• **flag** – flag == True if the values are temperatures ; flag == False if the value is mass flow

Return nodes_df DataFrame with values at each node (1xn)

Rtype nodes_df DataFrame

cEA.technologies.thermal_network.thermal_network.write_substation_values_to_nodes_df(all_node_df, df_value)

The function writes values (temperatures or mass flows) from each substations to the corresponding nodes in the edge node matrix.

Parameters

• **all_nodes_df** – DataFrame containing all nodes and whether a node n is a consumer or plant node (and if so, which building that node corresponds to), or neither. (2 x n)

• **df_value** – DataFrame of value of each substation

• **flag** – flag == True if the values are temperatures ; flag == False if the value is mass flow

Return nodes_df DataFrame with values at each node (1xn)

Rtype nodes_df DataFrame

cEA.technologies.thermal_network.thermal_network_costs module

class cEA.technologies.thermal_network.thermal_network_costs.Thermal_Network(locator, config, network_type)

Bases: object

Storage of information for the network currently being calculated.

__init__(locator, config, network_type)

x.__init__(...) initializes x; see help(type(x)) for signature

weather_data

cEA.technologies.thermal_network.thermal_network_costs.calc_CT_load_from_chiller_load(COP_chiller, chiller_load_kW)

calculates loads of cooling towers (CT) according to chiller loads

: param COP_chiller: float

: param chiller_load_kW: float

: return: Q_CT_kW, float

cEA.technologies.thermal_network.thermal_network_costs.calc_Capex_a_network_pipes(network_info)

Calculates network piping costs

cEA.technologies.thermal_network.thermal_network_costs.calc_Ctot_cooling_plants(network_info)

Calculates costs of centralized cooling plants (chillers and cooling towers).

Parameters network_info (NetworkInfo) – an object storing information of the current network

Returns

cEA.technologies.thermal_network.thermal_network_costs.calc_Ctot_cs_disconnected_buildings(network_info)

Calculates the space cooling cost of disconnected buildings. The calculation for partially disconnected buildings is done in calc_Ctot_cs_disconnected_loads.

: param network_info: an object storing information of the current network

: return:
cea.technologies.thermal_network.thermal_network_costs.calc_Ctot_cs_disconnected_loads

Calculates the space cooling cost of disconnected loads at the building level. The calculation for entirely disconnected buildings is done in calc_Ctot_cs_disconnected_buildings. :param Thermal_Network network_info: an object storing information of the current network :return:

cea.technologies.thermal_network.thermal_network_costs.calc_Ctot_cs_district

Calculates the total costs for cooling of the entire district, which includes the cooling networks and disconnected loads & buildings. Maintenance of network neglected, see Documentation Master Thesis Lennart Rogenhofer :param Thermal_Network network_info: an object storing information of the current network :return:

cea.technologies.thermal_network.thermal_network_costs.calc_Ctot_network_pump

Computes the total pump investment and operational cost, slightly adapted version of original in optimization main script. :type network_info: class storing network information :returns Capex_a: annualized capital cost :returns Opex_a_fixed: annual fixed operation and maintenance cost :returns Opex_var: annual variable operation cost

cea.technologies.thermal_network.thermal_network_costs.calc_network_size

Reads in the total network length and average pipe diameter :param network_info: Object storing network information. :return:

cea.technologies.thermal_network.thermal_network_costs.find_cooling_systems_string

Returns string of cooling load column names to read in demand at building for disconnected supply :param disconnected_systems: a list containing different cooling systems (ahu, aru, scu...) :return:

cea.technologies.thermal_network.thermal_network_costs.find_supplied_systems_annual

This function iterates through all buildings to find out from which loads we have a demand, and return the non zero loads. :param network_info: an object storing information of the current network :return:

cea.technologies.thermal_network.thermal_network_costs.find_supplied_systems_t

This function iterates through all buildings to find out from which loads we have a demand, and return the non zero loads. :param network_info: an object storing information of the current network :param t: hour we are looking at :return:

cea.technologies.thermal_network.thermal_network_costs.main

This function calculates the total costs of a network after running simulation from thermal_network_matrix. :param config: :return:

cea.technologies.thermal_network.thermal_network_loss module

Hydraulic - thermal network

cea.technologies.thermal_network.thermal_network_loss.calc_temperature_out_per_pipe

Parameters

- t_in – in Kelvin
- m – in kg/s
- k – in kW/K
- \( t_{\text{ground}} \) – in Kelvin

**Returns**

**cea.technologies.thermal_network.thermal_network_optimization module**

hydraulic network

**class** `cea.technologies.thermal_network.thermal_network_optimization.NetworkInfo(locator, config)`

Bases: `object`

Storage of information for the network currently being calculated.

```
__init__(locator, config)
```

**locate_individual_results**(individual)

**weather_data**

`cea.technologies.thermal_network.thermal_network_optimization.add_one_random_plant(individual_, network_info)`

**add_one_random_plant**(individual_buildings, network_info)

This function returns a random index within the individual at which a plant is permissible. :param NetworkInfo location: Object storing network information. :return: permissible index of plant within an individual

`cea.technologies.thermal_network.thermal_network_optimization.admissible_plant_location(network_info)`

This function returns a random index within the individual at which a plant is permissible. :param NetworkInfo location: Object storing network information. :return: permissible index of plant within an individual

**breed_new_generation**(selected_individuals, network_info)

Breeds new generation for genetic algorithm. Here we don’t assure that each parent is chosen at least once, but the expected value is that each parent should be chosen twice. E.g. we have N individuals. The chance of being parent 1 is 1/N, the chance of being parent 2 is (1-1/N)*1/(N-1). So the probability of being one of the two parents of any child is 1/N + (1-1/N)*1/(N-1) = 2/N. Since there are N children, the expected value is 2. :param selected_individuals: list of individuals to breed :param NetworkInfo network_info: Object storing network information. :return: newly bred generation

**calc_anchor_load_building**(network_info)

Finds the building with the highest load. :param network_info: Object storing network information. :return: building index of system load anchor

**disconnect_buildings**(network_info, new_plants)

Disconnects a random amount of buildings from the network. Setting the value ‘2’ means the building is disconnected. :param network_info: Object containing network information. :param new_plants: list of plants. :return: list of plant locations

**generate_initial_population**(network_info, network_layout)

Generates the initial population for network optimization.

**Parameters**
• **network_info** (*NetworkInfo*) – Object storing global network information (information about the whole optimization)

• **network_layout** (*NetworkLayout*) – Stores information about the specific network layout of an individual

**Returns** returns list of individuals as initial population for genetic algorithm

```python
cea.technologies.thermal_network.thermal_network_optimization.generate_plants(network_info, new_plants)
```
Generates the number of plants given in the config files at random, permissible, building locations. :param NetworkInfo network_info: Object storing network information. :return: list of plant locations

```python
cea.technologies.thermal_network.thermal_network_optimization.main(config)
```
Runs an optimization calculation for the plant location in the thermal network.

```python
cea.technologies.thermal_network.thermal_network_optimization.mutateLocation(individual, network_info)
```
Mutates an individuals plant location and number of plants, making sure not to violate any constraints. :param individual: List containing individual information :param network_info: Object storing network information :return: list of mutated individual information

```python
cea.technologies.thermal_network.thermal_network_optimization.mutateLoop(individual)
```
Mutates an individuals loop/branch information, making sure not to violate any constraints. :param individual: List containing individual information :return: list of mutated individual information

```python
cea.technologies.thermal_network.thermal_network_optimization.mutate_connections(individual, network_info)
```
Mutates an individuals plant location and number of plants, making sure not to violate any constraints. :param individual: List containing individual information :param network_info: Object storing network information :return: list of mutated individual information

```python
cea.technologies.thermal_network.thermal_network_optimization.mutate_generation(new_generation, network_info)
```
Checks if an individual should be mutated and calls the corresponding functions. :param new_generation: Generation to mutate :param NetworkInfo network_info: Object storing network information :return: Mutated generation

```python
cea.technologies.thermal_network.thermal_network_optimization.mutate_load(individual, network_info)
```
Mutates an individuals type of heat loads covered by the network, making sure not to violate any constraints. :param individual: List containing individual information :param network_info: Object storing network information :return: list of mutated individual information

```python
cea.technologies.thermal_network.thermal_network_optimization.network_cost_calculation(population, network_layout, network_info)
```
Main function which calls the objective function and stores values :param NetworkLayout network_layout: :param population: List containing all individuals of this generation :param NetworkInfo network_info: Object storing network information. :return: List of sorted tuples, lowest cost first. Each tuple consists of the cost, followed by the individual as a string.
cea.technologies.thermal_network.thermal_network_optimization.objective_function(
    network_info,
    network_layout,
    thermal_network
)

Calculates the cost of the given individual by generating a network and simulating it.

:return: total cost, opex and capex of the given individual

cea.technologies.thermal_network.thermal_network_optimization.output_results_of_all_individuals(
    config,
    locator,
    network_info
)

This function gathers all individuals evaluated and output results in one file.

:return:

cea.technologies.thermal_network.thermal_network_optimization.remove_one_random_plant(
    individual_buildings,
    network_info
)

Selects individuals from the previous generation for breeding and adds a predefined number of new “lucky” individuals which are new individuals added the gene pool.

:param sorted_previous_population: List of tuples of individuals from previous generation, sorted by increasing cost :param network_info: Object storing network information.
:return: list of individuals to breed

cea.technologies.thermal_network.thermal_network_optimization.thermal_network_optimization

cea.technologies.thermal_network.thermal_network_optimization.translate_individual(
    network_info,
    individual
)

Translates individual to prepare cost evaluation
.

Extract building plant locations (building names) and a list of disconnected buildings from the individual

Parameters

- **individual** – the individual to extract the information (coded as genes) from
- **network_info** (NetworkInfo) – Object storing network information.

Returns

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.boiler module

calc_Cinv_boiler(Q_design_W, technology_type, boiler_cost_data)

Calculates the annual cost of a boiler (based on A+W cost of oil boilers) [CHF / a] and Faz. 2012 data

Parameters
- **Q_design_W** (float) – Design Load of Boiler in [W]
- **technology_type**
- **boiler_cost_data**

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

calc_Cop_boiler(q_load_Wh, Q_nom_W, T_return_to_boiler_K)

This function calculates efficiency for operation of condensing Boilers based on LHV. This efficiency accounts for boiler efficiency only (not plant efficiency!)

Parameters
- **q_load_Wh** (float) – Load of time step
- **Q_nom_W** (float) – Design Load of Boiler
- **T_return_to_boiler_K** (float) – Return Temperature of the network to the boiler [K]

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

cond_boiler_op_cost(Q_therm_W, Q_design_W, T_return_to_boiler_K)

Calculates the operation cost of a Condensing Boiler supplying hot water up to 100 °C

Parameters
- **Q_therm_W** (float) – Load of time step
- **Q_design_W**
- **T_return_to_boiler_K** (float) – return temperature to Boiler (from DH network)

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

Returns C_boil_per_Wh float: cost per Wh in CHF / kWh

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

Returns E_aux_Boiler float: auxiliary electricity of boiler operation
cea.technologies.boiler.cond_boiler_operation($Q_{load\_W}$, $Q_{design\_W}$, $T_{return\_to\_boiler\_K}$)

This function calculates efficiency for operation of condensing Boilers supplying hot water up to 100 C 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

- $Q_{load\_W}$ (float) – Load of time step
- $Q_{design\_W}$ (float) – Design Load of Boiler
- $T_{return\_to\_boiler\_K}$ (float) – Return Temperature of the network to the boiler [K]

Returns

boiler_eff float

_efficiency of Boiler (Lower Heating Value), in abs. numbers


clea.technologies.burner module

gas burners

clea.technologies.burner.burner_op_cost($Q_{load\_W}$, $Q_{design\_W}$, FuelType, lca, prices)

This function calculates the operation cost of gas burners supplying heat directly to the high temperature generators in double effect absorption chillers. Assume similar operating cost as boilers.

- $Q_{load\_W}$ (float) – Load of time step
- $Q_{design\_W}$ (float) – Design Load of Boiler
- $T_{return\_to\_boiler\_K}$ (float) – return temperature to Boiler (from DH network)
- $C_{boil\_therm}$ (float) – Total generation cost for required load (per hour) in CHF
- $C_{boil\_per\_Wh}$ (float) – cost per Wh in CHF/kWh
- $Q_{primary}$ (float) – required thermal energy per hour (in Wh Natural Gas)
- $E_{aux\_Boiler}$ (float) – auxiliary electricity of boiler operation

Returns

- $C_{inv\_Ca}$ (float) – Annualized investment costs in CHF/a including Maintenance Cost
- $burner\_eff$ (float) – efficiency of Boiler (Lower Heating Value), in abs. numbers

_clea.technologies.chiller_absorption module

Absorption chillers

clas clea.technologies.chiller_absorption.AbsorptionChiller(chiller_prop, ACH_type)

Bases: object

__init__(chiller_prop, ACH_type)

x.__init__(...) initializes x; see help(type(x)) for signature
__slots__ = ['code', 'chiller_prop', 'm_cw_kgpers', 'm_hw_kgpers', 's_e', 'r_e', 's_g'

a_e
a_g
chiller_prop
code
e_e
e_g
m_cw_kgpers
m_hw_kgpers
r_e
r_g
s_e
s_g

update_data(chiller_prop)

Due to how AbsorptionChiller is currently used (FIXME: can we fix this?), we sometimes need to update
the instance variables from the dataframe chiller_prop.

cea.technologies.chiller_absorption.calc_Cinv_ACH(Q_nom_W, Absorption_chiller_cost_data, ACH_type)

Annualized investment costs for the vapor compressor chiller :type Q_nom_W : float :param Q_nom_W: peak
cooling demand in [W] :returns InvCa: annualized chiller investment cost in CHF/a :rtype InvCa: float

cea.technologies.chiller_absorption.calc_chiller_main(mdot_chw_kgpers,
T_chw_sup_K, T_chw_re_K,
T_hw_in_C, T_ground_K,
absorption_chiller)

This model calculates the operation conditions of the absorption chiller given the chilled water loads in evaporators
and the hot water inlet temperature in the generator (desorber). This is an empirical model using characteristic
equation method developed by [Kuhn A. & Ziegler F., 2005]. The parameters of each absorption chiller can
be derived from experiments or performance curves from manufacturer’s catalog, more details are described in
[Puig-Arnavat M. et al, 2010]. Assumptions: constant external flow rates (chilled water at the evaporator, cooling
water at the condenser and absorber, hot water at the generator). :param mdot_chw_kgpers: required chilled
water flow rate :type mdot_chw_kgpers: float :param T_chw_sup_K: required chilled water supply temperature
(outlet from the evaporator) :type T_chw_sup_K: float :param T_chw_re_K: required chilled water return tem-
perature (inlet to the evaporator) :type T_chw_re_K: float :param T_hw_in_C: hot water inlet temperature to
the generator :type T_hw_in_C: float :param T_ground_K: ground temperature :type T_ground_K: float :param
and adaptation of the characteristic equation. In: Proceedings of the interantional conference solar air condition-
characteristic equations of single- and double-effect absorption chillers by means of multivariable regression.

cea.technologies.chiller_absorption.calc_operating_conditions(absorption_chiller,
input_conditions)

Calculates chiller operating conditions at given input conditions by solving the characteristic equations and
the energy balance equations. This method is adapted from [Kuhn A. & Ziegler F., 2005]. The heat rejec-
tion to cooling tower is approximated with the energy balance: Q(condenser) + Q(absorber) = Q(generator) + Q(evaporator)
Parameters

- **chiller_prop** (*AbsorptionChiller*) – parameters in the characteristic equations and the external flow rates.

- **input_conditions** (*dict*) –

Returns a dict with operating conditions of the chilled water, cooling water and hot water loops in a absorption chiller.

To improve speed, the system of equations was solved using sympy for the output variable \( q_{hw\_kW} \) which is then used to compute the remaining output variables. The following code was used to create the expression to calculate \( q_{hw\_kW} \):

```python
# use symbolic computation to derive a formula for q_hw_kW:

# first, make sure all the variables are sympy symbols:
T_chw_in_C, T_chw_out_C, T_cw_in_C, T_hw_in_C, mcp_cw_kWperK, mcp_hw_kWperK, q_chw_kW = sympy.symbols("T_chw_in_C, T_chw_out_C, T_cw_in_C, T_hw_in_C, mcp_cw_kWperK, mcp_hw_kWperK, q_chw_kW")
T_hw_out_C, T_cw_out_C, q_hw_kW = sympy.symbols('T_hw_out_C, T_cw_out_C, q_hw_kW')
a_e, a_g, e_e, e_g, r_e, r_g, s_e, s_g = sympy.symbols("a_e, a_g, e_e, e_g, r_e, r_g, s_e, s_g")

# the system of equations:

eq_e = s_e * ddt_e + r_e - q_chw_kW

# solve the system of equations with sympy:

eq_sys = [eq_e, eq_g, eq_bal_g, eq_ddt_e, eq_ddt_g]
unknown_variables = (T_hw_out_C, T_cw_out_C, q_hw_kW, ddt_e, ddt_g)

a, b = sympy.linear_eq_to_matrix(eq_sys, unknown_variables)
T_hw_out_C, T_cw_out_C, q_hw_kW, ddt_e, ddt_g = tuple(*(sympy.linsolve(eq_sys, unknown_variables)))
q_hw_kW.simplify()
```


**cea.technologies.chiller_absorption.calc_power_demand**(*q_chw_W*, *chiller_prop*)

Calculates the power demand of the solution and refrigeration pumps in absorption chillers. Linear equations derived from manufacturer’s catalog _[Broad Air Conditioning, 2018]._

```python
:returns:
run the whole preprocessing routine test case 1) \( q_{hw}\_W = 24213 \), \( q_{chw}\_W = 20088 \), \( EER = 0.829 \), \( T_{hw}\_out\_C = 67.22 \) \([\text{Kuhn, 2011}]\) test case 2) \( q_{hw}\_W = 824105 \), \( q_{chw}\_W = 1163011 \), \( EER = 1.41 \), \( T_{hw}\_out\_C = 165.93 \) \([\text{Shirazi, 2016}]\) test case 3) \( q_{hw}\_W = 623379 \), \( q_{chw}\_W = 1163430 \), \( EER = 1.87 \), \( T_{hw}\_out\_C = 195.10 \) \([\text{Shirazi, 2016}]\)


### cea.technologies.chiller_vapor_compression module

Vapor-compressor chiller

```python
ccea.technologies.chiller_vapor_compression.calc_COP(T\_cw\_in\_K, T\_chw\_re\_K, q\_chw\_load\_Wh)
ccea.technologies.chiller_vapor_compression.calc_COP\_with\_carnot\_efficiency(T\_evap\_K, T\_cond\_K, g\_value)
ccea.technologies.chiller_vapor_compression.calc_Cinv\_VCC(Q\_nom\_W, locator, technology\_type)
ccea.technologies.chiller_vapor_compression.calc_VCC(q\_chw\_load\_Wh, T\_chw\_sup\_K, T\_chw\_re\_K, T\_cw\_in\_K, g\_value)
ccea.technologies.chiller_vapor_compression.calc_VCC\_COP(weather\_data, load\_types, centralized=True)
```

#### Example


Calculates the VCC COP based on evaporator and compressor temperatures, VCC g-value, and an assumption of auxiliary power demand for centralized and decentralized systems. This approximation only works in tropical climates


#### Parameters

- **load\_types** – a list containing the systems (aru, ahu, scu) that the chiller is supplying for
- **centralized** –
Returns

cea.technologies.chiller_vapor_compression.get_max_VCC_unit_size(locator, 
  VCC_code='CH3')

cea.technologies.chiller_vapor_compression.main()

c.technologies.cogeneration module

cogeneration (combined heat and power)

cea.technologies.cogeneration.calc_CC_operation(el_output_from_GT_W, 
  GT_size_W, 
  fuel_type, T_sup_K)
Operation Function of Combined Cycle at given electricity input to run the gas turbine (el_input_W). The 
gas turbine (GT) exhaust gas is used by the steam turbine (ST). :type el_output_from_GT_W : float :param 
el_output_from_GT_W: Electricity input to run the gas turbine (only GT output, not CC output!) :type 
GT_size_W : float :param GT_size_W: size of the gas turbine and (not CC)(P_el_max) :type fuel_type : string 
:param fuel_type: fuel used, either ‘NG’ (natural gas) or ‘BG’ (biogas) :type T_sup_K : float :param T_sup_K: 
plant supply temperature to district heating network (hot) or absorption chiller :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_el : float :param eta_el: total electric efficiency :rtype eta_thermal : 
float :returns eta_thermal: total thermal efficiency :rtype eta_total : float :returns eta_total: sum of total electric 
and thermal efficiency

cea.technologies.cogeneration.calc_Cinv_CCGT(CC_size_W, CCGT_cost_data)
Annualized investment costs for the Combined cycle :type CC_size_W : float :param CC_size_W: 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 polygeneration energy 
conversion technologies., PhD Thesis, EPFL

cea.technologies.cogeneration.calc_Cinv_FC(P_design_W, FC_cost_data)
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+Mitarbeitende&utm_ 
campaign=06d2c528a5-1_Newsletter_2014_Mitarbeitende_DE&utm_medium=email&utm_term=0_
2e79c1703c-06d2c528a5- :type P_design_W : float :param P_design_W: Design thermal Load of Fuel 
Cell [W_th] :rtype InvCa: float :returns InvCa: annualized investment costs in CHF

cea.technologies.cogeneration.calc_GT_operation_fullload(gt_size_W, fuel_type)
Calculates gas turbine efficiency and exhaust gas mass flow rate at full load.
fuel_type: fuel used, either NG (Natural Gas) or BG (Biogas) :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.calc_GT_operation_partload(wdot_W, gt_size_W, eta0, 
  m0_exhaust_from_GT_kgpers, 
  fuel_type)
Calculates GT operational parameters at part load
  :type wdot_W : float :param wdot_W: GT electric output (load) :type gt_size_W : float :param gt_size_W: 
Maximum electric load that is demanded to the gas turbine :type eta0 : float :param eta0: GT electric efficiency 
at full-load :type m0_exhaust_from_GT_kgpers : float :param m0_exhaust_from_GT_kgpers: GT exhaust gas 
mass flow at full-load :type fuel_type : string :param fuel_type: fuel used, either ‘NG’ (natural gas) or ‘BG’ 
temperature :rtype mdotfuel : float :returns mdotfuel: mass flow rate of fuel(gas) requirement ..}[C. Weber,

**cea.technologies.cogeneration.calc_ST_operation**

\[ \text{m}_{\text{exhaust_GT}_{\text{kgpers}}} \text{, } T_{\text{exhaust_GT}_{\text{K}}} \text{, } T_{\text{sup}_{\text{K}}} \text{, } \text{fuel_type} \]

Operation of a double pressure (LP,HP) steam turbine connected to a district heating network following [C. Weber, 2008].

- \( \text{m}_{\text{exhaust_GT}_{\text{kgpers}}} \): float
- \( T_{\text{exhaust_GT}_{\text{K}}} \): float
- \( T_{\text{sup}_{\text{K}}} \): float
- \( \text{fuel_type} \): fuel used, either ‘NG’ (natural gas) or ‘BG’ (biogas)

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

**cea.technologies.cogeneration.calc_cop_CCGT**

\[ \text{GT}_{\text{size_W}} \text{, } T_{\text{sup}_{\text{K}}} \text{, } \text{fuel_type} \]

This function calculates the COP of a combined cycle, the gas turbine (GT) exhaust gas is used by the steam turbine (ST) to generate electricity and heat. This function iterates the combined cycle operation between its nominal capacity and minimum load and generate linear functions of the GT operation. The generated function calculates operation points and associated costs of the cogeneration at given thermal load \( Q_{\text{therm-requested}} \).

How to use the return functions: input \( Q_{\text{therm-requested}} \) into the output interpolation functions

- \( \text{GT}_{\text{size_W}} \): float
- \( T_{\text{sup}_{\text{K}}} \): float
- \( \text{fuel_type} \): string

**cea.technologies.cogeneration.calc_eta_FC**

\[ Q_{\text{load_W}} \text{, } Q_{\text{design_W}} \text{, } \phi_{\text{threshold}} \text{, } \text{approach_call} \]

Efficiency for operation of a SOFC (based on LHV of NG) including all auxiliary losses Valid for \( Q_{\text{load}} \) in range of 1-10 [kW_el] Modeled after:

- **Approach B (Empiric Approach)**: [Iain Staffell]_
cea.technologies.constants module

Constants used throughout the cea.technologies package.

History lesson: This is a first step at removing the cea.globalvars.GlobalVariables object.

cea.technologies.cooling_tower module

System Modeling: Cooling tower

cea.technologies.cooling_tower.calc_CT(q_hot_Wh, Q_nom_W)
For the operation of a water condenser + direct cooling tower based on [B. Stephane, 2012]. Maximum cooling power is 10 MW.

:type q_hot_Wh : float :param q_hot_Wh: heat rejected from chiller condensers :type Q_nom_W : float :param Q_nom_W: installed CT size


cea.technologies.cooling_tower.calc_CT_partload_factor(q_part_load_ratio)

cea.technologies.cooling_tower.calc_CT_yearly(q_hot_kWh)
For the operation of a water condenser + direct cooling tower with a fit function based on the hourly calculation in calc_CT.

:type q_hot_kWh : float :param q_hot_kWh: heat rejected from chiller condensers

cea.technologies.cooling_tower.calc_Cinv_CT(Q_nom_CT_W, locator, technology_type)
Annualized investment costs for the Combined cycle

:type Q_nom_CT_W : float :param Q_nom_CT_W: Nominal size of the cooling tower in [W]
:rtype InvCa : float :returns InvCa: annualized investment costs in Dollars

cea.technologies.cooling_tower.main(locator)

direct expansion units

cia.technologies.direct_expansion_units.calc_Cinv_DX(Q_design_W)

cia.technologies.direct_expansion_units.calc_DX(mdot_kgpers, T_sup_K, T_re_K)
ncia.technologies.direct_expansion_units.calc_cop_DX(Q_load_W)
**cea.technologies.furnace module**

*cea.technologies.furnace.calc_Cinv_furnace*(Q_design_W, locator, technology_type)
Calculates the annualized investment cost of a Furnace based on Bioenergy 2020 (AFO) and POLYCITY Ostfeldern.

- :type Q_design_W : float :param Q_design_W: Design Load of Boiler
- :type Q_annual_W : float :param Q_annual_W: annual thermal Power output [Wh]
- :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)
Efficiency for co-generation plant with wood chip furnace, based on LHV. Electricity is produced through organic rankine cycle.

- 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’)
- 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 eta_el: electric efficiency of Furnace (LHV), in abs. numbers
- :rtype Q_aux : float :returns Q_aux: auxiliary power for Plant operation [W]

*cea.technologies.furnace.furnace_op_cost*(Q_therm_W, Q_design_W, T_return_to_boiler_K, MOIST_TYPE)
Calculates the operation cost of a furnace plant (only operation, no annualized cost!)

- :type Q_therm_W : float :param Q_therm_W: thermal energy required from furnace plant in [Wh]
- :type Q_design_W : float :param Q_design_W: Design Load of Boiler [W]
- :type T_return_to_boiler_K : float :param T_return_to_boiler_K: return temperature to the boiler
- :type MOIST_TYPE : float :param MOIST_TYPE: moisture type of the fuel, set in MasterToSlaveVariables (‘wet’ or ‘dry’)
- :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

```python
cea.technologies.heat_exchangers.calc_Cinv_HEX(Q_design_W, locator, config, technology_type)
```

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

- :type Q_design_W : float :param Q_design_W: Design Load of Boiler
- :rtype InvC_return : float :returns InvC_return: total investment Cost in [CHF]
- :rtype InvCa : float :returns InvCa: annualized investment costs in [CHF/a]

```python
cea.technologies.heat_exchangers.calc_Cinv_HEX_hisaka(network_info)
```

Calculates costs of all substation heat exchangers in a network. Used in thermal_network_optimization.

**cea.technologies.heating_coils module**

Heating and cooling coils of Air handling units

```python
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)
```

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

**Parameters**

- Q – cooling load
- thi – inlet temperature of primary side
- tho – outlet temperature of primary side
- tci – inlet 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

**Returns**

- tci = inlet temperature of secondary side (district cooling network)
- tco = outlet temperature of secondary side (district cooling network)
- cc = capacity mass flow rate secondary side

```python
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)
```

this function calculates the state of the heat exchanger at the substation of every customer with heating needs in an analogous way to the cooling coil calculation

**Parameters**

- Q – cooling load
• $\text{thi}$ – in temperature of primary side
• $\text{tho}$ – out temperature of primary side
• $\text{tci}$ – in temperature of secondary side
• $\text{ch}$ – capacity mass flow rate primary side
• $\text{ch}_0$ – nominal capacity mass flow rate primary side
• $Q_{\text{nom}}$ – nominal cooling load
• $\text{thi}_0$ – nominal in temperature of primary side
• $\text{tci}_0$ – nominal in temperature of secondary side
• $\text{tho}_0$ – nominal out temperature of primary side

Returns

• $\text{tci}$ = inlet temperature of secondary side (district heating network)
• $\text{tco}$ = out temperature of secondary side (district heating network)
• $\text{cc}$ = capacity mass flow rate secondary side

**cea.technologies.heatpumps module**

heatpumps

ccea.technologies.heatpumps.GHP_Op_max ($Q_{\text{max\_GHP\_W}}, tsup_K, tground_K$)

For the operation of a Geothermal heat pump (GSHP) at maximum capacity supplying DHN.

:type $tsup_K$ : float :param $tsup_K$: supply temperature to the DHN (hot)
:type $tground_K$ : float :param $tground_K$: ground temperature
:type nProbes: float :param nProbes: number of probes
: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 ($mdot_kgpers, t_sup_K, t_re_K, t_sup_GHP_K, Q_{\text{therm\_GHP\_W}}$)**

Operation cost of sewage water HP supplying DHN

:type $mdot_kgpers$ : float :param $mdot_kgpers$: supply mass flow rate to the DHN
:type $t_sup_K$ : float :param $t_sup_K$: supply temperature to the DHN (hot)
:type $t_re_K$ : float :param $t_re_K$: return temperature from the DHN (cold)
:type $t_sup_GHP_K$ : float :param $t_sup_GHP_K$: sewage supply temperature
:rtype $C_{\text{HP\_Sew\_el\_pure}}$: float :returns $C_{\text{HP\_Sew\_el\_pure}}$: electricity cost of sewage water HP operation

**Rtype C_{\text{HP\_Sew\_per\_kWh\_th\_pure}}** float

**Returns C_{\text{HP\_Sew\_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_{\text{therm}}** float

**Returns q_{\text{therm}}** thermal energy supplied to DHN

**Rtype wdot** float

**Returns wdot** electricity required for sewage water HP operation
For the operation of a Heat pump between a district heating network and a lake:

```python
cea.technologies.heatpumps.HPLake_Op (mdot_kgpsers, t_sup_K, t_re_K, t_lake_K)
```

- `type mdot_kgpsers : float` - supply mass flow rate to the DHN
- `type t_sup_K : float` - supply temperature to the DHN (hot)
- `type t_re_K : float` - return temperature from the DHN (cold)
- `type t_lake_K : float` - lake temperature
- `rtype wdot_el : float` - total electric power requirement for compressor and auxiliary el.
- `rtype qcolddot : float` - cold power requirement

For the operation of lake heat pump supplying DHN:

```python
cea.technologies.heatpumps.HPLake_op_cost (Q_gen_W, tsup_K, tret_K, tlake)
```

- `type mdot_kgpsers : float` - supply mass flow rate to the DHN
- `type tsup_K : float` - supply temperature to the DHN (hot)
- `type tret_K : float` - return temperature from the DHN (cold)
- `type tlake : float` - lake temperature
- `rtype C_HPL_el: float` - electricity cost of Lake HP operation
- `Rtype wdot float` - electricity required for Lake HP operation
- `Rtype Q_cold_primary float` - cold power requirement
- `Rtype Q_therm float` - thermal energy supplied to DHN

For the operation of sewage water HP supplying DHN:

```python
cea.technologies.heatpumps.HPSew_op_cost (mdot_kgpsers, t_sup_K, t_re_K, t_sup_sew_K, Q_therm_Sew_W)
```

- `type mdot_kgpsers : float` - supply mass flow rate to the DHN
- `type t_sup_K : float` - supply temperature to the DHN (hot)
- `type t_re_K : float` - return temperature from the DHN (cold)
- `type t_sup_sew_K : float` - sewage supply temperature
- `rtype C_HPSew_el_pure: float` - electricity cost of sewage water HP operation
- `Rtype C_HPSew_per_kWh_th_pure float` - electricity cost per kWh thermal energy produced from sewage water HP
- `Rtype qcolddot float` - cold power requirement
- `Rtype q_therm float` - thermal energy supplied to DHN
- `Rtype wdot float` - electricity required for sewage water HP operation
For the operation of a heat pump (direct expansion unit) connected to minisplit units

```python
cea.technologies.heatpumps.HP_air_air(mdot_cp_WC, t_sup_K, t_re_K, tsource_K)
```

For the operation of a heat pump (direct expansion unit) connected to minisplit units

**Parameters**
- `mdot_cp_WC`: float :param mdot_cp_WC: capacity mass flow rate.
- `t_sup_K`: float :param t_sup_K: supply temperature to the minisplit unit (cold).
- `t_re_K`: float :param t_re_K: return temperature from the minisplit unit (hot).
- `tsource_K`: float :param tsource_K: temperature of the source.

**Returns**
- `wdot_el`: float: total electric power requirement for compressor and auxiliary el.
- `qcolddot`: float: cold power requirement.

Calculates the annualized investment costs for the geothermal heat pump

```python
cea.technologies.heatpumps.calc_Cinv_GHP(GHP_Size_W, GHP_cost_data, BH_cost_data)
```

**Parameters**
- `GHP_Size_W`: float :param GHP_Size_W: Design electrical size of the heat pump in [Wel]

**Returns**
- `InvCa`: float: annualized investment costs in EUROS/a

Calculates the annualized investment costs for a water to water heat pump.

```python
cea.technologies.heatpumps.calc_Cinv_HP(HP_Size, locator, technology_type)
```

**Parameters**
- `HP_Size`: float :param HP_Size: Design thermal size of the heat pump in [W]

**Returns**
- `InvCa`: float: annualized investment costs in [CHF/a]

For the operation of a Geothermal heat pump (GSHP) supplying DHN.

```python
cea.technologies.heatpumps.calc_Cop_GHP(ground_temp_K, mdot_kgpers, T_DH_sup_K, T_re_K)
```

**Parameters**
- `mdot_kgpers`: float :param mdot_kgpers: supply mass flow rate to the DHN.
- `T_DH_sup_K`: float :param T_DH_sup_K: supply temperature to the DHN (hot).
- `T_re_K`: float :param T_re_K: return temperature from the DHN (cold).

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

Calculates the annualized investment costs for a water to water heat pump.

```python
cea.technologies.heatpumps.Pump_operation(P_design)
```

**Modeled after:**
- 05_merkblatt_wirtschaftlichkeit_14.pdf
- 23_merkblatt_pumpen_web.pdf
- ER_2010_11_Heizungspumpen.pdf
P\_design [float] Load of time step

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

c\textvisiblespace{}cea.technologies.pumps\textvisiblespace{}.\textvisiblespace{calc\textvisiblespace{Cinv\textvisiblespace{}pump}}\textvisiblespace{}(deltaP, \textvisiblespace{mdot\textvisiblespace{kgpers}, \textvisiblespace{eta\textvisiblespace{pumping}, locator, technology\textvisiblespace{type}}})

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 Pe\_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\_kgpers : float :param mdot\_kgpers: 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 :type InvCa : float :returns InvCa: annualized investment costs in CHF/year

cea.technologies.pumps\textvisiblespace{}.\textvisiblespace{calc\textvisiblespace{Ctot\textvisiblespace{}pump}}\textvisiblespace{}(master\_to\_slave\_vars, \textvisiblespace{network\_features, locator, network\_type})

Computes the total pump investment cost :type master\_to\_slave\_vars : class context :type network\_features : class ntwFeatures :rtype pump\_Costs : float :returns pump\_Costs: pumping cost

cea.technologies.pumps\textvisiblespace{}.\textvisiblespace{calc\textvisiblespace{water\textvisiblespace{}body\textvisiblespace{uptake\textvisiblespace{}pumping}}\textvisiblespace{}(Q\_gen\_W, T\_district\_return\_K, T\_district\_supply\_K)\textvisiblespace{}

cea.technologies.radiators module

heating radiators

cea.technologies.radiators\textvisiblespace{}.\textvisiblespace{calc\textvisiblespace{radiator}}\textvisiblespace{}(Qh, tair, Qh0, tair0, tsh0, trh0)

Calculates the supply and return temperature as well as the flow rate of water in radiators similar to the model implemented by Holst (1996) using a Newton-Raphson solver.

\textbf{Parameters}

- Qh – Space heating demand in the building
- tair – environment temperature in the building
- Qh0 – nominal radiator power
- tair0 –
- tsh0 –
- trh0 –

\textbf{Returns}


cea.technologies.radiators\textvisiblespace{}.\textvisiblespace{fh}(x, delta\_t, Qh0, Qh, tair, LMRT0, nh)

Static radiator heat balance equation from Holst (1996), eq. 6.

\textbf{Parameters}

- x – radiator exhaust temperature
- mCw0 – nominal radiator mass flow rate
- Qh – space heating demand
- Qh0 – nominal space heating power
- tair – air temperature in the building
• **LMRT** – nominal logarithmic temperature difference

• **nh** – radiator constant

**Returns**

`cea.technologies.radiators.lmrt(tair, trh, tsh)`  
Logarithmic temperature difference (Eq. 3 in Holst, 1996) :param `tair`: environment temperature in the room :param `trh`: radiator exhaust temperature :param `tsh`: radiator supply temperature :return:

**cea.technologies.storage_tank module**

Sensible Heat Storage - Fully Mixed tank

`cea.technologies.storage_tank.calc_cold_storage_charge_HEX(mcp_cold_0, Q_cold_0, T_hot_in, T_cold_in, T_cold_out)`

**Parameters**

• **mcp_cold_0** – nominal capacity mass flow rate primary side  
• **Q_cold_0** – nominal cooling load  
• **T_hot_in** – nominal in temperature of secondary side  
• **T_cold_in** – nominal in temperature of primary side  
• **T_cold_out** – nominal out temperature of primary side

**Return**  
Area_HEX_m2 Heat exchanger area in [m2]  
Return UA_WperK [W/K]

`cea.technologies.storage_tank.calc_cold_storage_discharge_HEX(m_hot_0, Q_hot_0, T_cold_in, T_hot_in, T_hot_out)`

**Parameters**

• **mcp_hot** – secondary side  
• **Q_hot** – required heat from the primary side  
• **T_cold_in**  
• **T_hot_in**  
• **T_hot_out**

**Returns**

`cea.technologies.storage_tank.calc_cold_tank_heat_loss(Area_tank_surface_m2, T_tank_C, T_ambient_C)`

`cea.technologies.storage_tank.calc_dhw_tank_heat_balance(T_int_C, T_ext_C, T_tank_C, V_tank_m3, q_tank_discharged_W, area_tank_surface_m2)`

This algorithm calculates the heat flows within a fully mixed water storage tank. Heat flows include sensible heat loss to the environment (q_loss_W), heat charged into the tank (q_charged_W), and heat discharged from the tank (q_discharged_W).
Parameters

- \( T_{\text{tank}_C}(\text{float}) \) – tank temperature in [C]
- \( T_{\text{ww\_setpoint}}(\text{float}) \) – DHW temperature set point in [C]
- \( T_{\text{int}_C}(\text{float}) \) – room temperature in [C]
- \( T_{\text{ext}_C}(\text{float}) \) – ambient temperature in [C]
- \( V_{\text{tank\_m3}}(\text{float}) \) – DHW tank size in [m3]

Return \( q_{\text{loss\_W}} \) storage sensible heat loss in [Wh].

Return \( q_{\text{discharged\_W}} \) heat discharged from the tank in [Wh], including dhw heating demand and distribution heat loss.

Return \( q_{\text{charged\_W}} \) heat charged into the tank in [Wh].

Rtype \( q_{\text{loss\_W}} \) float

Rtype \( q_{\text{discharged\_W}} \) float

Rtype \( q_{\text{charged\_W}} \) float

\begin{verbatim}
cea.technologies.storage_tank.calc_fully_mixed_tank(T_{\text{start}_C}, T_{\text{ambient}_C}, q_{\text{discharged}_W}, q_{\text{charged}_W}, V_{\text{tank\_m3}}, Area_{\text{tank\_surface\_m2}}, tank_type)
\end{verbatim}

Temporary inputs: \( T_{\text{start}_C} = 6 \) degree C \( T_{\text{ambient}_C} = \) external temperature of each time step \( q_{\text{discharged}_W} = \) cooling loads fulfilled by tank \( q_{\text{charged}_W} = \) excess cooling from VCC or ACH \( V_{\text{tank\_m3}} = \) volume of tank

the output should be saved and used as \( T_{\text{start}_C} \) in the next time step

Parameters

- \( T_{\text{start}_C} \) – Tank temperature at the beginning of the time step
- \( T_{\text{ambient}_C} \) – Ambient temperature at the location of tank
- \( q_{\text{discharged}_W} \) – thermal energy discharged from tank
- \( q_{\text{charged}_W} \) – thermal energy charged to tank
- \( V_{\text{tank\_m3}} \) – tank volume

Returns

\begin{verbatim}
cea.technologies.storage_tank.calc_hot_tank_heat_loss(Area_{\text{tank\_surface\_m2}}, T_{\text{tank}_C}, tamb)
\end{verbatim}

\begin{verbatim}
cea.technologies.storage_tank.calc_storage_tank_volume(Qc_{\text{tank\_capacity\_Wh}}, T_{\text{tank\_fully\_charged}_K}, T_{\text{tank\_fully\_discharged}_K})
\end{verbatim}

\begin{verbatim}
cea.technologies.storage_tank.calc_tank_surface_area(V_{\text{tank\_m3}})
\end{verbatim}

\begin{verbatim}
cea.technologies.storage_tank.calc_tank_temperature(T_{\text{start}_C}, q_{\text{loss}_W}, q_{\text{discharged}_W}, q_{\text{charged}_W}, V_{\text{tank\_m3}}, tank_type)
\end{verbatim}

This algorithm solves the differential equation, ode.

Parameters

- \( T_{\text{start}_C}(\text{float}) \) – initial tank temperature in [C]
• **q_loss_W** (*float*) – storage tank sensible heat loss in Wh.
• **q_discharged_W** (*float*) – heat discharged from the tank in Wh.
• **q_charged_W** (*float*) – heat charged into the tank in Wh.
• **V_tank_m3** (*float*) – DHW tank size in [m3]

Returns **T_tank_C** tank temperature after the energy balance

Rtype **T_tank_C** float
c.technologies.storage_tank.ode_cold_water_tank(*y*, *t*, **q_gain_W**, **q_discharged_W**, **q_charged_W**, **V_tank_m3**)

This algorithm describe the energy balance of the dhw tank with a differential equation.

Parameters
• **y** (*float*) – storage temperature in C.
• **t** (*float*) – time steps.
• **q_gain_W** (*float*) – storage tank sensible heat loss in W.
• **q_discharged_W** (*float*) – heat discharged from the tank in W.
• **q_charged_W** (*float*) – heat charged into the tank in W.
• **V_tank_m3** (*float*) – DHW tank size in [m3]

Return **dydt** change in temperature at each time step.

c.technologies.storage_tank.ode_hot_water_tank(*y*, *t*, **q_loss_W**, **q_discharged_W**, **q_charged_W**, **V_tank_m3**)

This algorithm describe the energy balance of the dhw tank with a differential equation.

Parameters
• **y** (*float*) – storage temperature in C.
• **t** (*float*) – time steps.
• **q_loss_W** (*float*) – storage tank sensible heat loss in W.
• **q_discharged_W** (*float*) – heat discharged from the tank in W.
• **q_charged_W** (*float*) – heat charged into the tank in W.
• **V_tank_m3** (*float*) – DHW tank size in [m3]

Return **dydt** change in temperature at each time step.

c.technologies.substation module

Substation Model
c.technologies.substation.calc_DC_return(*t_0*, *t_1*)

This function calculates the return temperature of the district cooling network according to the maximum observed (different to zero) in all buildings connected to the grid.

Parameters
• **t_0** – last maximum temperature
• **t_1** – current maximum temperature to evaluate

Returns **tmin**, new maximum temperature
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

cala.technologies.substation.calc_DH_return(t_0, t_1)

This function calculates the return temperature of the district heating network according to the minimum observed in all buildings connected to the grid.

**Parameters**
- `t_0` – last minimum temperature
- `t_1` – current minimum temperature

**Returns** `tmax`, new minimum temperature

cala.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.

**Parameters**
- `t_0` – last maximum temperature
- `t_1` – current maximum temperature

**Returns** `tmax`, new maximum temperature

cala.technologies.substation.calc_HEX_mix_2_flows(Q1, Q2, m1, m2, t1, t2)

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.

cala.technologies.substation.calc_HEX_mix_3_flows(Q1, Q2, Q3, m1, m2, m3, t1, t2, t3)

cala.technologies.substation.calc_area_HEX(Qnom, dTm_0, U)

This function calculates the area of a heat exchanger at nominal conditions.

**Parameters**
- `Qnom` – nominal load
- `dTm_0` – nominal logarithmic temperature difference
• $U$ – coefficient 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_compound_Tcs(building_demand_df, cooling_configuration)
```

```
cea.technologies.substation.calc_compound_Ths(building_demand_df, heating_configuration)
```

```
cea.technologies.substation.calc_dTm_HEX(thi, tho, tci, tco)
```

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

Returns

• $dtm$ = logarithmic temperature difference

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

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

Parameters

• $Q$ – cooling load
• $thi$ – in temperature of primary side
• $tho$ – out temperature of primary side
• $tci$ – in temperature of secondary side
• $ch$ – capacity mass flow rate primary side in W
• $ch_0$ – 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

Returns

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

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

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

• \( Q \) – heating load
• \( \text{thi} \) – in temperature of secondary side
• \( \text{tco} \) – out temperature of primary side
• \( \text{tci} \) – in temperature of primary side
• \( \text{cc} \) – capacity mass flow rate primary side
• \( \text{cc}_0 \) – nominal capacity mass flow rate primary side
• \( \text{Qnom} \) – nominal cooling load
• \( \text{thi}_0 \) – nominal in temperature of secondary side
• \( \text{tci}_0 \) – nominal in temperature of primary side
• \( \text{tco}_0 \) – nominal out temperature of primary side

Returns

• \( \text{tho} \) = out temperature of secondary side (district cooling network)
• \( \text{ch} \) = capacity mass flow rate secondary side
• \( \text{Area}_{\text{HEX heating}} \) = area of heat exchanger.

```python
ccea.technologies.substation.calc_temp_hex_building_side_cooling(building_demand_df, cooling_configuration)
ccea.technologies.substation.calc_temp_hex_building_side_heating(building_demand_df, heating_configuration)
ccea.technologies.substation.calc_temp_this_building_cooling(T_supply_to_cs_ref, T_supply_to_cs_ref_data)
ccea.technologies.substation.calc_temp_this_building_heating(Tww_Ths_supply_C)
ccea.technologies.substation.main(config)
run the whole network summary routine
ccea.technologies.substation.substation_main_cooling(locator, total_demand, buildings_name_with_cooling, cooling_configuration=['aru', 'ahu', 'scu'], DCN_barcode='')
ccea.technologies.substation.substation_main_heating(locator, total_demand, buildings_name_with_heating, heating_configuration=7, DHN_barcode='')
ccea.technologies.substation.substation_model_cooling(name, building, T_DC_supply_to_cs_ref_C, T_DC_supply_to_cs_ref_data_C, Tcs_supply_C, Tcs_return_C, cs_configuration, locator, DCN_barcode='')```
cea.technologies.substation.substation_model_heating(name, building_demand_df, T_DH_supply_C, Ths_supply_C, Ths_return_C, hs_configuration, locator, DHN_barcode="")

Parameters

• locator – path to locator function
• building_demand_df – dataframe with consumption data per building
• T_heating_sup_C – vector with hourly temperature of the district heating network without losses
• T_DH_sup_C – vector with hourly temperature of the district heating network with losses
• T_DC_sup_C – vector with hourly temperature of the district cooling 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
• DHN_barcode – this is default to "" which means that it is created for decentralized buildings "0101011001" is a common type used during optimization

Returns

• Dataframe stored for every building with the mass flow rates and temperatures district heating and cooling
• where fName_result: ID of the building accounting for the individual at which it belongs to.

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 “Thermodynamische Systeme (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_storage module

thermal storage

cea.technologies.thermal_storage.calc_Cinv_storage(V_tank_m3, locator, config, technology_type)

calculate the annualized investment cost of a thermal storage tank

**Parameters**

- V_tank_m3 (float) – storage tank volume

**Returns**

InvCa

cea.tests package

Submodules

cea.tests.create_unittest_data module

Create the data for cea/tests/test_calc_thermal_loads.py

Run this script when the core algorithms get updated and the unitests in test_calc_thermal_loads.py stop working. The script overwrites the file cea/tests/test_calc_thermal_loads.config which contains the data used for the unit tests. You can safely ignore the output printed to STDOUT - it is used for debugging purposes only.

**NOTE:** Check first to make sure the core algorithms are correct, i.e. the changes to the outputs behave as expected.

cea.tests.create_unittest_data.main(output_file)

cea.tests.create_unittest_data.run_for_single_building (building, bpr, weather_data, date_range, locator, use_dynamic_infiltration_calculation, resolution_outputs, loads_output, mass-flows_output, temperatures_output, config, debug)

cea.tests.dodo module

“Makefile” for doit to test the whole package with Jenkins.

This file gets run with the cea test command as well as by the Jenkins continuous integration server. It runs all the unit tests in the cea/tests/ folder as well as some of the CEA scripts, to make sure they at least run through.

In order to run reference cases besides the one called “open”, you will need to set up authentication to the private GitHub repository. To do this, you need to create a GitHub [authentication token](https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/). Then, create a text file called cea_github.auth in your home directory (e.g. C:\Users\your-user-name for Windows systems or equivalent on POSIX systems, ask your administrator if you don’t know what this is). The file should contain two lines, the first being your GitHub user name the second the authentication token.

The reference cases can be found here: https://github.com/architecture-building-systems/cea-reference-case/archive/master.zip

cea.tests.dodo.get_github_auth()

get the username / token for github from a file in the home directory called “cea_github.auth”. The first line contains the user, the second line the personal access token.
Returns (user, token)

cea.tests.dodo.main(config)

cea.tests.dodo.task_download_reference_cases()
Download the (current) state of the reference cases

cea.tests.dodo.task_run_data_helper()
Run the data helper for each reference case

cea.tests.dodo.task_run_demand()
run the demand script for each reference cases and weather file

cea.tests.dodo.task_run_embodied_energy()
Run the embodied energy script for each reference case

cea.tests.dodo.task_run_emissions_mobility()
run the emissions mobility script for each reference case

cea.tests.dodo.task_run_emissions_operation()
run the emissions operation script for each reference case

cea.tests.dodo.task_run_sensitivity()
Run the sensitivity analysis for the the reference-case-open

cea.tests.dodo.task_run_thermal_network()
run the thermal_network for the included reference case

cea.tests.dodo.task_run_unit_tests()
run the unittests

clea.tests.test_calc_thermal_loads module

class clea.tests.test_calc_thermal_loads.TestCalcThermalLoads(methodName='runTest')
Bases: unittest.case.TestCase

This test case contains the two tests :py:meth:`test_calc_thermal_loads` and test_calc_thermal_loads_other_buildings(). They are not stricty unit tests, but rather test the whole thermal loads calculation (for the built-in reference case) against a set of known results, stored in the file test_calc_thermal_loads.config - if the results should change and the change has been verified, you can use the script create_unittest_data.py to update the config file with the new results.

classmethod setUpClass()
Hook method for setting up class fixture before running tests in the class.

test_calc_thermal_loads()

test_calc_thermal_loads_other_buildings()
Test some other buildings just to make sure we have the proper data

cea.tests.test_calc_thermal_loads.run_for_single_building(building, bpr, weather_data, date, locator, use_dynamic_infiltration_calculation, resolution_output, loads_output, mass-flows_output, temperatures_output, config, debug)
cea.tests.test_check_for_radiation_input_in_demand_script module

class cea.tests.test_check_for_radiation_input_in_demand_script.TestCheckForRadiationInputInDemandScript
    Bases: unittest.case.TestCase

Tests to make sure the demand script raises a `ValueError` if applied to a reference case that does not have the radiation script output.
This fixes the issue #222

test_demand_checks_radiation_daysim_script()

cea.tests.test_config module

Test the `cea.config.Configuration()`

class cea.tests.test_config.TestConfiguration (methodName='runTest')
    Bases: unittest.case.TestCase

test_applying_parameters()
test_can_be_pickled()
test_changing_scenario()
test_pickling_parameters()
test_update_parameter_value()
test_update_parameter_values_after_pickling()

cea.tests.test_dbf module

Test the utilities/dbf.py file

class cea.tests.test_dbf.TestDbf (methodName='runTest')
    Bases: unittest.case.TestCase

test_roundtrip()
    Make sure the roundtrip df -> dbf -> df keeps the data intact.

cea.tests.test_inputlocator module

class cea.tests.test_inputlocator.TestInputLocator (methodName='runTest')
    Bases: unittest.case.TestCase

classmethod setUpClass()
    Hook method for setting up class fixture before running tests in the class.

test_get_archetypes_properties()
test_get_life_cycle_inventory_building_systems()
test_get_supply_systems_cost()
test_weather()
test_weather_names()
cea.tests.test_plots module

```python
class cea.tests.test_plots.TestCategories (methodName='runTest')
    Bases: unittest.case.TestCase

    test_category_names_in_plots()
    Test to make sure each plot defines the category_name attribute and that it is the same as the category the plot is defined in.
```

cea.tests.test_schedules module

This module contains unit tests for the schedules used by the CEA. The schedule code is tested against data in the file `test_schedules.config` that can be created by running this file. Note, however, that this will overwrite the test data - you should only do this if you are sure that the new data is correct.

```python
class cea.tests.test_schedules.TestBuildingPreprocessing (methodName='runTest')
    Bases: unittest.case.TestCase

    test_mixed_use_archetype_values()

class cea.tests.test_schedules.TestScheduleCreation (methodName='runTest')
    Bases: unittest.case.TestCase

    test_mixed_use_schedules()
```

cea.tests.test_schedules.calculate_mixed_use_archetype_values_results (locator)
    calculate the results for the test - refactored, so we can also use it to write the results to the config file.

cea.tests.test_schedules.create_data()
    Create test data to compare against - run this the first time you make changes that affect the results. Note, this will overwrite the previous test data.

cea.tests.test_schedules.get_test_config_path()
    return the path to the test data configuration file (cea/tests/test_schedules.config)

cea.tests.trace_inputlocator module

Trace the InputLocator calls in a selection of scripts.

```python
cea.tests.trace_inputlocator.create_trace_function (results_set)
    results_set is a set of tuples (locator, filename)

cea.tests.trace_inputlocator.get_csv_schema (filename)

cea.tests.trace_inputlocator.get_dbf_schema (filename)

cea.tests.trace_inputlocator.get_epw_schema (filename)

cea.tests.trace_inputlocator.get_html_schema (_)
    We don’t need to keep a schema of html files - these are outputs anyway

cea.tests.trace_inputlocator.get_json_schema (filename)

cea.tests.trace_inputlocator.get_meta (df_series, attribute_name)

cea.tests.trace_inputlocator.get_shp_schema (filename)

cea.tests.trace_inputlocator.get_tif_schema (filename)

cea.tests.trace_inputlocator.get_xls_schema (filename)
```
cea.tests.trace_inputlocator.is_date(data)
cea.tests.trace_inputlocator.main(config)
cea.tests.trace_inputlocator.meta_to_yaml(config, trace_data, meta_output_file)
cea.tests.trace_inputlocator.replace_repetitive_attr(attr)

cea.utilities package

cea.utilities.remap(x, in_min, in_max, out_min, out_max)
   Scale x from range [in_min, in_max] to [out_min, out_max] Based on this StackOverflow answer: https://stackoverflow.com/a/43567380/2260

Submodules

cea.utilities.color_fader module

cea.utilities.compile_pyd_files module

Compile the .pyd files using Numba pycc to speed up the calculation of certain modules. Currently used for:
   • calc_tm.pyd (used in demand/sensible_loads.py)
   • calc_radiator.pyd (used in technologies/radiators.py)
In order to run this script, you will need to install Numba. Try: conda install numba
cea.utilities.compile_pyd_files.compile_radiators()
cea.utilities.compile_pyd_files.compile_rc_model_sia()
cea.utilities.compile_pyd_files.compile_storagetank()
cea.utilities.compile_pyd_files.copy_pyd(source, destination)
cea.utilities.compile_pyd_files.delete_pyd(*pathspec)
   Delete the file with the pathspec. pathspec is an array of path segments.
cea.utilities.compile_pyd_files.main()

cea.utilities.date module

cea.utilities.date.get_dates_from_year(year)
   creates date range for the year of the calculation :param year: year of first row in weather file :type year: int
   :return: pd.date_range with 8760 values :rtype: pandas.data_range

cea.utilities.dbf module

cea.utilities.dbf.dataframe_to_dbf(df, dbf_path, specs=None)
   Given a pandas Dataframe, write a dbase database to dbf_path.

Parameters specs(list{tuple(basestring, int, int)}) – A list of column specifications for the dbase table. Each column is specified by a tuple (datatype, size, decimal) - we support datatype in ('N', 'C') for strings, integers and floating point numbers, if no specs are provided (see TYPE_MAPPING)
cea.utilities.dbf.\texttt{dbf\_to\_dataframe}(\texttt{dbf\_path}, index=None, cols=False, include_index=False)

cea.utilities.dbf.\texttt{dbf\_to\_xls}(\texttt{input\_file}, output\_path, output\_file\_name)

cea.utilities.dbf.\texttt{main}(\texttt{config})

cea.utilities.dbf.\texttt{xls\_to\_dbf}(\texttt{input\_file}, output\_path, output\_file\_name)

\textbf{cea.utilities.doc\_glossary module}

doc\_glossary.py

Builds input\_files.rst and output\_files.rst using a jinja 2 template located in docs/templates. Both input\_files.rst and output\_files.rst are referenced by glossary.rst.

cea.utilities.doc\_glossary.\texttt{main}(\_)

\textbf{cea.utilities.doc\_graphviz module}

doc\_graphviz.py

Creates the graphviz output used to visualize script dependencies. This file relies on the schema.yml to create the graphviz plots.

cea.utilities.doc\_graphviz.\texttt{create\_graphviz\_files}(\texttt{graphviz\_data}, documentation\_dir)

cea.utilities.doc\_graphviz.\texttt{get\_list\_of\_digraphs}(documentation\_dir, schema\_scripts)

cea.utilities.doc\_graphviz.\texttt{main}(\_)

\textbf{cea.utilities.doc\_html module}

doc\_html.py

This script performs the following:

- Cross references the api documentation, building new files and deleting outdated ones.
- Runs a sphinx html build from the docs directory via the docs make.bat
- Opens the html files of the corresponding change files from Gitdiff (not yet functional)

cea.utilities.doc\_html.\texttt{get\_all\_module\_rsts}(cea\_path)

cea.utilities.doc\_html.\texttt{main}(\_)

cea.utilities.doc\_html.\texttt{preview\_files}(documentation\_dir)

This method performs a Gitdiff, storing the documentation relevant change files as a set.

cea.utilities.doc\_html.\texttt{rebuild\_altered\_module\_documentation}(documentation\_dir)

This method deletes any old api documentation; rebuilding relevant cea.modules.rst files.

\textbf{cea.utilities.doc\_update\_naming module}

doc\_update\_naming.py

Rebuilds all restructured text files for the api documentation Returns a dict containing potentially outdated files and ones yet to be documented.

cea.utilities.doc\_update\_naming.\texttt{main}(\_)

9.1. cea package
**cea.utilities.epwreader module**

Energyplus file reader

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

`cea.utilities.epwreader.calc_wetbulb(Tdrybulb, RH)`

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

`cea.utilities.epwreader.main(config)`

**cea.utilities.latin_hypercube module**

This code was originally published by the following individuals for use with Scilab:

Copyright (C) 2012 - 2013 - Michael Baudin Copyright (C) 2012 - Maria Christopoulou Copyright (C) 2010 - 2011 - INRIA - Michael Baudin Copyright (C) 2009 - Yann Collette Copyright (C) 2009 - CEA - Jean-Marc Martinez

website: forge.scilab.org/index.php/p/scidoe/sourcetree/master/macros

Much thanks goes to these individuals. It has been converted to Python by Abraham Lee.

Copyright (c) 2014, Abraham D. Lee All rights reserved. BSD 3-clause “New” or “Revised” License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

`cea.utilities.latin_hypercube.lhs(n, samples=None, criterion=None, iterations=None)`

Generate a latin-hypercube design

**Parameters**

- **n** [int] The number of factors to generate samples for

**Optional**

- **samples** [int] The number of samples to generate for each factor (Default: n)
criterion [str] Allowable values are “center” or “c”, “maximin” or “m”, “centermaximin” or “cm”, and “correlation” or “corr”. If no value given, the design is simply randomized.

iterations [int] The number of iterations in the maximin and correlations algorithms (Default: 5).

Returns
H [2d-array] An n-by-samples design matrix that has been normalized so factor values are uniformly spaced between zero and one.

Example
A 3-factor design (defaults to 3 samples):
```python
>>> lhs(3)
array([[ 0.40069325, 0.08118402, 0.69763298],
       [ 0.19524568, 0.41383587, 0.29947106],
       [ 0.85341601, 0.75460699, 0.360024 ]])
```

A 4-factor design with 6 samples:
```python
>>> lhs(4, samples=6)
array([[ 0.27226812, 0.02811327, 0.62792445, 0.91988196],
       [ 0.76945538, 0.43501682, 0.01107457, 0.09583358],
       [ 0.45702981, 0.76073773, 0.90245401, 0.18773015],
       [ 0.99342115, 0.85814198, 0.16996665, 0.65069309],
       [ 0.63092013, 0.22148567, 0.33616859, 0.3632478],
       [ 0.05276917, 0.5819198 , 0.67194243, 0.78703262]])
```

A 2-factor design with 5 centered samples:
```python
>>> lhs(2, samples=5, criterion='center')
array([[ 0.3, 0.5],
       [ 0.7, 0.9],
       [ 0.1, 0.3],
       [ 0.9, 0.1],
       [ 0.5, 0.7]])
```

A 3-factor design with 4 samples where the minimum distance between all samples has been maximized:
```python
>>> lhs(3, samples=4, criterion='maximin')
array([[ 0.02642564, 0.55576963, 0.50261649],
       [ 0.51606589, 0.88933259, 0.34040838],
       [ 0.98431735, 0.0380364 , 0.01621717],
       [ 0.40414671, 0.33339132, 0.84845707]])
```

A 4-factor design with 5 samples where the samples are as uncorrelated as possible (within 10 iterations):
```python
>>> lhs(4, samples=5, criterion='correlate', iterations=10)
```

cea.utilities.parallel module

Standardizes multiprocessing use. In the CEA, some functions are run using the standard multiprocessing library. They are run by map'ing the function to a list of arguments (see `multiprocessing.Pool.map_async) and waiting for the processes to finish, while at the same time piping STDOUT, STDERR through cea.utilities.workerstream.QueueWorkerStream - this ensures that the dashboard interface can read the output from the sub-processes.
The way this was done in CEA < v2.23 included boiler plate code that needed to be repeated every time multiprocessing was used. Issue [#2344](https://github.com/architecture-building-systems/CityEnergyAnalyst/issues/2344) was a result of not applying this technique to the demand script.

This module exports the function `map` which is intended to replace both `map_async` and the builtin `map` function (which was used when `config.multiprocessing == False`). This simplifies multiprocessing.

```python
cea.utilities.parallel.__apply_func_with_worker_stream(args)
    Call func, using queue to redirect stdout and stderr, with a tuple of args because multiprocessing.Pool.map only accepts one argument for the function.
    This function is called _inside_ a separate process.
```

```python
cea.utilities.parallel.__multiprocess_wrapper(func, processes, on_complete)
    Create a worker pool to map the function, taking care to set upSTDOUT and STDERR
```

```python
cea.utilities.parallel.single_process_wrapper(func, on_complete)
    The simplest form of vectorization: Just loop
```

```python
cea.utilities.parallel.test(a, b)
```

```python
cea.utilities.parallel.vectorize(func, processes=1, on_complete=None)
    Similar to numpy.vectorize, this function wraps `func` so that it operates on sequences (of same length) of inputs and outputs a sequence of results, similar to `map(func, *args)`.
    The main point of using `vectorize` is to unify single-processing with multi-processing - if processes > 1, then multiprocessing is used and the function will be run on a pool of processes. STDOUT and STDERR of these processes are fed through a `cea.workerstream.QueueWorkerStream` so it can be shown in the dashboard job output.
    The parameter `on_complete` is an optional callable that is called for each completed call of `func`. It takes 4 arguments:
    • i: the 0-based order in which this call was completed
    • n: the total number of function calls to be made
    • args: the arguments passed to this call to `func`
    • result: the return value of this call to `func`

    Parameters
    • `func` – The function to vectorize
    • `processes` (int) – The number of processes to use (use `config.get_number_of_processes()`)
    • `on_complete` – An optional function to call for each completed call to `func`.
```

### cea.utilities.physics module

Physical functions

```python
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.rename_building module

A simple CEA script that renames a building in the input files - NOTE: you’ll have to re-run the simulation and analysis scripts to get the changes as only the files defined in inputs.yml (the files you see in the CEA Dashboard input editor) are changed.

This is the script behind `cea rename-building --old <building> --new <building>`

```python
cea.utilities.rename_building.main(config)
```

```python
cea.utilities.rename_building.rename_dbf_file(path, pk, old, new)
```

```python
cea.utilities.rename_building.rename_shp_file(path, pk, old, new)
```

cea.utilities.reporting module

Functions for Report generation

```python
cea.utilities.reporting.full_report_to_xls(tsd, output_folder, basename)
```

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

```python
cea.utilities.reporting.quick_visualization_tsd(tsd, output_folder, basename)
```

cea.utilities.schedule_reader module

```python
cea.utilities.schedule_reader.main(config)
```

```python
cea.utilities.schedule_reader.read_cea_schedule(path_to_cea_schedule)
```

reader of schedule file .ceaschedule :param path: :return:

```python
cea.utilities.schedule_reader.save_cea_schedule(schedule_data, schedule_complementary_data, path_to_building_schedule)
```

cea.utilities.solar_equations module

solar equations

```python
class cea.utilities.solar_equations.SunProperties(g, Sz, Az, ha, trr_mean, worst_sh, worst_Az)
```

Bases: tuple

```python
Az
```

Alias for field number 2

```python
Sz
```

Alias for field number 1

```python
__getnewargs__()
```

Return self as a plain tuple. Used by copy and pickle.

```python
__getstate__()
```

Exclude the OrderedDict from pickling

```python
static __new__(_cls, g, Sz, Az, ha, trr_mean, worst_sh, worst_Az)
```

Create new instance of SunProperties(g, Sz, Az, ha, trr_mean, worst_sh, worst_Az)
__repr__

Return a nicely formatted representation string

__slots__ = ()

_asdict()

Return a new OrderedDict which maps field names to their values

_fields = ('g', 'Sz', 'Az', 'ha', 'trr_mean', 'worst_sh', 'worst_Az')

classmethod _make(iterable, new=<built-in method __new__ of type object>, len=<built-in function len>)

Make a new SunProperties object from a sequence or iterable

_replace(**kwds)

Return a new SunProperties object replacing specified fields with new values

g

Alias for field number 0

ha

Alias for field number 3

trr_mean

Alias for field number 4

worst_Az

Alias for field number 6

worst_sh

Alias for field number 5

toa.utilities.solar_equations._ephem_setup(latitude, longitude, altitude, pressure, temperature)

toa.utilities.solar_equations.cal_radiation_type(group, hourly_radiation, weather_data)

toa.utilities.solar_equations.calc_angle_of_incidence(g, lat, ha, tilt, teta_z)

To calculate angle of incidence from solar vector and surface normal vector. (Validated with Sandia pvlib.irradiance.aoi)

Parameters

- lat (float) – latitude of the location of case study [radians]
- g (float) – declination of the solar position [radians]
- ha (float) – hour angle [radians]
- tilt (float) – panel surface tilt angle [radians]
- teta_z (float) – panel surface azimuth angle [radians]

Return teta_B angle of incidence [radians]

Rtype teta_B float

toa.utilities.solar_equations.calc_categoriesroof(teta_z, B, GB, Max_Isol)

To categorize solar panels by the surface azimuth, tilt angle and yearly radiation.

Parameters

- teta_z (float) – surface azimuth [degree], 0 degree north (east positive, west negative)
- B (float) – solar panel tile angle [degree]
- GB (float) – yearly radiation of sensors [Wh/m2/year]
• **Max_Isol** (*float*) – maximum radiation received on surfaces [Wh/m²/year]

Return CATteta_z category of surface azimuth

Rtype CATteta_z float

Return CATB category of tilt angle

Rtype CATB float

Return CATBG category of yearly radiation

Rtype CATBG float

`cea.utilities.solar_equations.calc_datetime_local_from_weather_file(weather_data, latitude, longitude)`

`cea.utilities.solar_equations.calc_groups(radiation_of_sensors_clean, sensors_metadata_cat)`

To calculate the mean hourly radiation of sensors in each group.

Parameters

• *radiation_of_sensors_clean* (*dataframe*) – radiation data of the filtered sensors

• *sensors_metadata_cat* (*dataframe*) – data of filtered sensor points categorized with module tilt angle, array spacing, surface azimuth, installed PV module area of each sensor point

Return number_groups number of groups of sensor points

Rtype number_groups float

Return hourlydata_groups mean hourly radiation of sensors in each group

Rtype hourlydata_groups dataframe

Return number_points number of sensor points in each group

Rtype number_points array

Return prop_observers values of sensor properties of each group of sensors

Rtype prop_observers dataframe

`cea.utilities.solar_equations.calc_incident_angle_beam(g, lat, ha, tilt, teta_z)`

`cea.utilities.solar_equations.calc_optimal_angle(teta_z, latitude, transmissivity)`

To calculate the optimal tilt angle of the solar panels.

Parameters

• *teta_z* (*float*) – surface azimuth, 0 degree south (east negative) or 0 degree north (east positive)

• *latitude* (*float*) – latitude of the case study site

• *transmissivity* (*float*) – clearness index [-]

Return abs(b) optimal tilt angle [radians]

Rtype abs(b) float

To calculate the optimal spacing between each panel to avoid shading.

**Parameters**

- **sun_properties** (*SunProperties*) – SunProperties, using worst_sh (Solar elevation at the worst hour [degree]) and worst_Az (Solar Azimuth [degree] at the worst hour)
- **tilt_angle** (*float*) – optimal tilt angle for panels on flat surfaces [degree]
- **module_length** (*float*) – [m]

**Return** D, optimal distance in [m]

**Rtype** D float

Calculate surface azimuth from the surface normal vector \((x, y, z)\) and tilt angle \((B)\). Following the geological sign convention, an azimuth of 0 and 360 degree represents north, 90 degree is east.

**Parameters**

- **xdir** (*float*) – surface normal vector \(x\) in \((x, y, z)\) representing east-west direction
- **ydir** (*float*) – surface normal vector \(y\) in \((x, y, z)\) representing north-south direction
- **B** (*float*) – surface tilt angle in degree

**Returns** surface azimuth, the azimuth of the surface of a solar panel in degree

**Rtype** surface_azimuth float


**Parameters**

- **latitude** (*float*) – latitude of the site [degree]
- **weather_data** (*pd.dataframe*) – weather data of the site
- **solar_window_solstice** (*float*) – the desired hour of shade-free solar window on the winter solstice.

**Return** worst_hour, the hour to calculate minimum spacing

**Rtype** worst_hour float

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)

¹ [http://pysolar.org/](http://pysolar.org/)
• DEC – float, The declination of the Sun

cea.utilities.solar_equations.filter_low_potential(radiation_json_path, metadata_csv_path, config)

To filter the sensor points/hours with low radiation potential.

1. keep sensors above min radiation
2. eliminate points when hourly production < 50 W/m²
3. augment the solar radiation due to differences between panel reflectance and original reflectances used in daysim

Parameters

• radiation_csv (csv) – solar insulation data on all surfaces of each building
• metadata_csv (csv) – solar insulation sensor data of each building

Return max_annual_radiation yearly horizontal radiation [Wh/m²/year]

Rtype max_annual_radiation float

Return annual_radiation_threshold minimum yearly radiation threshold for sensor selection [Wh/m²/year]

Rtype annual_radiation_threshold float

Return sensors_rad_clean radiation data of the filtered sensors [Wh/m²]

Rtype sensors_rad_clean dataframe

Return sensors_metadata_clean data of filtered sensor points measuring solar insulation of each building

Rtype sensors_metadata_clean dataframe

Following assumptions are made:

1. Sensor points with low yearly radiation are deleted. The threshold (minimum yearly radiation) is a percentage of global horizontal radiation. The percentage threshold (min_radiation) is a global variable defined by users.
2. For each sensor point kept, the radiation value is set to zero when radiation value is below 50 W/m².
3. No solar panels on windows.

cea.utilities.solar_equations.get_equation_of_time(day_date)

cea.utilities.solar_equations.get_hour_angle(longitude_deg, min_date, hour_date, day_date)

cea.utilities.solar_equations.get_local_etc_timezone(latitude, longitude)

This function gets the time zone at a given latitude and longitude in ‘Etc/GMT’ format. This time zone format is used in order to avoid issues caused by Daylight Saving Time (DST) (i.e., redundant or missing times in regions that use DST). However, note that ‘Etc/GMT’ uses a counter intuitive sign convention, where West of GMT is POSITIVE, not negative. So, for example, the time zone for Zurich will be returned as ‘Etc/GMT-1’.

Parameters

• latitude – Latitude at the project location
• longitude – Longitude at the project location
**cea.utilities.solar_equations.get_solar_time**

*long, min_date, hour_date, day_date*

returns solar time in hours for the specified longitude and time, accurate only to the nearest minute. longitude_deg min_date hour_date day_date

**cea.utilities.solar_equations.optimal_angle_and_tilt**

*sensors_metadata_clean, latitude, solar_properties, max_rad_Wperm2yr, panel_properties*

This function first determines the optimal tilt angle, row spacing and surface azimuth of panels installed at each sensor point. Secondly, the installed PV module areas at each sensor point are calculated. Lastly, all the modules are categorized with its surface azimuth, tilt angle, and yearly radiation. The output will then be used to calculate the absorbed radiation.

**Parameters**

- **sensors_metadata_clean** *(dataframe)* – data of filtered sensor points measuring solar insulation of each building
- **latitude** *(float)* – latitude of the case study location
- **solar_properties** *(cea.utilities.solar_equations.SunProperties)* – A SunProperties, using worst_sh: solar elevation at the worst hour [degree], worst_Az: solar azimuth at the worst hour [degree] and trr_mean: transmissivity / clearness index [-]
- **module_length_m** *(float)* – length of the PV module [m]
- **max_rad_Wperm2yr** *(float)* – max radiation received on surfaces [Wh/m2/year]

**Returns**

**sensors_metadata_clean** data of filtered sensor points categorized with module tilt angle, array spacing, surface azimuth, installed PV module area of each sensor point and the categories

**Rtype** **sensors_metadata_clean** dataframe

**Assumptions:**

1. **Tilt angle:** If the sensor is on tilted roof, the panel will have the same tilt as the roof. If the sensor is on a wall, the tilt angle is 90 degree. Tilt angles for flat roof is determined using the method from Quinn et al.
2. **Row spacing:** Determine the row spacing by minimizing the shadow according to the solar elevation and azimuth at the worst hour of the year. The worst hour is a global variable defined by users.
3. **Surface azimuth (orientation) of panels:** If the sensor is on a tilted roof, the orientation of the panel is the same as the roof. Sensors on flat roofs are all south facing.

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

**cea.utilities.standardize_coordinates module**

- **cea.utilities.standardize_coordinates.ensure_cpg_file** *(shapefile_path)*
- **cea.utilities.standardize_coordinates.get_geographic_coordinate_system**()
- **cea.utilities.standardize_coordinates.get_lat_lon_projected_shapefile** *(data)*
- **cea.utilities.standardize_coordinates.get_projected_coordinate_system** *(lat, lon)*
- **cea.utilities.standardize_coordinates.raster_to_WSG_and_UTM** *(raster_path, lat, lon)*
- **cea.utilities.standardize_coordinates.shapefile_to_WSG_and_UTM** *(shapefile_path)*
### cea.utilities.workerstream module

This file implements `WorkerStream` for capturing stdout and stderr.

```python
class cea.utilities.workerstream.HttpWorkerStream(name, jobid, url)
    Bases: object
    __init__(name, jobid, url)
    x.__init__(...) initializes x; see help(type(x)) for signature

class cea.utilities.workerstream.QueueWorkerStream(name, queue)
    Bases: object
    File-like object for wrapping the output of the scripts with queues - to be created in child process
    __init__(name, queue)
    x.__init__(...) initializes x; see help(type(x)) for signature
    __repr__() <==> repr(x)
    close()
    flush()
    isatty()
    write(str)

class cea.utilities.workerstream.WorkerStream(name, connection)
    Bases: object
    File-like object for wrapping the output of the scripts into connection messages
    __init__(name, connection)
    x.__init__(...) initializes x; see help(type(x)) for signature
    __repr__() <==> repr(x)
    close()
    flush()
    isatty()
    write(str)
```

### cea.utilities.yaml_ordered_dict module

```python
class cea.utilities.yaml_ordered_dict.OrderedDictYAMLLoader(*args, **kwargs)
    Bases: yaml.loader.Loader
    A YAML loader that loads mappings into ordered dictionaries.
    __init__(*args, **kwargs)
    Initialize the scanner.
    construct_mapping(node, deep=False)
    construct_yaml_map(node)
```

9.1. cea package
**cea.workflows package**

**Submodules**

**cea.workflows.workflow module**

Run a workflow.yml file - this is like a cea-aware “batch” file for running multiple cea scripts including parameters. The `cea workflow` can also pick up from previous (failed?) runs, which can help in debugging.

- `do_config_step(config, step)`
  - update the :py:class:`cea.config.Configuration` object, returning the new value for the config

- `do_script_step(config, step)`
  - Run a script based on the step’s “script” and “parameters” (optional) keys.

- `main(config)`

- `read_resume_info(resume_yml, workflow_yml)`

- `run(config, script, **kwargs)`

- `set_parameter(config, parameter, value)`
  - Set a parameter to a value (expand with environment vars) without tripping the restricted_to part of config

- `set_up_environment_variables(config)`
  - create some environment variables to be used when configuring stuff. This includes the variable `NOW`, plus one variable for each config parameter, named “CEA_{SECTION}_{PARAMETER}”.

  - This is useful for referring to the “user” config, when basing a workflow off the default config.

**9.1.2 Submodules**

**9.1.3 cea.api module**

Provide access to the scripts exported by the City Energy Analyst.

- `compile(config=<cea.config.Configuration object>, **kwargs)`
  - Compile the .pyd files using Numba pycc to speed up the calculation of certain modules. Currently used for:
    - `calc_tm.pyd` (used in demand/sensible_loads.py)
    - `calc_radiator.pyd` (used in technologies/radiators.py)

  - In order to run this script, you will need to install Numba. Try: `conda install numba`

- `dashboard(config=<cea.config.Configuration object>, **kwargs)`
  - FIXME: Add API documentation to cea.interfaces.dashboard.dashboard

- `data_helper(config=<cea.config.Configuration object>, **kwargs)`
  - building properties algorithm

- `dbf_to_excel_to_dbf(config=<cea.config.Configuration object>, **kwargs)`
  - FIXME: Add API documentation to cea.utilities.dbf

- `decentralized(config=<cea.config.Configuration object>, **kwargs)`
  - Disconnected buildings

  - This computes the close-to-optimal supply system for single buildings.

- `demand(config=<cea.config.Configuration object>, **kwargs)`
  - Analytical energy demand model algorithm
cea.api.emissions (config=<cea.config.Configuration object>, **kwargs)
Emissions analysis (LCA) This script is used to calculate the LCA

cea.api.excel_to_shapefile (config=<cea.config.Configuration object>, **kwargs)
Implements the CEA script excel-to-shapefile
Similar to how excel-to-dbf takes a dBase database file (example.dbf) and converts that to Excel format, this does the same with a Shapefile.
It uses the geopandas.GeoDataFrame class to read in the shapefile. The geometry column is serialized to a nested list of coordinates using the JSON notation.

cea.api.extract_reference_case (config=<cea.config.Configuration object>, **kwargs)
Extract the reference case (cea/examples/reference-case-open.zip).

cea.api.glossary (config=<cea.config.Configuration object>, **kwargs)
doc_glossary.py
Builds input_files.rst and output_files.rst using a jinja 2 template located in docs/templates. Both input_files.rst and output_files.rst are referenced by glossary.rst.

cea.api.graphviz (config=<cea.config.Configuration object>, **kwargs)
doc_graphviz.py
Creates the graphviz output used to visualize script dependencies. This file relies on the schema.yml to create the graphviz plots.

cea.api.html (config=<cea.config.Configuration object>, **kwargs)
doc_html.py
This script performs the following:
- Cross references the api documentation, building new files and deleting outdated ones.
- Runs a sphinx html build from the docs directory via the docs make.bat
- Opens the html files of the corresponding change files from Gitdiff (not yet functional)

cea.api.install_arcgis (config=<cea.config.Configuration object>, **kwargs)
Install the toolbox into ArcGIS Desktop 10.4 and 10.5

cea.api.install_grasshopper (config=<cea.config.Configuration object>, **kwargs)
Install the grasshopper interface. This assumes that the python path for grasshopper Python scripts is in %APPDATA%\McNeel\Rhinoceros.0\scripts.

cea.api.list_demand_graphs_fields (config=<cea.config.Configuration object>, **kwargs)
List the fields that can be used for the demand-graphs --analysis-fields parameter given a scenario

cea.api.multi_criteria_analysis (config=<cea.config.Configuration object>, **kwargs)
Multi criteria decision analysis

cea.api.network_layout (config=<cea.config.Configuration object>, **kwargs)
FIXME: Add API documentation to cea.technologies.network_layout.main

cea.api.operation_costs (config=<cea.config.Configuration object>, **kwargs)
costs according to supply systems

cea.api.optimization (config=<cea.config.Configuration object>, **kwargs)
multi-objective optimization of supply systems for the CEA

cea.api.photovoltaic (config=<cea.config.Configuration object>, **kwargs)
Photovoltaic

cea.api.photovoltaic_thermal (config=<cea.config.Configuration object>, **kwargs)
Photovoltaic thermal panels

9.1. cea package
cea.api.plots(config=<cea.config.Configuration object>, **kwargs)
This is the dashboard of CEA

cea.api.radiation(config=<cea.config.Configuration object>, **kwargs)
Radiation engine and geometry handler for CEA

cea.api.register_scripts()

cea.api.rename_building(config=<cea.config.Configuration object>, **kwargs)
A simple CEA script that renames a building in the input files - NOTE: you’ll have to re-run the simulation and analysis scripts to get the changes as only the files defined in inputs.yml (the files you see in the CEA Dashboard input editor) are changed.

This is the script behind `cea rename-building --old <building> --new <building>`

cea.api.schedule_maker(config=<cea.config.Configuration object>, **kwargs)
FIXME: Add API documentation to `cea.demand.schedule_maker.schedule_maker`

cea.api.sensitivity_demand_analyze(config=<cea.config.Configuration object>, **kwargs)
Analyze the results in the samples folder and write them out to an Excel file. This script assumes:

- a samples folder with the files `samples.npy` and `problem.pickle` was created with `sensitivity_demand_samples.py`
- all the results have been added to the samples folder in the format `result.%i.csv`, with `%i` replaced by the index into the samples array. Use the script `sensitivity_demand_simulate.py` to create the results.
- each result file has the same list of columns (the --output-parameters for the simulations were the same)
- the `analyze_sensitivity` function is called with the same method and arguments as the sampling routine (`sensitivity_demand_samples.py`).

cea.api.sensitivity_demand_samples(config=<cea.config.Configuration object>, **kwargs)
Create a list of samples in a specified folder as input for the demand sensitivity analysis.

This script creates:

- a samples folder with the files `samples.npy` and `problem.pickle` to be used by the scripts `sensitivity_demand_count.py`, `sensitivity_demand_simulate.py` and `sensitivity_demand_analyze.py`.

The file `samples.npy` is a NumPy array of samples to simulate, as generated by either the morris or the sobol sampler. Each sample is a a row in the array and each row consists of a list of parameter values to use for the simulation. The parameters correspond to the parameter names defined by the `variable_groups` input. `variable_groups` refers to worksheets in the uncertainty db, an Excel file in `cea/databases/CH/Uncertainty/uncertainty_distributions.xls`, which specifies row-by-row the variables to sample and their distribution parameters.

The file `problem.pickle` is a python dictionary that is saved using the standard `pickle` module and contains the following data:

- `num_vars`: int, the number of variables being analyzed.
- `names`: list of str, the variable names in the same order as the values in each sample row. Used to apply the sample values to the input data using the override mechanism. See:
  - `cea.analysis.sensitivity.sensitivity_demand_simulate.apply_sample_parameters` (write overrides)
  - `cea.demand.thermal_loads.BuildingProperties #__init__` (read overrides)
- `bounds`: list of tuple(min, max), lower and upper bounds for each variable to sample (only used by the sampler)
- `groups`: None (currently not used)

cea.api.sensitivity_demand_simulate(config=<cea.config.Configuration object>, **kwargs)
Simulate a single sample or a batch of samples from the samples folder using the demand script.
The script `sensitivity_demand_samples.py` creates a samples folder containing a list of samples stored in the NumPy array `samples.npy`. Each sample is a list of parameter values to set for a list of variables (the names of the variables are stored in the file `problem.pickle` in the samples folder).

This script runs the samples `--sample-index` through `--sample-index + --number-of-samples` and writes the results out to the samples folder as files of the form `results.$i.csv` (with $i$ set to the index into the samples array).

```python
ccea.sewage_potential(config=<cea.config.Configuration object>, **kwargs)
    sewage source heat exchanger
```

```python
ccea.shallow_geothermal_potential(config=<cea.config.Configuration object>, **kwargs)
    FIXME: Add API documentation to ceea.resources.geothermal
```

```python
ccea.shapefile_to_excel(config=<cea.config.Configuration object>, **kwargs)
    Implements the CEA script `shapefile-to-excel`
    Similar to how `dbf-to-excel` takes a dBase database file (example.dbf) and converts that to Excel format, this does the same with a Shapefile.
    It uses the `geopandas.GeoDataFrame` class to read in the shapefile. And serializes the `geometry` column to Excel as well as a serialized list of tuples.
```

```python
ccea.solar_collector(config=<cea.config.Configuration object>, **kwargs)
    solar collectors
```

```python
ccea.streets_helper(config=<cea.config.Configuration object>, **kwargs)
    This script extracts streets from Open street maps
```

```python
ccea.surroundings_helper(config=<cea.config.Configuration object>, **kwargs)
    This script extracts surrounding buildings of the zone geometry from Open street maps
```

```python
ccea.terrain_helper(config=<cea.config.Configuration object>, **kwargs)
    This script extracts terrain elevation from NASA - SRTM [https://www2.jpl.nasa.gov/srtm/](https://www2.jpl.nasa.gov/srtm/)
```

```python
ccea.test(config=<cea.config.Configuration object>, **kwargs)
    “Makefile” for doit to test the whole package with Jenkins.
    This file gets run with the `cea test` command as well as by the Jenkins continuous integration server. It runs all the unit tests in the `cea/tests/` folder as well as some of the CEA scripts, to make sure they at least run through.
    In order to run reference cases besides the one called “open”, you will need to set up authentication to the private GitHub repository. To do this, you need to create a GitHub [authentication token](https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/). Then, create a text file called `cea_github.auth` in your home directory (e.g. `C:\Users\your-user-name` for Windows systems or equivalent on POSIX systems, ask your administrator if you don’t know what this is). The file should contain two lines, the first being your GitHub user name the second the authentication token.
    The reference cases can be found here: [https://github.com/architecture-building-systems/cea-reference-case/archive/master.zip](https://github.com/architecture-building-systems/cea-reference-case/archive/master.zip)
```

```python
ccea.thermal_network(config=<cea.config.Configuration object>, **kwargs)
    Hydraulic - thermal network
```

```python
ccea.thermal_network_optimization(config=<cea.config.Configuration object>, **kwargs)
    hydraulic network
```

```python
ccea.trace_inputlocator(config=<cea.config.Configuration object>, **kwargs)
    Trace the InputLocator calls in a selection of scripts.
```

```python
ccea.update_naming(config=<cea.config.Configuration object>, **kwargs)
    doc_update_naming.py
```
Rebuilds all restructured text files for the api documentation Returns a dict containing potentially outdated files and ones yet to be documented.

\texttt{cea.api.water\_body\_potential}(config=\texttt{cea.config.Configuration object}, **kwargs)

Sewage source heat exchanger

\texttt{cea.api.weather\_helper}(config=\texttt{cea.config.Configuration object}, **kwargs)

The weather-helper script sets the weather data used (\texttt{inputs/weather.epw}) for simulations.

\texttt{cea.api.workflow}(config=\texttt{cea.config.Configuration object}, **kwargs)

Run a workflow.yml file - this is like a cea-aware “batch” file for running multiple cea scripts including parameters. \texttt{cea workflow} can also pick up from previous (failed?) runs, which can help in debugging.

\texttt{cea.api.zone\_helper}(config=\texttt{cea.config.Configuration object}, **kwargs)

This is a template script - an example of how a CEA script should be set up. NOTE: ADD YOUR SCRIPT’S DOCUMENTATION HERE (what, why, include literature references)

\subsection{9.1.4 \texttt{cea.config} module}

Manage configuration information for the CEA. The Configuration class is built dynamically based on the type information in \texttt{default.config}.

\texttt{class cea.config.BooleanParameter(name, section, config)}

Bases: \texttt{cea.config.Parameter}

Read / write boolean parameters to the config file.

\texttt{._boolean\_states = \{'0': False, '1': True, 'false': False, 'no': False, 'off': False, 'true': True, 'yes': True}}

\texttt{decode(value)}

Decode value to the type supported by this Parameter

\texttt{encode(value)}

Encode value to a string representation for writing to the configuration file

\texttt{typename = 'BooleanParameter'}

\texttt{class cea.config.BuildingsParameter(name, section, config)}

Bases: \texttt{cea.config.MultiChoiceParameter}

A list of buildings in the zone

\texttt{._choices}

\texttt{initialize(parser)}

Override this function to initialize a parameter with values as read from the default.config

\texttt{typename = 'BuildingsParameter'}

\texttt{class cea.config.ChoiceParameter(name, section, config)}

Bases: \texttt{cea.config.Parameter}

A parameter that can only take on values from a specific set of values

\texttt{decode(value)}

Decode value to the type supported by this Parameter

\texttt{encode(value)}

Encode value to a string representation for writing to the configuration file

\texttt{initialize(parser)}

Override this function to initialize a parameter with values as read from the default.config

\texttt{typename = 'ChoiceParameter'}
class cea.config.Configuration(config_file='/home/docs/cea.config')
    Bases: object

    _getattr_(item)
    Return either a Section object or the value of a Parameter

    _getstate__()
    when we pickle, we only really need to pickle the user_config

    _init__(config_file='/home/docs/cea.config')
    x.__init__(...) initializes x; see help(type(x)) for signature

    _setattr__(key, value)
    Set the value on a parameter in the general section

    __setstate__(state)
    read in the user_config and re-initialize the state (this basically follows the __init__)

    apply_command_line_args(args, option_list)
    Apply the command line args as passed to cea.interfaces.cli.cli (the cea command). Each argument is assumed to follow this pattern: --PARAMETER-NAME VALUE, with PARAMETER-NAME being one of the options in the config file and VALUE being the value to override that option with.

    get(fqname)
    Given a string of the form “section:parameter”, return the value of that parameter

    get_number_of_processes()
    Returns the number of processes to use for multiprocessing. :param config: Configuration file. :return number_of_processes: Number of processes to use.

    get_parameter(fqname)
    Given a string of the form “section:parameter”, return the parameter object

    ignore_restrictions()
    Create a with block where the config file restrictions are not kept. Usage:

    with config.ignore_restrictions():
        config.my_section.my_property = value

    matching_parameters(option_list)
    Return a tuple (Section, Parameter) for all parameters that match the parameters in the option_list.
    option_list is a sequence of parameter names in the form section[:parameter] if only a section is mentioned, all the parameters of that section are added. Otherwise, only the specified parameter is added to the resulting list.

    restrict_to(option_list)
    Restrict the config object to only allowing parameters as defined in the option_list parameter. option_list is a list of strings of the form section or section:parameter as used in the scripts.yml file for the parameters attribute of a script.

    The purpose of this is to ensure that scripts don’t use parameters that are not specified as options to the scripts. This only solves half of the possible issues with cea.config.Configuration: the other is that a script creates its own config file somewhere down the line. This is hard to check anyway.

    save(config_file='/home/docs/cea.config')
    Save the current configuration to a file. By default, the configuration is saved to the user configuration file (~/.cea.config). If config_file is set to the default configuration file cea.config. DEFAULT_CONFIG, then nothing is saved - this is to prevent overwriting the default configuration file.

class cea.config.DateParameter(name, section, config)
    Bases: cea.config.Parameter
Decode value to the type supported by this Parameter

Encode value to a string representation for writing to the configuration file

typename = 'DateParameter'

class cea.config.FileParameter (name, section, config)
    Bases: cea.config.Parameter

Describes a file in the system.

Decode value to the type supported by this Parameter

Encode value to a string representation for writing to the configuration file

initialize (parser)
    Override this function to initialize a parameter with values as read from the default.config

typename = 'FileParameter'

class cea.config.IntegerParameter (name, section, config)
    Bases: cea.config.Parameter

Read / write integer parameters to the config file.

Decode value to the type supported by this Parameter

Encode value to a string representation for writing to the configuration file

initialize (parser)
    Override this function to initialize a parameter with values as read from the default.config

typename = 'IntegerParameter'

class cea.config.JsonParameter (name, section, config)
    Bases: cea.config.Parameter

A parameter that gets / sets JSON data (useful for dictionaries, lists etc.)

Decode value to the type supported by this Parameter

Encode value to a string representation for writing to the configuration file

typename = 'JsonParameter'

class cea.config.ListParameter (name, section, config)
    Bases: cea.config.Parameter

A parameter that is a list of comma-separated strings. An error is raised when writing strings that contain commas themselves.

Decode value to the type supported by this Parameter

Encode value to a string representation for writing to the configuration file

typename = 'ListParameter'
class `cea.config.MultiChoiceParameter` *(name, section, config)*

    Bases: `cea.config.ChoiceParameter`

Like ChoiceParameter, but multiple values from the choices list can be used.

    `decode`(value)
        Decode value to the type supported by this Parameter

    `encode`(value)
        Encode value to a string representation for writing to the configuration file

typename = 'MultiChoiceParameter'

class `cea.config.OptimizationIndividualListParameter` *(name, section, config)*

    Bases: `cea.config.ListParameter`

    `get_folders`(project=None)

    `initialize`(parser)
        Override this function to initialize a parameter with values as read from the default.config

typename = 'OptimizationIndividualListParameter'

class `cea.config.OptimizationIndividualParameter` *(name, section, config)*

    Bases: `cea.config.Parameter`

    `get_folders`(project=None)

    `get_generations`(scenario, project=None)

    `get_individuals`(scenario, generation, project=None)

    `initialize`(parser)
        Override this function to initialize a parameter with values as read from the default.config

typename = 'OptimizationIndividualParameter'

class `cea.config.Parameter` *(name, section, config)*

    Bases: `object`

    `__init__`(name, section, config)

        Parameters

        • `name` *(str)* – The name of the parameter (as it appears in the configuration file, all lowercase)

        • `section` *(Section)* – The section this parameter is to be defined for

        • `config` *(Configuration)* – The Configuration instance this parameter belongs to

    `__repr__`() <==> `repr(x)`

    `decode`(value)
        Decode value to the type supported by this Parameter

    `default`

    `encode`(value)
        Encode value to a string representation for writing to the configuration file

    `get`()
        Return the value from the config file

    `get_raw`()
        Return the value from the config file, but without replacing references and also without decoding.
initialize(parser)
    Override this function to initialize a parameter with values as read from the default.config

py_name
replace_references(encoded_value)
set(value)

typename = 'Parameter'

class cea.config.PathParameter(name, section, config)
    Bases: cea.config.Parameter
    Describes a folder in the system
    decode(value)
        Always return a canonical path
    initialize(parser)
        Override this function to initialize a parameter with values as read from the default.config
    typename = 'PathParameter'

class cea.config.PlantNodeParameter(name, section, config)
    Bases: cea.config.ChoiceParameter
    A parameter that refers to valid PLANT nodes of a thermal-network
    _choices
    decode(value)
        Decode value to the type supported by this Parameter
    encode(value)
        Allow encoding None, because not all scenarios have a thermal network
    initialize(parser)
        Override this function to initialize a parameter with values as read from the default.config
    typename = 'PlantNodeParameter'

class cea.config.RealParameter(name, section, config)
    Bases: cea.config.Parameter
    Read / write floating point parameters to the config file.
    decode(value)
        Decode value to the type supported by this Parameter
    encode(value)
        Encode value to a string representation for writing to the configuration file
    initialize(parser)
        Override this function to initialize a parameter with values as read from the default.config
    typename = 'RealParameter'

class cea.config.RegionParameter(name, section, config)
    Bases: cea.config.ChoiceParameter
    A parameter that can either be set to a region-specific CEA Database (e.g. CH or SG) or to a user-defined folder that has the same structure.
    _choices
        List the technology database template names available
decode (value)
Return either built-in region name (e.g. CH or SG) OR a full path to the user-supplied template folder

encode (value)
Make sure to use the friendly shorthands (e.g. CH and SG) if possible

initialize (parser)
Override this function to initialize a parameter with values as read from the default.config

is_valid_template (path)
True, if the path is a valid template path - containing the same excel files as the standard regions.

typename = 'RegionParameter'
class cea.config.ScenarioNameParameter (name, section, config)
Bases: cea.config.ChoiceParameter
A parameter that can be set to a scenario-name

_choices
decode (value)
Decode value to the type supported by this Parameter
encode (value)
Make sure the scenario folder exists
initialize (parser)
Override this function to initialize a parameter with values as read from the default.config

typename = 'ScenarioNameParameter'
class cea.config.ScenarioParameter (name, section, config)
Bases: cea.config.Parameter
This parameter type is special in that it is derived from two other parameters (project, scenario-name)
decode (value)
Make sure the path is nicely formatted
get_raw ()
Return the value from the config file, but without replacing references and also without decoding.
set (value)
Update the {general:project} and {general:scenario-name} parameters

typename = 'ScenarioParameter'
class cea.config.Section (name, config)
Bases: object
Instances of Section describe a section in the configuration file.

_getattr_ (item)
Return the value of the parameter with that name.

__init__ (name, config)

Parameters

- name (str) – The name of the section (as it appears in the configuration file, all lowercase)
- config (Configuration) – The Configuration instance this section belongs to

_repr__ () <==> repr(x)
__setattr__ (key, value)
Set the value on a parameter

class cea.config.SingleBuildingParameter (name, section, config)
    Bases: cea.config.ChoiceParameter
    A (single) building in the zone
    _choices
    encode (value)
    Encode value to a string representation for writing to the configuration file
    initialize (parser)
    Override this function to initialize a parameter with values as read from the default.config
typename = 'SingleBuildingParameter'

class cea.config.StringParameter (name, section, config)
    Bases: cea.config.Parameter
typename = 'StringParameter'

class cea.config.SubfoldersParameter (name, section, config)
    Bases: cea.config.ListParameter
    A list of subfolder names of a parent folder.
    decode (value)
    Only return the folders that exist
    get_folders ()
    initialize (parser)
    Override this function to initialize a parameter with values as read from the default.config
typename = 'SubfoldersParameter'

class cea.config.WeatherPathParameter (name, section, config)
    Bases: cea.config.Parameter
    decode (value)
    Decode value to the type supported by this Parameter
default
    override base default, since in decode we’ve banned empty weather file parameters
    initialize (parser)
    Override this function to initialize a parameter with values as read from the default.config
typename = 'WeatherPathParameter'

class cea.config.WorkflowParameter (name, section, config)
    Bases: cea.config.Parameter
    decode (value)
    Decode value to the type supported by this Parameter
examples = {'district-cooling-system': '/home/docs/checkouts/readthedocs.org/user_builds/city-energy-analyst/conda/latest/lib/python2.7/site-packages/cea/workflows/district_heating_system.yml'}
typename = 'WorkflowParameter'

cea.config.config_identifier (python_identifier)
    For vanity, keep keys and section names in the config file with dashes instead of underscores and all-lowercase
**cea.config.construct_parameter** *(parameter_name, section, config)*
Create the appropriate subtype of Parameter based on the .type option in the default.config file.

- **param parameter_name**: The name of the parameter (as it appears in the configuration file, all lowercase)
- **type parameter_name**: str
- **param section**: The section this parameter is to be defined for
- **type section**: Section
- **param config**: The Configuration instance this parameter belongs to
- **type config**: Configuration

**cea.config.parse_command_line_args** *(args)*
Group the arguments into a dictionary: parameter-name -> value

**cea.config.parse_string_to_list** *(line)*
Parse a line in the csv format into a list of strings

### 9.1.5 cea.constants module

This file contains the constants used in many folders in CEA. If few constants are only used in a subfolder, it is highly recommended to keep those constants in a separate file in the subfolder. This is to make sure we declare the constants closest to the point of usage.

### 9.1.6 cea.glossary module

Contains some helper methods for working with glossary.csv

**cea.glossary._path_to_glossary_csv** *

**cea.glossary.read_glossary_df** *
Returns the glossary as a DataFrame

**cea.glossary.read_glossary_dicts** *
Returns the glossary as a list of dicts

### 9.1.7 cea.inputlocator module

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

**class cea.inputlocator.InputLocator** *(scenario)*

**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).

**PVT_metadata_results** *(building_name)*
scenario/outputs/data/potentials/solar/{building_name}_SC_sensors.csv

**PVT_results** *(building_name)*
scenario/outputs/data/potentials/solar/{building_name}_SC.csv

**PVT_total_buildings** *
scenario/outputs/data/potentials/solar/{building_name}_PV.csv

**PVT_totals** *
scenario/outputs/data/potentials/solar/{building_name}_PV.csv

**PV_metadata_results** *(building_name)*
scenario/outputs/data/potentials/solar/{building_name}_PV_sensors.csv

**PV_network** *(network)*
scenario/outputs/data/potentials/solar/{building_name}_PV.csv
PV_results (building_name)
    scenario/outputs/data/potentials/solar/[building_name]_PV.csv

PV_total_buildings()
    scenario/outputs/data/potentials/solar/[building_name]_PV.csv

PV_totals()
    scenario/outputs/data/potentials/solar/[building_name]_PV.csv

SC_metadata_results (building_name, panel_type)
    scenario/outputs/data/potentials/solar/[building_name]_SC_sensors.csv

SC_results (building_name, panel_type)
    scenario/outputs/data/potentials/solar/[building_name]_SC.csv

SC_total_buildings (panel_type)
    scenario/outputs/data/potentials/solar/[building_name]_PV.csv

SC_totals (panel_type)
    scenario/outputs/data/potentials/solar/[building_name]_PV.csv

__init__ (scenario)
    x.__init__(...) initializes x; see help(type(x)) for signature

static _ensure_folder (*components)
    Return the *components joined together as a path to a folder and ensure that that folder exists on disc. If it
doesn’t exist yet, attempt to make it with os.makedirs.

are_equal (path_a, path_b)
    Checks to see if two paths are equal

cHECK_CPG (shapefile_path)

ensure_parent_folder_exists (file_path)
    Use os.makedirs to ensure the folders exist

get_4D_demand_plot (period)
    scenario/outputs/plots/timeseries

get_4D_pv_plot (period)
    scenario/outputs/plots/timeseries

get_4D_pvt_plot (period)
    scenario/outputs/plots/timeseries

get_4D_radiation_plot (period)
    scenario/outputs/plots/timeseries

get_4D_sc_plot (period)
    scenario/outputs/plots/timeseries

get_archetypes_properties()
    Returns the database of construction properties to be used by the data-helper. These are copied to the
scenario if they are not yet present, based on the configured region for the scenario.

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

get_building_air_conditioning()
    scenario/inputs/building-properties/air_conditioning_systems.dbf

get_building_architecture()
    scenario/inputs/building-properties/architecture.dbf This file is generated by the data-helper 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_citygml()
    scenario/outputs/data/solar-radiation/district.gml

get_building_geometry_folder()
    scenario/inputs/building-geometry/

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_properties_folder()
    scenario/inputs/building-properties/

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

get_building_weekly_schedules(building_name)
    scenario/inputs/building-properties/schedules/{building_name}.csv This file contains schedules of occupancy, appliance use, etc of each building. Schedules are 8760 values per year :param building_name:
:return:

get_building_weekly_schedules_folder()
    scenario/inputs/building-properties/schedules/

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

get_calibration_cluster_mcda(generation)

get_calibration_cluster_mcda_folder()

get_calibration_cluster_opt_checkpoint(generation, building)
    scenario/outputs/data/calibration/clustering/checkpoints/

get_calibration_cluster_opt_checkpoint_folder()

get_calibration_clustering_clusters_folder()
    scenario/outputs/data/calibration

get_calibration_clustering_folder()
    scenario/outputs/data/calibration

get_calibration_clustering_plots_folder()

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

get_calibration_folder()
    scenario/outputs/data/calibration

get_calibration_gaussian_emulator(building_name, building_load)
    scenario/outputs/data/calibration

get_calibration_posteriors(building_name, building_load)
    scenario/outputs/data/calibration
get_calibration_problem (building_name, building_load)
    scenario/outputs/data/calibration

get_concept_network_on_streets (ind_num, gen_num)
    scenario/outputs/data/calibration/clustering/checkpoints/…

get_concept_network_plot (ind_num, gen_num)
    scenario/outputs/data/calibration/clustering/checkpoints/…

get_costs_folder ()
    scenario/outputs/data/costs

get_costs_operation_file ()
    scenario/outputs/data/costs/{load}_cost_operation.pdf

get_database_air_conditioning_systems ()

get_database_envelope_systems ()
    databases/Systems/envelope_systems.csv

get_database_lca_buildings ()
    Returns the database of life cycle inventory for buildings systems. These are copied to the scenario if they
    are not yet present, based on the configured region for the scenario.

get_database_lca_mobility ()
    Returns the database of life cycle inventory for supply systems. These are copied to the scenario if they
    are not yet present, based on the configured region for the scenario.

get_database_standard_schedules ()

get_database_standard_schedules_use (path_to_database, use)

get_database_supply_systems ()
    Returns the database of supply systems for cost analysis. These are copied to the scenario if they are not
    yet present, based on the configured region for the scenario.

get_databases_folder ()
    Returns the inputs folder of a scenario

get_daysim_mat ()
    this gets the file that documents all of the radiance/default_materials

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

get_demand_measured_folder ()
    scenario/outputs/data/demand

get_demand_results_file (building_name, format='csv')
    scenario/outputs/data/demand/{building_name}.csv

get_demand_results_folder ()
    scenario/outputs/data/demand

get_docs_folder ()
    Returns docs

get_electric_network_output_location (name)
    scenario/inputs/building-geometry/zone.shp

get_electric_substation_input_location ()
    scenario/inputs/building-geometry/zone.shp
get_electric_substation_output_location()  
scenario/inputs/building-geometry/zone.shp

get_electrical_and_thermal_network_optimization_all_individuals()  
scenario/outputs/data/calibration/clustering/checkpoints/...

get_electrical_and_thermal_network_optimization_checkpoint()  
scenario/outputs/data/calibration/clustering/checkpoints/...

get_electrical_and_thermal_network_optimization_checkpoint_final()  
scenario/outputs/data/calibration/clustering/checkpoints/...

get_electrical_and_thermal_network_optimization_checkpoint_initial()  
scenario/outputs/data/calibration/clustering/checkpoints/...

get_electrical_and_thermal_network_optimization_individuals_in_generation()  
scenario/outputs/data/calibration/clustering/checkpoints/...

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

get_electrical_and_thermal_network_optimization_results_folder()  
scenario/outputs/data/optimization

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

get_electrical_and_thermal_network_optimization_slave_storage_operation_data()  
scenario/outputs/data/calibration/clustering/checkpoints/...

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

get_input_folder()  
Returns the inputs folder of a scenario

get_input_network_folder()  
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_measurements()  
scenario/inputs/ Operation pattern for decentralized buildings

get_minimum_spanning_tree()  
get_minmaxscalar_model()  
scenario/outputs/data/surrogate/neural_network_folder

get_minmaxscaler_folder()  
scenario/outputs/data/surrogate/neural_network_folder

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

9.1. cea package
get_mpc_results_building_definitions_folder (output_folder='mpc-building')
get_mpc_results_controls (building, output_folder)
get_mpc_results_district_plot_grid (output_folder='mpc-district')
get_mpc_results_district_plot_streets (output_folder='mpc-district')
get_mpc_results_electric_power (output_folder)
get_mpc_results_folder (output_folder='mpc-building')
get_mpc_results_max_outputs (building, output_folder)
get_mpc_results_max_temperature (output_folder)
get_mpc_results_min_outputs (building, output_folder)
get_mpc_results_min_temperature (output_folder)
get_mpc_results_outputs (building, output_folder)
get_mpc_results_predicted_temperature (output_folder)
get_mpc_results_set_temperature (output_folder)
get_mpc_results_states (building, output_folder)
get_multi_criteria_analysis (generation)
get_multi_criteria_results_folder ()
get_naming ()
get_network_energy_pumping_requirements_file (network_type, network_name, representative_week=False)
get_network_input_paths (name)
get_network_layout_edges_shapefile (network_type, network_name)
get_network_layout_nodes_csv_file (network)
get_network_layout_nodes_shapefile (network_type, network_name="")

Returns plots/naming.csv
Pressure drop over an entire district heating or cooling network at each time step
Network layout files for nodes of district heating or cooling networks
get_network_layout_pipes_csv_file(network)
scenario/outputs/data/optimization/network/layout/DH_PipesData.csv or DC_PipesData.csv Network layout files for pipes of district heating or cooling networks

get_network_linear_pressure_drop_edges(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_P_DeltaP.csv or DC_P_DeltaP.csv Pressure drop over an entire district heating or cooling network at each time step

get_network_linear_thermal_loss_edges_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_qloss_System_kw.csv

get_network_pressure_at_nodes(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_P_DeltaP.csv or DC_P_DeltaP.csv Pressure drop over an entire district heating or cooling network at each time step

get_network_temperature_plant(network_type, network_name, representative_week=False)

get_network_temperature_return_nodes_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_T_Return.csv or DC_T_Return.csv Return temperatures at each node for each time step for a district heating or cooling network

get_network_temperature_supply_nodes_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_T_Supply.csv or DC_T_Supply.csv Supply temperatures at each node for each time step for a district heating or cooling network

get_network_thermal_loss_edges_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_qloss_System_kw.csv

get_network_total_pressure_drop_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_P_DeltaP.csv or DC_P_DeltaP.csv Pressure drop over an entire district heating or cooling network at each time step

get_network_total_thermal_loss_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_qloss_System_kw.csv

get_networks_folder()

get_networks_plots_file(network_name, category)
scenario/outputs/plots/timeseries/{network_name}.html

get_neural_network_estimates()
scenario/outputs/data/surrogate/neural_network_folder

get_neural_network_folder()
scenario/outputs/data/surrogate/neural_network_folder

get_neural_network_model()
scenario/outputs/data/surrogate/neural_network_folder

get_neural_network_resume()
scenario/outputs/data/surrogate/neural_network_folder

get_nn_inout_folder()
scenario/outputs/data/surrogate
get_nominal_edge_mass_flow_csv_file\(\text{network_type, network_name}\)

scenario/outputs/data/optimization/network/layout/DH_NodesData.csv or DC_NodesData.csv Network layout files for nodes of district heating or cooling networks

get_nominal_node_mass_flow_csv_file\(\text{network_type, network_name}\)

scenario/outputs/data/optimization/network/layout/DH_NodesData.csv or DC_NodesData.csv Network layout files for nodes of district heating or cooling networks

generate_optimization_all_individuals ()

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_checkpoint (generation)

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_checkpoint_final ()

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_checkpoint_initial ()

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_clustering_folder ()

scenario/outputs/data/optimization/clustering_sax Clustering results for decentralized buildings

generate_optimization_connected_cooling_capacity (ind_num, gen_num)

generate_optimization_connected_electricity_capacity (ind_num, gen_num)

generate_optimization_connected_heating_capacity (ind_num, gen_num)

generate_optimization_decentralized_folder ()

scenario/outputs/data/optimization/decentralized Operation pattern for decentralized buildings

generate_optimization_decentralized_folder_building_cooling_activation (buildingname, configuration='AHU_ARU_SCU')

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_decentralized_folder_building_result_cooling (buildingname, configuration='AHU_ARU_SCU')

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_decentralized_folder_building_result_heating (buildingname)

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_decentralized_folder_building_result_heating_activation (buildingname)

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_decentralized_folder_disc_op_summary_cooling ()

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_decentralized_folder_disc_op_summary_heating ()

scenario/outputs/data/calibration/clustering/checkpoints/...

generate_optimization_decentralized_result_file (building_name)

scenario/outputs/data/optimization/decentralized/DiscOp_${building_name}_result.csv

generate_optimization_disconnected_cooling_capacity (ind_num, gen_num)

generate_optimization_disconnected_heating_capacity (ind_num, gen_num)
get_optimization_generation_connected_performance(gen_num)  
scenario/outputs/data/calibration/clustering/checkpoints/…

get_optimization_generation_disconnected_performance(gen_num)  
scenario/outputs/data/calibration/clustering/checkpoints/…

get_optimization_generation_electricity_performance(gen_num)  
scenario/outputs/data/calibration/clustering/checkpoints/…

get_optimization_generation_heating_performance(gen_num)  
scenario/outputs/data/calibration/clustering/checkpoints/…

get_optimization_generation_total_performance(gen_num)  
scenario/outputs/data/calibration/clustering/checkpoints/…

get_optimization_generation_total_performance_halloffame(gen_num)  
scenario/outputs/data/calibration/clustering/checkpoints/…

get_optimization_generation_total_performance_pareto(gen_num)  
scenario/outputs/data/calibration/clustering/checkpoints/…

get_optimization_individuals_in_generation(gen_num)  
scenario/outputs/data/calibration/clustering/checkpoints/…

get_optimization_master_results_folder()  
Returns the folder containing the scenario’s optimization Master Checkpoints

get_optimization_network_all_individuals_results_file(network_type)  
scenario/outputs/data/optimization/network/layout/DH_T_Return.csv or DC_T_Return.csv Folder to results file of this generation

get_optimization_network_generation_folder(generation)  
scenario/outputs/data/calibration/clustering/checkpoints/…

get_optimization_network_generation_individuals_results_file(network_type, generation)  
scenario/outputs/data/optimization/network/layout/DH_T_Return.csv or DC_T_Return.csv Folder to results file of this generation

get_optimization_network_individual_results_file(network_type, individual)  
scenario/outputs/data/optimization/network/layout/DH_T_Return.csv or DC_T_Return.csv Folder to results file of this generation

get_optimization_network_layout_costs_file(network_type)  
scenario/outputs/data/optimization/network/layout/DC_costs.csv Optimized network layout files for pipes of district heating networks

get_optimization_network_layout_costs_file_concept(network_type, network_number, generation_number)  
scenario/outputs/data/optimization/network/layout/DC_costs.csv Optimized network layout files for pipes of district heating networks

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

get_optimization_network_layout_pipes_file(network_type)  
scenario/outputs/data/optimization/network/layout/DH_PipesData.csv Optimized network layout files for pipes of district heating networks

get_optimization_network_results_folder()  
scenario/outputs/data/optimization/network Network summary results
get_optimization_network_results_summary(network_type, key)
scenario/outputs/data/calibration/clustering/checkpoints/

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

get_optimization_network_totals_folder_total(network_type, indCombi)
scenario/outputs/data/calibration/clustering/checkpoints/

get_optimization_results_folder()
Returns the folder containing the scenario’s optimization results

generate_optimization_slave_building_connectivity(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_connected_performance(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_cooling_activation_pattern(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_cooling_opex_var(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_disconnected_performance(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_electricity_activation_pattern(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_electricity_performance(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_electricity_requirements_data(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_energy_mix_based_on_technologies(ind_num, gen_num, category)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_heating_activation_pattern(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_heating_opex_var_pattern(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_heating_performance(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_natural_gas_imports(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

get_optimization_slave_results_folder(gen_num)
Returns the folder containing the scenario’s optimization Slave results (storage + operation pattern)

generate_optimization_slave_storage_flag(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_storage_operation_data(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_storage_sizing_parameters(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/

generate_optimization_slave_total_performance(ind_num, gen_num)
scenario/outputs/data/calibration/clustering/checkpoints/
get_optimization_substations_folder()
scenario/outputs/data/optimization/substations Substation results for decentralized buildings

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

get_optimization_substations_total_file(genome, network_type)
scenario/outputs/data/optimization/substations/Total_${genome}.csv

get_optimization_thermal_network_data_file(network_data_file)
scenario/outputs/data/optimization/network Network summary results

get_plant_nodes(network_type, network_name)
Return the list of “PLANT” nodes in a thermal network

get_plots_folder(category)
scenario/outputs/plots/timeseries

get_potentials_folder()
scenario/outputs/data/potentials

get_potentials_retrofit_folder()
scenario/outputs/data/potentials/retrofit.csv

get_potentials_solar_folder()
scenario/outputs/data/potentials/solar Contains raw solar files

get_predefined_hourly_setpoints(building_name, type_of_district_network)
scenario/outputs/data/demand/${building_name}_schedules.csv This file contains schedules of occupancy, appliance use, etc of each building. Schedules are 8760 values per year.

get_predefined_hourly_setpoints_folder(type_of_district_network)

get_preprocessing_costs()
scenario/outputs/data/calibration/clustering/checkpoints/

get_project_path()
Returns the parent folder of a scenario - this is called a project or ‘case-study’

get_radiation_building(building_name)
scenario/outputs/data/solar-radiation/${building_name}_insolation.json

get_radiation_building_sensors(building_name)
scenario/outputs/data/solar-radiation/${building_name}_insolation_Whm2.json

get_radiation_materials()
scenario/outputs/data/solar-radiation/${building_name}_geometry.csv

get_radiation_metadata(building_name)
scenario/outputs/data/solar-radiation/${building_name}_geometry.csv

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

get_result_building_NN(name)
scenario/outputs/data/surrogate/neural_network_folder

get_retrofit_filters(name_retrofit)
scenario/outputs/data/potentials/retrofit.csv

get_schedule_model_file(building_name)
scenario/inputs/building-properties/${building_name}_schedules.csv This file contains schedules of occupancy, appliance use, etc of each building. Schedules are 8760 values per year.

9.1. cea package 339
get_schedule_model_folder()
scenario/outputs/data/optimization/slave' Slave results folder (storage + operation pattern)

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_site_polygon()
scenario/inputs/building-geometry/site.shp

get_solar_radiation_folder()
scenario/outputs/data/solar-radiation

get_street_network()

get_surrogate_folder()
scenario/outputs/data/surrogate

get_surroundings_geometry()
scenario/inputs/building-geometry/surroundings.shp

get_systems_seasonality()
Returns the database of region-specific system control parameters. These are copied to the scenario if they
are not yet present, based on the configured region for the scenario.

Parameters region –

Returns

get_technology_template_for_region (region)
get path to technology database for the region as shipped by the CEA or return the region path, assuming
it’s a user-supplied technology folder

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_terrain_folder()

get_thermal_demand_csv_file (network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_NodesData.csv or DC_NodesData.csv Network
layout files for nodes of district heating or cooling networks

get_thermal_network_edge_list_file (network_type, network_name=“”)
scenario/outputs/data/optimization/network/layout/DH_AllEdges.csv or DC_AllEdges.csv List of edges
in a district heating or cooling network and their start and end nodes

get_thermal_network_edge_node_matrix_file (network_type, network_name)
scenario/outputs/data/optimization/network/layout/DH_EdgeNode.csv or DC_EdgeNode.csv Edge-node
matrix for a heating or cooling network

get_thermal_network_folder()
get_thermal_network_layout_massflow_edges_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_MassFlow.csv or DC_MassFlow.csv Mass flow rates at each edge in a district heating or cooling network

get_thermal_network_layout_massflow_nodes_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_MassFlow.csv or DC_MassFlow.csv Mass flow rates at each edge in a district heating or cooling network

get_thermal_network_node_types_csv_file(network_type, network_name)
scenario/outputs/data/optimization/network/layout/DH_Nodes.csv or DC_NodesData.csv Network layout files for nodes of district heating or cooling networks

get_thermal_network_plant_heat_requirement_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_Plant_heat_requirement.csv or DC_Plant_heat_requirement.csv Heat requirement at from the plants in a district heating or cooling network

get_thermal_network_pressure_losses_edges_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_qloss_System_kw.csv

get_thermal_network_substation_ploss_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_qloss_substations_kw.csv

get_thermal_network_velocity_edges_file(network_type, network_name, representative_week=False)
scenario/outputs/data/optimization/network/layout/DH_MassFlow.csv or DC_MassFlow.csv Mass flow rates at each edge in a district heating or cooling network

get_timeseries_plots_file(building_name, category="")
scenario/outputs/plots/timeseries/{building_name}.html :param category:

get_total_demand(format='csv')
scenario/outputs/data/demand/Total_demand.csv

get_uncertainty_checkpoint(generation)
scenario/outputs/data/calibration/clustering/checkpoints/...

get_uncertainty_db()
databases/CH/Uncertainty/uncertainty_distributions.xls

get_uncertainty_results_folder()

get_water_body_potential()

get_weather(name=None)
weather/{name}.epw Returns the path to a weather file with the name name. This can either be one of the pre-configured weather files (see get_weather_names) or a path to an existing weather file. Returns the default weather file if no other file can be resolved. ..note: scripts should not use this, instead, use get_weather_file() - see the weather-helper script.

get_weather_dict()
Return a dictionary with weather_name -> weather_path for the builtin weather files

get_weather_file()
inputs/weather/weather.epw path to the weather file to use for simulation - run weather-helper to set this

get_weather_folder()
get_weather_names()
Return a list of all installed epw files in the system

get_zone_building_names()
Return the list of buildings in the Zone

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

list_optimization_all_individuals()
Return a list of “scenario/generation/individual” strings for scenario comparisons

radiation_results(building_name)
scenario/outputs/data/potentials/solar/[building_name]_PV.csv

solar_potential_folder()

class cea.inputlocator.ReferenceCaseOpenLocator
Bases: cea.inputlocator.InputLocator
This is a special InputLocator that extracts the builtin reference case (cea/examples/reference-case-open.zip) to the temporary folder and uses the baseline scenario in there

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

get_default_weather()
The reference-case-open uses the Zug weather file…

9.1.8 cea.scripts module

Provides the list of scripts known to the CEA - to be used by interfaces built on top of the CEA.

class cea.scripts.CeaScript(script_dict, category)
Bases: object

__init__(script_dict, category)
x.__init__(...) initializes x; see help(type(x)) for signature

__repr__( ) <=> repr(x)

_lookup_args(config, locator, args)
returns a list of arguments to a locator method

missing_input_files(config)
Return a list of bound cea.inputlocator.InputLocator method names, one for each file re-
quired as input for this script that is not present yet as well as the applied path searched for. :return: Sequence[str]

print_missing_input_files(config)

print_script_configuration(config, verb='Running')
Print a list of script parameters being used for this run of the tool. Historically, each tool was responsible
for printing their own parameters, but that requires manually keeping track of these parameters.

tools_api.cea.scripts._get_categories_dict()
Load the categories -> [script] mapping either from the YAML file or, in the case of arcgis / grasshopper, which
don’t support YAML, load from a pickled version generated on the call to cea install-toolbox.

tools_api.cea.scripts.by_name(script_name)

tools_api.cea.scripts.for_interface(interface='cli')
Return the list of CeaScript instances that are listed for the interface
cea.scripts.get_schema_scripts(schema)

This method returns a set of all variables within the schema.yml. The set is organised by: (variable_name, locator_method, script, file_name:sheet_name) If the variable is from an input database, the script is replaced by “-”. Also, if the variable is not from a tree data shape (such as xlsx or xls), the ‘file_name:sheet_name’ becomes ‘file_name’ only. The sheet_name is important to consider as a primary key for each variable can only be made through combining the ‘file_name:sheet_name’ and ‘variable_name’. Along with the locator_method, the set should contain all information necessary for most tasks.

cea.scripts.get_schema_variables(schema)

This method returns a set of all variables within the schema.yml. The set is organised by: (variable_name, locator_method, script, file_name:sheet_name) If the variable is from an input database, the script is replaced by “-”. Also, if the variable is not from a tree data shape (such as xlsx or xls), the ‘file_name:sheet_name’ becomes ‘file_name’ only. The sheet_name is important to consider as a primary key for each variable can only be made through combining the ‘file_name:sheet_name’ and ‘variable_name’. Along with the locator_method, the set should contain all information necessary for most tasks.

cea.scripts.list_scripts()
List all scripts

cea.scripts.schemas()
Return the contents of the schemas.yml file

9.1.9 cea.worker module

cea-worker: a worker-process that uses the /server/jobs/ api to figure out what needs to be loaded to complete a job. All output is wired through the /server/streams/ api.

In the future, (Dataframe-) file reading / writing will happen through a /server/data api.

This script is _not_ part of the scripts.yml, because it’s a _consumer_ of that file: It is installed with it’s own distutils entry-point (cea-worker) with it’s own semantics for argument processing: A single integer argument, the jobid, that is used to fetch all other information from the /server/jobs api, as well as an URL for locating the /server/jobs api.

class cea.worker.JobServerStream(jobid, server, stream)
Bases: object

A File-like object for capturing STDOUT and STDERR from cea-worker processes on the server.

__init__(jobid, server, stream)

x.__init__(...) initializes x; see help(type(x)) for signature

close()
Send sentinel that we’re done writing

flush()

isatty()

write(str)

cea.worker.configure_streams(jobid, server)
Capture STDOUT and STDERR writes and post them to the /server/

cea.worker.consume_nowait(queue, msg)
Read from queue as much as possible and concatenate the results to msg, returning that. If an EOFError is read from the queue, put it back and return the msg so far.

cea.worker.fetch_job(jobid, server)

cea.worker.main(config=None)

cea.worker.parse_arguments(default_url)

cea.worker.post_error(exc, jobid, server)

cea.worker.post_started(jobid, server)

cea.worker.post_success(jobid, server)
cea.worker.read_parameters(job)
    Return the parameters of the job in a format that is valid for using as **kwargs

deli.worker.read_script(job)
    Locate the script defined by the job dictionary in the cea.api module, take care of dashes

deli.worker.run_job(config, job, server)

deli.worker.stream_poster(jobid, server, queue)
    Post items from queue until a sentinel (the EOFError class object) is read.

deli.worker.worker(config, jobid, server)
    This is the main logic of the cea-worker.
CHAPTER 10

Indices and tables

- genindex
- modindex
- search
Bibliography


[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

Python Module Index

a
cea.analysis, 94
cea.analysis.costs, 94
cea.analysis.costs.operation_costs, 94
cea.analysis.ica, 94
cea.analysis.ica.embodied, 94
cea.analysis.ica.main, 96
cea.analysis.ica.operation, 97
cea.analysis.ica.multiplicity, 97
cea.analysis.multicriteria, 97
cea.analysis.sensitivity, 97
cea.analysis.sensitivity.sensitivity_demand_analyse, 98
cea.analysis.sensitivity.sensitivity_demand_count, 99
cea.analysis.sensitivity.sensitivity_demand_samples, 100
cea.analysis.sensitivity.sensitivity_demand_simulate, 101
cea.api, 318
c
cea, 93
cea.config, 322
cea.constants, 329
d
cea.datamanagement, 103
cea.datamanagement.create_new_project, 103
cea.datamanagement.data_helper, 104
cea.datamanagement.databases_verification, 106
cea.datamanagement.schedule_helper, 106
cea.datamanagement.streets_helper, 107
cea.datamanagement.surroundings_helper, 107
cea.datamanagement.terrain_helper, 108
cea.datamanagement.weather_helper, 108
cea.datamanagement.zone_helper, 109
cea.demand, 109
cea.demand.airconditioning_model, 111
cea.demand.building_properties, 113
cea.demand.constants, 120
cea.demand.control_heating_cooling_systems, 120
cea.demand.control_ventilation_systems, 124
cea.demand.datacenter_loads, 125
cea.demand.demand_main, 125
cea.demand.demand_writers, 126
cea.demand.electrical_loads, 127
cea.demand.hotwater_loads, 128
cea.demand.hourly_procedure_heating_cooling_systems, 130
cea.demand.latent_loads, 133
cea.demand.model_SIA, 136
cea.demand.refrigeration_loads, 139
cea.demand.schedule_maker, 109
cea.demand.schedule_maker.schedule_maker, 109
cea.demand.sensible_loads, 140
cea.demand.set_point_from_predefined_file, 141
cea.demand.space_emission_systems, 142
cea.demand.thermal_loads, 144
cea.demand.ventilation_air_flows_detailed, 146
cea.demand.ventilation_air_flows_simple, 151
e
cea.examples, 152
cea.examples.extract_reference_case, 152
cea.examples.template, 152
f

g
cea.glossary, 329

349
Python Module Index
Python Module Index
## Index

### Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>__apply_func_with_worker_stream()</td>
<td>(in module cea.utilities.parallel)</td>
<td>310</td>
</tr>
<tr>
<td><strong>getattr</strong>()</td>
<td>(cea.config.Configuration method)</td>
<td>323</td>
</tr>
<tr>
<td><strong>getattr</strong>()</td>
<td>(cea.config.Section method)</td>
<td>327</td>
</tr>
<tr>
<td><strong>getitem</strong>()</td>
<td>(cea.demand.building_properties.BuildingProperties method)</td>
<td>113</td>
</tr>
<tr>
<td><strong>getnewargs</strong>()</td>
<td>(cea.technologies.thermal_network.thermal_network.HourlyThermalResults method)</td>
<td>257</td>
</tr>
<tr>
<td><strong>getnewargs</strong>()</td>
<td>(cea.utilities.solar_equations.SunProperties method)</td>
<td>311</td>
</tr>
<tr>
<td><strong>getstate</strong>()</td>
<td>(cea.config.Configuration method)</td>
<td>323</td>
</tr>
<tr>
<td><strong>getstate</strong>()</td>
<td>(cea.technologies.thermal_network.thermal_network.HourlyThermalResults method)</td>
<td>257</td>
</tr>
<tr>
<td><strong>getstate</strong>()</td>
<td>(cea.utilities.solar_equations.SunProperties method)</td>
<td>311</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.config.Configuration method)</td>
<td>323</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.config.Parameter method)</td>
<td>325</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.config.Section method)</td>
<td>327</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.datamanagement.schedule_helper.ScheduleData method)</td>
<td>106</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.datamanagement.schedule_helper.ScheduleData method)</td>
<td>113</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.demand.building_properties.BuildingProperties method)</td>
<td>116</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.demand.building_properties.EnvelopeProperties method)</td>
<td>118</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.demand.building_properties.SolarProperties method)</td>
<td>118</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.demand.demand_writers.DemandWriter method)</td>
<td>126</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.demand.demand_writers.HourlyDemandWriter method)</td>
<td>126</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.demand.demand_writers.MonthlyDemandWriter method)</td>
<td>127</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.demand.demand_writers.YearlyDemandWriter method)</td>
<td>127</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.inputlocator.InputLocator method)</td>
<td>330</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.inputlocator.ReferenceCaseOpenLocator method)</td>
<td>342</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.interfaces.arcgis.CityEnergyAnalyst.DemandTool method)</td>
<td>153</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.interfaces.arcgis.CityEnergyAnalyst.RadiationDaysimTool method)</td>
<td>153</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.interfaces.arcgis.CityEnergyAnalyst.Toolbox method)</td>
<td>153</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.interfaces.arcgis.arcgishelper.ParameterInfoBuilder method)</td>
<td>155</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.optimization.distribution.network_optimization_features method)</td>
<td>165</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.optimization.lca_calculations.LcaCalculations method)</td>
<td>192</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.optimization.prices.Prices method)</td>
<td>193</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.optimization.slave.daily_storage.load_leveling.LoadLevelingDailyStorage method)</td>
<td>182</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.optimization.slave.daily_storage.load_leveling.LoadLevelingMonthlyStorage method)</td>
<td>182</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.optimization.slave.daily_storage.load_leveling.LoadLevelingYearlyStorage method)</td>
<td>182</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.plots.Dashboard method)</td>
<td>193</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.plots.cache.MemoryPlotCache method)</td>
<td>227</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.plots.cache.NullPlotCache method)</td>
<td>228</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.plots.cache.PlotCache method)</td>
<td>228</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.plots.categories.PlotCategory method)</td>
<td>229</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.plots.comparisons.Annual_costs.ComparisonsAnnualCostsPlot method)</td>
<td>197</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.plots.comparisons.ComparisonsPlotBase method)</td>
<td>194</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.plots.comparisons.old.main.Plots method)</td>
<td>196</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>(cea.plots.demand.DemandPlotBase method)</td>
<td>198</td>
</tr>
</tbody>
</table>
Index
blockPrint() (in module cea.resources.radiation_daysim.geometry_generator), 231
BooleanParameter (class in cea.config), 322
breed_new_generation() (in module cea.technologies.thermal_network.thermal_network_optimization), 278
building_2d_to_3d() (in module cea.resources.radiation_daysim.geometry_generator), 231
building_node_from_name() (in module cea.technologies.network_layout.steiner_spanning_tree), 239
BuildingData (class in cea.resources.radiation_daysim.geometry_generator), 230
BuildingDataFinale (class in cea.resources.radiation_daysim.geometry_generator), 231
BuildingProperties (class in cea.demand.building_properties), 113
BuildingPropertiesRow (class in cea.demand.building_properties), 116
buildings_connected_costs_and_emissions() (in module cea.optimization.master.cost_model), 165
buildings_disconnected_costs_and_emissions() (in module cea.optimization.master.cost_model), 172
buildings_hourly (cea.plots.thermal_networks.ThermalNetworksPlotBase attribute), 223
calc_absorbed_radiation_PV() (in module cea.technologies.solar.photovoltaic), 242
calc_ACH_operation() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 177
calc_aggregated_heat_conduction_coefficient() (in module cea.technologies.thermal_network.thermal_network), 260
calc_air_flow_mass_balance() (in module cea.resources.radiation_daysim.geometry_generator), 231
calc_air_flows() (in module cea.demand.ventilation_air_flows_detailed), 147
calc_air_flow_mass_fraction() (in module cea.technologies.thermal_network.thermal_network), 260
calc_air_mass_flow_mechanical_ventilation() (in module cea.resources.radiation_daysim.geometry_generator), 230
calc_air_mass_flow_window_ventilation() (in module cea.demand.ventilation_air_flows_simple), 151
calc_anchor_load_building() (in module cea.technologies.thermal_network.thermal_network_optimization), 278
calc_area_buildings() (in module cea.resources.geothermal), 233
calc_area_HEX() (in module cea.technologies.substation), 298
calc_area_window_cros() (in module cea.technologies.thermal_network.substation_matrix), 252
calc_area_window_free() (in module cea.demand.ventilation_air_flows_detailed), 147
calc_area_window_tot() (in module cea.demand.ventilation_air_flows_detailed), 147
calc_available_area_solar() (in module cea.optimization.master.master_to_slave), 172
calc_available_generation_PV() (in module cea.optimization.slave.electricity_main), 189
C

Index 357
calc_available_generation_solar() (in module cea.optimization.slave.seasonal_storage.storage_main), 288

calc_average() (in module cea.datamanagement.schedule_helper), 106

calc_blinds_activation() (in module cea.technologies.blinds), 280

calc_boiler_operation() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 178

calc_bounding_box() (in module cea.datamanagement.streets_helper), 107

calc_bounding_box_geom() (in module cea.demand.building_properties.BuildingProperties method), 114

calc_bounding_box_projected_coordinates() (in module cea.datamanagement.terrain_helper), 108

calc_building_connectivity_dict() (in module cea.optimization.master.generation), 169

calc_building_geometry_surroundings() (in module cea.resources.radiation_daysim.geometry_generator), 231

calc_building_geometry_zone() (in module cea.resources.radiation_daysim.geometry_generator), 231

calc_burner_operation() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 178

calc_Capex_a_network_pipes() (in module cea.technologies.thermal_network.thermal_network_costs), 276

calc_categoriesroof() (in module cea.utilities.solar_equations), 312

calc_category() (in module cea.datamanagement.data_helper), 104

calc_CC_operation() (in module cea.technologies.cogeneration), 286

calc_cell_temperature() (in module cea.technologies.solar.photovoltaic), 242

calc_chiller_absorption_operation() (in module cea.optimization.slave.cooling_resource_activation), 188

calc_chiller_main() (in module cea.technologies.chiller_absorption), 283

calc_Cinv_ACH() (in module cea.technologies.chiller_absorption), 283

calc_Cinv_boiler() (in module cea.technologies.boiler), 281

calc_Cinv_burner() (in module cea.technologies.burner), 282

calc_Cinv_CCGT() (in module cea.technologies.cogeneration), 286

calc_Cinv_CT() (in module cea.technologies.cooling_tower), 288

calc_Cinv_DX() (in module cea.technologies.direct_expansion_units), 288

calc_Cinv_FC() (in module cea.technologies.cogeneration), 286

calc_Cinv_furnace() (in module cea.technologies.furnace), 289

calc_Cinv_gas() (in module cea.resources.natural_gas), 234

calc_Cinv_GHP() (in module cea.technologies.heatpumps), 293

calc_Cinv_HEX() (in module cea.technologies.heat_exchangers), 290

calc_Cinv_HEX_hisaka() (in module cea.technologies.heat_exchangers), 290

calc_Cinv_HP() (in module cea.technologies.pumps), 294

calc_Cinv_HX() (in module cea.technologies.heat_exchangers), 290

calc_Cinv_PVT() (in module cea.technologies.solar.photovoltaic_thermal), 245

calc_Cinv_SC() (in module cea.technologies.solar.solar_collector), 247

calc_Cinv_storage() (in module cea.technologies.thermal_storage), 302

calc_Cinv_VCC() (in module cea.technologies.chiller_vapor_compression), 285

calc_code() (in module cea.analysis.lca.embodied), 94

calc_coeff_lea_zone() (in module cea.demand.ventilation_air_flows_detailed), 147

calc_coeff_vent_zone() (in module cea.demand.ventilation_air_flows_detailed), 147

calc_cold_storage_charge_HEX() (in module cea.technologies.storage_tank), 295

calc_cold_storage_discharge_HEX() (in module cea.technologies.storage_tank), 295

calc_cold_tank_heat_loss() (in module cea.technologies.storage_tank), 295

calc_combined_cooling_loads() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 178

358 Index
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Module Name</th>
<th>Module Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calc_comparison()</td>
<td>cea.datamanagement.data_helper</td>
<td>104</td>
</tr>
<tr>
<td>calc_compound_Tcs()</td>
<td>cea.technologies.substation</td>
<td>299</td>
</tr>
<tr>
<td>calc_compound_Ths()</td>
<td>cea.technologies.substation</td>
<td>299</td>
</tr>
<tr>
<td>calc_connected_names()</td>
<td></td>
<td>172</td>
</tr>
<tr>
<td>calc_connectivity_network()</td>
<td>cea.technologies.network_layout.connectivity_potential</td>
<td>236</td>
</tr>
<tr>
<td>calc_convergence_metrics()</td>
<td>cea.plots.optimization.paretocurve_convergence.OptimizationPerformance</td>
<td>215</td>
</tr>
<tr>
<td>calc_COP()</td>
<td>cea.technologies.chiller_vapor_compression</td>
<td>285</td>
</tr>
<tr>
<td>calc_Cop_boiler()</td>
<td>cea.technologies.boiler</td>
<td>281</td>
</tr>
<tr>
<td>calc_Cop_burner()</td>
<td>cea.technologies.burner</td>
<td>282</td>
</tr>
<tr>
<td>calc_Cop_CCGT()</td>
<td>cea.technologies.cogeneration</td>
<td>287</td>
</tr>
<tr>
<td>calc_Cop_DX()</td>
<td>cea.technologies.direct_expansion_units</td>
<td>288</td>
</tr>
<tr>
<td>calc_Cop_GHP()</td>
<td>cea.technologies.heatpumps</td>
<td>293</td>
</tr>
<tr>
<td>calc_Cop_with_carnot_efficiency()</td>
<td></td>
<td>285</td>
</tr>
<tr>
<td>calc_costs_emissions_decentralized_DC()</td>
<td>cea.optimization.master.cost_model</td>
<td>166</td>
</tr>
<tr>
<td>calc_costs_emissions_decentralized_DH()</td>
<td></td>
<td>166</td>
</tr>
<tr>
<td>calc_ctrem_pv()</td>
<td>cea.technologies.solar.photovoltaic</td>
<td>241</td>
</tr>
<tr>
<td>calc_CT()</td>
<td>cea.technologies.cooling_tower</td>
<td>288</td>
</tr>
<tr>
<td>calc_CT_partload_factor()</td>
<td>cea.technologies.cooling_tower</td>
<td>288</td>
</tr>
<tr>
<td>calc_CT_yearly()</td>
<td>cea.technologies.cooling_tower</td>
<td>288</td>
</tr>
<tr>
<td>calc_Ctot_cooling_plants()</td>
<td>cea.technologies.thermal_network.thermal_network_costs</td>
<td>276</td>
</tr>
<tr>
<td>calc_Ctot_cs_disconnected_buildings()</td>
<td>cea.technologies.thermal_network.thermal_network_costs</td>
<td>276</td>
</tr>
<tr>
<td>calc_Ctot_cs_district()</td>
<td>cea.technologies.thermal_network.thermal_network_costs</td>
<td>277</td>
</tr>
<tr>
<td>calc_Ctot_network_pump()</td>
<td>cea.technologies.thermal_network.thermal_network_costs</td>
<td>277</td>
</tr>
<tr>
<td>calc_data()</td>
<td>cea.plots.demand.comfort_chart</td>
<td>199</td>
</tr>
<tr>
<td>calc_data_frame()</td>
<td></td>
<td>224</td>
</tr>
<tr>
<td>calc_datetime_local_from_weather_file()</td>
<td>cea.utilities.solar_equations</td>
<td>313</td>
</tr>
<tr>
<td>calc_DC_return()</td>
<td>cea.technologies.substation</td>
<td>297</td>
</tr>
<tr>
<td>calc_DC_supply()</td>
<td>cea.technologies.substation</td>
<td>297</td>
</tr>
</tbody>
</table>

Index 359
calc_dehumidification_moisture_load() (in module cea.demand.latent_loads), 133
calc_delta_p_path() (in module cea.demand.ventilation_air_flows_detailed), 147
calc_delta_theta_int_inc_cooling() (in module cea.demand.space_emission_systems), 142
calc_delta_theta_int_inc_heating() (in module cea.demand.space_emission_systems), 142
calc_DH_return() (in module cea.technologies.substation), 298
calc_DH_supply() (in module cea.technologies.substation), 298
calc_DH_supply() (in module cea.technologies.thermal_network.substation_matrix), 251
calc_DH_ww_with_tank_losses() (in module cea.demand.hotwater_loads), 128
calc_dhw_tank_heat_balance() (in module cea.technologies.storage_tank), 295
calc_dictionary_of_all_individuals_tested() (in module cea.optimization.master.master_main), 170
calc_diffuseground_comp() (in module cea.technologies.solar.photovoltaic), 243
calc_disls() (in module cea.demand.hotwater_loads), 129
calc_district_system_electricity_generated() (in module cea.optimization.slave.electricity_main), 189
calc_district_system_electricity_requirements() (in module cea.optimization.slave.electricity_main), 189
calc_dTm_HEX() (in module cea.technologies.substation), 299
calc_dTm_HEX() (in module cea.technologies.thermal_network.substation_matrix), 253
calc_DX() (in module cea.technologies.direct_expansion_units), 288
calc_E_sys() (in module cea.demand.electrical_loads), 127
calc_Eal_Epro() (in module cea.demand.electrical_loads), 127
calc_Eaux() (in module cea.demand.electrical_loads), 127
calc_Eaux_fw() (in module cea.demand.electrical_loads), 127
calc_Eaux_panels() (in module cea.technologies.solar.solar_collector), 247
calc_Eaux_Qhs_Qcs() (in module cea.demand.electrical_loads), 127
calc_Eaux_SC() (in module cea.technologies.solar.solar_collector), 247
calc_Eaux_ww() (in module cea.demand.electrical_loads), 128
calc_Ef() (in module cea.demand.electrical_loads), 128
calc_effective_stack_height() (in module cea.demand.ventilation_air_flows_detailed), 148
calc_Ef_ic() (in module cea.demand.rc_model_SIA), 136
calc_Ef_im() (in module cea.demand.rc_model_SIA), 136
calc_Ef_sc() (in module cea.demand.rc_model_SIA), 136
calc_Ef_sm() (in module cea.demand.rc_model_SIA), 136
calc_Ef_ve() (in module cea.demand.rc_model_SIA), 136
calc_Ef_ww() (in module cea.demand.rc_model_SIA), 136
calc_Ef_cs_dis() (in module cea.demand.electrical_loads), 128
calc_Ef_fs_dis() (in module cea.demand.electrical_loads), 128
calc_Ef_ht_dis() (in module cea.demand.electrical_loads), 128
calc_Edata() (in module cea.demand.datacenter_loads), 125
calc_edge_temperatures() (in module cea.technologies.thermal_network.thermal_network), 262
calc_emissions_connected_buildings() (in module cea.optimization.master.cost_model), 166
calc_emissions_Whyr_to_tonCO2yr() (in module cea.optimization.master.emissions_model), 168
calc_eta_FC() (in module cea.technologies.cogeneration), 287
calc_eta_furnace() (in module cea.technologies.furnace), 289
calc_eta_FC() (in module cea.technologies.cogeneration), 287
calc_eta_furnace() (in module cea.technologies.furnace), 289
calc_electricity_performance_costs() (in module cea.optimization.slave.electricity_main), 189
calc_electricity_performance_emissions() (in module cea.optimization.slave.electricity_main), 189
calc_electricity_performance_emissions() (in module cea.optimization.slave.electricity_main), 189

calc_graph() (in module cea.plots.comparisons.old.operation_costs), 196
calc_graph() (in module cea.plots.comparisons.old.primary_energy), 196
calc_graph() (in module cea.plots.comparisons.old.primary_energy_intensity), 197
calc_graph() (in module cea.plots.demand.comfort_chart), 199
calc_graph() (in module cea.plots.demand.energy_balance), 202
calc_graph() (in module cea.plots.demand.energy_end_use), 203
calc_graph() (in module cea.plots.demand.load_duration_curve), 206
calc_graph() (in module cea.plots.life_cycle.emissions), 208
calc_graph() (in module cea.plots.life_cycle.emissions_intensity), 209
calc_graph() (in module cea.plots.life_cycle.operation_costs), 210
calc_graph() (in module cea.plots.life_cycle.primary_energy), 210
calc_graph() (in module cea.plots.life_cycle.primary_energy_intensity), 211
calc_graph() (in module cea.technologies.thermal_network.simplified_thermal_network), 250
calc_heat_loads_central_ac() (in module cea.demand.hourly_procedure_heating_cooling_system_load), 131
calc_heat_loads_radiator() (in module cea.demand.hourly_procedure_heating_cooling_system_load), 131
calc_heating_coil() (in module cea.technologies.heating_coils), 290
calc_heating_cooling_loads() (in module cea.demand.hourly_procedure_heating_cooling_system_load), 131
calc_heating_substation_heat_exchange() (in module cea.technologies.thermal_network.substation_matrix), 253
calc_hex_area_from_demand() (in module cea.technologies.thermal_network.substation_matrix), 253
calc_HEX_cooling() (in module cea.technologies.thermal_network.substation_matrix), 251
calc_HEX_heating() (in module cea.technologies.thermal_network.substation_matrix), 252
calc_HEX_mix() (in module cea.technologies.thermal_network.substation_matrix), 252
calc_HEX_mix_2_flows() (in module cea.technologies.substation), 298
calc_HEX_mix_3_flows() (in module cea.technologies.substation), 298
calc_hot_tank_heat_loss() (in module cea.technologies.storage_tank), 296
calc_hourly_dataframe() (cea.demand.demand_writers.DemandWriter method), 126
calc_hourly_value() (in module cea.demand.schedule_maker.schedule_maker), 109
calc_hr() (in module cea.demand.sensible_loads), 140
calc_humidification_moisture_load() (in module cea.demand.rc_model_SIA), 137
calc_I_rad() (in module cea.demand.latent_loads), 133
calc_I_sol() (in module cea.demand.latent_loads), 140
calc_IAM_beam_SC() (in module cea.technologies.solar.solar_collector), 248
calc_if_existing() (in module cea.analysis.lca.embodied), 94
calc_incident_angle_beam() (in module cea.utilities.solar_equations), 313
calc_individual_occupant_schedule() (in module cea.demand.schedule_maker.schedule_maker), 109
calc_input_variables() (cea.plots.comparisons.ComparisonsPlotBase method), 194
calc_intersection() (in module cea.resources.radiation_daysim.geometry_generator), 231
calc_intersection_face_solid() (in module cea.resources.radiation_daysim.geometry_generator), 232
calc_isol_daysim() (in module cea.demand.building_properties), 118
calc_kinematic_viscosity() (in module cea.technologies.thermal_network.thermal_network), 262
calc_lake_potential() (in module cea.resources.water_body_potential), 236
calc_linear_thermal_loss_coefficient() (in module cea.technologies.thermal_network.simplified_thermal_network), 250
calc_m_ve_leakage_simple() (in module cea.demand.ventilation_air_flows_simple), 151
calc_m_ve_required() (in module cea.demand.ventilation_air_flows_simple), 152
calc_mainuse() (in module cea.datamanagement.data_helper), 104
calc_mass_flow_edges() (in module cea.technologies.thermal_network.thermal_network), 262

calc_network_summary_DCN() (in module cea.optimization.slave.cooling_main), 187
calc_network_summary_DHN() (in module cea.optimization.slave.heating_main), 190
calc_new_load() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 178

calc_max_diameter() (in module cea.technologies.thermal_network.thermal_network), 250

calc_optimal_angle() (in module cea.technologies.solar.photovoltaic), 243

calc_optimal_angle() (in module cea.utilities.solar_equations), 313
calc_QH_sys_QC_sys()  (in module cea.demand.thermal_loads), 144
calc_Qhs_Qcs()  (in module cea.demand.thermal_loads), 144
calc_Qhs_Qcs_loss()  (in module cea.demand.sensible_loads), 140
calc_Qhs_Qcs_max()  (in module cea.demand.sensible_loads), 140
calc_Qhs_sys()  (in module cea.demand.thermal_loads), 144
calc_Qhs_sys_Qcs_sys()  (in module cea.demand.sensible_loads), 140
calc_Qhs_Qcs()  (in module cea.demand.sensible_loads), 140
calc_Qww()  (in module cea.demand.hotwater_loads), 128
calc_Qww_dis_ls_nr()  (in module cea.demand.hotwater_loads), 128
calc_Qww_dis_ls_r()  (in module cea.demand.hotwater_loads), 129
calc_Qww_sys()  (in module cea.demand.hotwater_loads), 129
calc_Qwwf()  (in module cea.demand.hotwater_loads), 129
calc_radiator()  (in module cea.technologies.radiators), 294
calc_raster_terrain_fixed_elevation()  (in module cea.datamanagement.terrain_helper), 108
calc_rc_cooling_demand()  (in module cea.demand.hourly_procedure_heating_cooling_system_load), 131
calc_rc_heating_demand()  (in module cea.demand.hourly_procedure_heating_cooling_system_load), 132
calc_rc_heating_demand()  (in module cea.demand.hourly_procedure_heating_cooling_system_load), 132
calc_rc_model_temperatures()  (in module cea.demand.refrigeration_loads), 139
calc_rc_model_temperatures_cooling()  (in module cea.demand.refrigeration_loads), 139
calc_rc_model_temperatures_heating()  (in module cea.demand.refrigeration_loads), 139
calc_rc_model_temperatures_no_heating_cooling()  (in module cea.demand.refrigeration_loads), 139
calc_rc_no_loads()  (in module cea.demand.hourly_procedure_heating_cooling_system_load), 132
calc_refrigeration_temperature_and_massflow()  (in module cea.demand.refrigeration_loads), 139
calc_required_moisture_mech_vent_dhu()  (in module cea.demand.latent_loads), 135
calc_required_moisture_mech_vent_hu()  (in module cea.demand.latent_loads), 135
calc_return_node_temperature()  (in module cea.technologies.thermal_network.thermal_network), 266
calc_return_temp()  (in module cea.optimization.master.summarize_network), 176
calc_return_temperatures()  (in module cea.technologies.thermal_network.thermal_network), 266
calc_reynolds()  (in module cea.technologies.thermal_network.thermal_network), 267
calc_rho_air()  (in module cea.utilities.physics), 310
calc_saturation_pressure()  (in module cea.demand.latent_loads), 135
calc_SC()  (in module cea.technologies.solar.solar_collector), 248
calc_SC_generation()  (in module cea.technologies.solar.solar_collector), 248
calc_schedules()  (in module cea.demand.schedule_maker.schedule_maker), 110
calc_seasonal_storage_costs()  (in module cea.optimization.master.cost_model), 167
calc_sensors_building()  (in module cea.resources.radiation_daysim.daysim_main), 230
calc_sensors_zone()  (in module cea.resources.radiation_daysim.daysim_main), 230

demand.thermal_loads), 144

demand.sensible_loads), 140
demand.sensible_loads), 140
demand.sensible_loads), 140

demand.sensible_loads), 140
calc_top_three_anchor_loads() (in module cea.plots.thermal_networks.nenergy_loss_bar),
calculate_end_points_intersections() (in module cea.technologies.network_layout.connectivity_potential),
calc_total_network_flow() (in module cea.technologies.thermal_network.substation_matrix),
calculate_external_temperature() (cea.demand.DemandPlotBase method),
calc_transmissivity() (in module cea.resources.radiation_daysim.radiation_main),
calculate_ground_temperature() (in module cea.technologies.thermal_network.simplified_thermal_network),
calc_u_wind_site() (in module cea.demand.ventilation_air_flows_detailed),
calculate_ground_temperature() (in module cea.technologies.thermal_network.thermal_network),
calc_useful_areas() (in module cea.demand.building_properties),
calculate_hourly_loads() (cea.demand.DemandPlotBase method),
calc_variable_costs_connected_buildings() (in module cea.optimization.master.cost_model),
calc_VCC() (in module cea.technologies.chiller_vapor_compression),
calc_VCC_COP() (in module cea.technologies.chiller_vapor_compression),
calc_VCC_CT_operation() (in module cea.optimization.slave.cooling_resource_activation),
calc_VCC_operation() (in module cea.optimization.slave.cooling_resource_activation),
calc_water_body_uptake_pumping() (in module cea.technologies.pumps),
calc_water_temperature() (in module cea.demand.hotwater_loads),
calc_wetbulb() (in module cea.utilities.epwreader),
calc_windows_walls() (in module cea.resources.radiation_daysim.geometry_generator),
calc_worst_hour() (in module cea.utilities.solar_equations),
calc_yearly_dataframe() (cea.demand.demand_writers.DemandWriter method),
calc_zone_building_solids() (cea.resources.radiation_daysim.geometry_generator.BuildingData method),
calculate_age_file() (in module cea.datamanagement.zone_helper),
calculate_average_multiuse() (in module cea.datamanagement.data_helper),
calculate_contributions() (in module cea.analysis.module),
(module), 160
cea.interfaces.dashboard (module), 161
cea.interfaces.dashboard.base (module), 161
cea.interfaces.dashboard.base.forms (module), 161
cea.interfaces.dashboard.base.routes (module), 163
cea.interfaces.dashboard.dashboard (module), 163
cea.interfaces.dashboard.inputs (module), 162
cea.interfaces.dashboard.inputs.routes (module), 162
cea.interfaces.dashboard.landing (module), 162
cea.interfaces.dashboard.plots (module), 162
cea.interfaces.dashboard.tools (module), 163
cea.interfaces.dashboard.tools.worker (module), 163
cea.interfaces.grasshopper (module), 163
cea.interfaces.grasshopper.ghhelper (module), 163
cea.interfaces.grasshopper.install_grasshopper (module), 164
ccea.optimization (module), 164
ccea.optimization.constants (module), 192
ccea.optimization.distribution (module), 165
ccea.optimization.distribution.network_optimization_features (module), 165
ccea.optimization.lca_calculations (module), 192
ccea.optimization.master (module), 165
ccea.optimization.master.cost_model (module), 165
ccea.optimization.master.crossover (module), 167
ccea.optimization.master.emissions_model (module), 168
ccea.optimization.master.evaluation (module), 168
ccea.optimization.master.generation (module), 169
ccea.optimization.master.master_main (module), 170
ccea.optimization.master.master_to_slave (module), 172
ccea.optimization.master.mutations (module), 175
ccea.optimization.master.normalization (module), 175
ccea.optimization.master.performance_aggregation (module), 175
ccea.optimization.master.summarize_network (module), 175
ccea.optimization.master.validation (module), 177
ccea.optimization.optimization_main (module), 192
ccea.optimization.preprocessing.decentralized_buildings_cooling (module), 177
ccea.optimization.preprocessing.decentralized_buildings_heating (module), 177
ccea.optimization.preprocessing.decentralized_buildings_electricity (module), 179
ccea.optimization.preprocessing.preprocessing_main (module), 180
ccea.optimization.preprocessing.processheat (module), 181
ccea.optimization.prices (module), 193
ccea.optimization.slave (module), 182
ccea.optimization.slave.cooling_main (module), 187
ccea.optimization.slave.cooling_resource_activation (module), 188
ccea.optimization.slave.daily_storage (module), 182
ccea.optimization.slave.daily_storage.load_leveling (module), 182
ccea.optimization.slave.electricity_main (module), 189
ccea.optimization.slave.heating_main (module), 190
ccea.optimization.slave.heating_resource_activation (module), 191
ccea.optimization.slave.natural_gas_main (module), 191
ccea.optimization.slave.seasonal_storage (module), 182
ccea.optimization.slave.seasonal_storage.design_operation (module), 185
ccea.optimization.slave.seasonal_storage.Import_Network_Data_functions (module), 182
ccea.optimization.slave.seasonal_storage.SolarPowerHandler_incl_Losses (module), 182
ccea.optimization.slave.seasonal_storage.storage_main (module), 186
ccea.optimization.slave.test (module), 191
ccea.optimization.slave_data (module), 193
cceaplots (module), 193
cceaplots.base (module), 227
cceaplots.cache (module), 228
cceaplots.categories (module), 229
create_trace_function() (in module cea.test.trace_inputlocator), 305
create_weather_parameters() (in module cea.interfaces.arcgis.arcgisfilehelper), 156
create_windows() (in module cea.request.ventilation_air_flows.detailed), 149
crossover_main() (in module cea.optimization.master.master_to_slave), 173
custom_database_not_found, 93
Dashboard (class in cea.plots), 193
dashboard() (in module cea.api), 318
dashboard_yaml_path() (in module cea.plots), 193
data (cea.plots.demand.comfort_chart.ComfortChartPlot attribute), 199
data (cea.plots.demand.DemandPlotBase attribute), 198
data (cea.plots.demand.heating_reset_schedule.HeatingResetSchedulePlot attribute), 204
data (cea.plots.demand.load_duration_curve.LoadDurationCurvePlot attribute), 205
data_frame_month (cea.plots.demand.energy_balance.EnergyBalancePlot attribute), 202
data_helper() (in module cea.api), 318
data_helper() (in module cea.datamanager.data_helper), 104
data_processed_costs (cea.plots.life_cycle.LifeCycleAnalysisPlotBase attribute), 208
data_processed_emissions (cea.plots.life_cycle.LifeCycleAnalysisPlotBase attribute), 208
data_processing() (cea.plots.supply_system.supply_system_map.SupplySystemPlotBase method), 222
DATA_TYPE_MAP (cea.interfaces.arcgis.arcgisfilehelper<ScalarParameter attribute>, 156
dataframe_to_dbf() (in module cea.utilities.dbf), 306
date (cea.plots.thermal_networks.ThermalNetworksPlotBase attribute), 223
DateTimeParameter (class in cea.config), 323
datetim_emination() (in module cea.plots.demand.comfort_chart), 200
db_to_dataframe() (in module cea.utilities.dbf), 306
db_to_excel_to_dbf() (in module cea.api), 318
db_to_xls() (in module cea.utilities.dbf), 307
decentralized() (in module cea.api), 318
declaration_degree() (in module cea.utilities.solar_equations), 314
decode() (cea.config.BooleanParameter method), 322
decode() (cea.config.ChoiceParameter method), 322
decode() (cea.config.DateParameter method), 323
decode() (cea.config.FileParameter method), 324
decode() (cea.config.IntegerParameter method), 324
decode() (cea.config.JsonParameter method), 324
decode() (cea.config.ListParameter method), 324
decode() (cea.config.MultiChoiceParameter method), 325
decode() (cea.config.Parameter method), 325
decode() (cea.config.PathParameter method), 326
decode() (cea.config.PlantNodeParameter method), 326
decode() (cea.config.RealParameter method), 326
decode() (cea.config.RegionParameter method), 326
decode() (cea.config.ScenarioNameParameter method), 327
decode() (cea.config.ScenarioParameter method), 327
decode() (cea.config.SubfoldersParameter method), 328
decode() (cea.config.WeatherPathParameter method), 328
decode() (cea.config.WorkflowParameter method), 328
default (cea.config.Parameter attribute), 325
default (cea.config.WeatherPathParameter attribute), 328
default_dashboard() (in module cea.plots), 193
delete_dashboard() (in module cea.plots), 193
delete_pyd() (in module cea.utilities.compile_pyd_files), 306
demand() (in module cea.api), 318
demand_calculation() (in module cea.demand.demand_main), 125
demand_files_exist() (in module cea.demand.optimization_main), 192
demand_graph_fields() (in module cea.resources.radiation_daysim.geometry_generator), 156
demand_graph_fields() (in module cea.interfaces.arcgis.cli.list_demand_graphs_fields), 160
DemandPlotBase (class in cea.plots.demand), 198
DemandSingleBuildingPlotBase (class in cea.plots.demand), 198
DemandTool (class in cea.interfaces.arcgis.CityEnergyAnalyst), 153
DemandWriter (class in cea.demand.demand_writers), 126

detailed_thermal_balance_to_tsd() (in module cea.demand.hourly_procedure_heating_cooling_system_load), 132
determine_building_supply_temperatures() (in module cea.technologies.thermal_network.substation_matrix), 255

df_to_json() (in module cea.interfaces.dashboard.inputs.routes), 162
dict_graph (cea.plots.demand.comfort_chart.ComfortChartPlot attribute), 199
dict_parameters() (in module cea.interfaces.arcgis.arcgishelper), 156
dir_last_updated() (in module cea.interfaces.dashboard.inputs.routes), 162
discharge_storage() (cea.optimization.slave.daily_storage.load_leveling.LoadLevelingDailyStorage method), 182
disconnect_buildings() (in module cea.optimization.preprocessing.decentralized_building_main), 177
disconnected_buildings_cooling_main() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 178
disconnected_buildings_heating_main() (in module cea.optimization.preprocessing.decentralized_buildings_heating), 180
disconnected_cooling_for_building() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 179
disconnected_heating_for_building() (in module cea.optimization.preprocessing.decentralized_buildings_heating), 180

dispatch_curve_district_cooling_plot (class in cea.plots.supply_system.dispatch_curve_cooling_plant), 218
dispatch_curve_district_electricity_plot (class in cea.plots.supply_system.dispatch_curve_electricity), 218
dispatch_curve_district_heating_plot (class in cea.plots.supply_system.dispatch_curve_heating_plant), 219
district_cooling_network() (in module cea.optimization.slave.cooling_main), 187
district_heating_network() (in module cea.optimization.slave.heating_main), 190
diversity_factor() (in module cea.plots.demand_peak_load), 207
diversity_factor() (in module cea.plots.demand_peak_load_supply), 208
Index
find_index_of_max() (in module cea.optimization.master.summarize_network), 176
find_isolated_endpoints() (in module cea.technologies.network_layout.connectivity_potential), 237
find_loops() (in module cea.technologies.thermal_network.thermal_network), 270
find_supplied_systems_annual() (in module cea.technologies.thermal_network.thermal_network_costs), 278
find_supplied_systems_t() (in module cea.technologies.thermal_network.thermal_network_costs), 277
find_toolbox_destination() (in module cea.interfaces.arcgis.install_toolbox), 157
find_toolbox_src() (in module cea.interfaces.arcgis.install_toolbox), 157
flush() (cea.utilities.workerstream.QueueWorkerStream method), 317
flush() (cea.utilities.workerstream.WorkerStream method), 317
flush() (cea.worker.JobServerStream method), 343
for_interface() (in module cea.scripts), 342
fuel_imports() (in module cea.optimization.slave.natural_gas_main), 191
full_report_to_xls() (in module cea.utilities.reporting), 311
furnace_op_cost() (in module cea.technologies.furnace), 289
get() (cea.config.Configuration method), 323
get() (cea.config.Parameter method), 325
generate_initial_population() (in module cea.technologies.thermal_network.thermal_network_optimization), 278
generate_main() (in module cea.optimization.master.generation), 169
generate_plants() (in module cea.technologies.thermal_network.thermal_network_optimization), 279
generate_sensor_surfaces() (in module cea.resources.radiation_daysim.geometry_generator), 230
generate_tools() (cea.interfaces.arcgis.CityEnergyAnalyst.Toolbox method), 153
GenerationPlotBase (class in cea.plots.optimization), 211
geometry_extractor_osm() (in module cea.datamanagement.surroundings_helper), 107
geometry_main() (in module cea.resources.radiation_daysim.geometry_generator), 232
geometry_reader_radiation_daysim() (cea.demand.building_properties.BuildingProperties method), 115
get() (cea.config.Configuration method), 323
get() (cea.config.Parameter method), 325
get_4D_demand_plot() (cea.inputlocator.InputLocator method), 330
get_4D_pv_plot() (cea.inputlocator.InputLocator method), 330
get_4D_pvt_plot() (cea.inputlocator.InputLocator method), 330
get_4D_radiation_plot() (cea.inputlocator.InputLocator method), 330
get_4D_sc_plot() (cea.inputlocator.InputLocator method), 330
get_all_module_rsts() (in module cea.utilities.doc_html), 307
get_archetypes_properties() (cea.inputlocator.InputLocator method), 330
get_building_age() (cea.inputlocator.InputLocator method), 330
get_building_air_conditioning() (cea.inputlocator.InputLocator method), 330
get_building_architecture() (cea.inputlocator.InputLocator method), 330
get_building_comfort() (cea.inputlocator.InputLocator method), 330
get_building_connectivity() (in module cea.plots.supply_system.supply_system_map), 222
get_building_geometry_citygml() (cea.inputlocator.InputLocator method), 331
get_building_geometry_folder() (cea.inputlocator.InputLocator method), 331
get_building_geometry_ventilation() (in
module cea.demand.ventilation_air_flows_detailed
get_calibration_problem() (cea.inputlocator.InputLocator method),
150
get_building_internal() (cea.inputlocator.InputLocator method),
get_cea_dst_folder() (in module cea.interfaces.arcgis.install_toolbox), 158
get_building_names_with_load() (in module cea.optimization.preprocessing.preprocessing_main),
180
go get_building_occupancy() (cea.inputlocator.InputLocator method),
get_cea_parameters() (in module cea.interfaces.arcgis.arcgishelper), 156
get_building_overrides() (cea.inputlocator.InputLocator method),
get_cea_parameters() (in module cea.interfaces.grasshopper.install_grasshopper), 164
get_building_properties_folder() (cea.inputlocator.InputLocator method),
get_concept_network_on_streets() (cea.inputlocator.InputLocator method), 332
get_building_supply() (cea.inputlocator.InputLocator method),
get_concept_network_plot() (cea.inputlocator.InputLocator method), 332
get_building_weekly_schedules() (cea.inputlocator.InputLocator method),
get_cooling_system_set_point() (in module cea.demand.control_heating_cooling_systems), 121
get_building_weekly_schedules_folder() (cea.inputlocator.InputLocator method), 331
get_calibration_cluster() (cea.inputlocator.InputLocator method),
get_costs_folder() (cea.inputlocator.InputLocator method), 332
get_calibration_cluster_mcda() (cea.inputlocator.InputLocator method),
get_costs_operation_file() (cea.inputlocator.InputLocator method), 332
get_calibration_cluster_mcda_folder() (cea.inputlocator.InputLocator method),
get_csv_schema() (in module cea.tests.trace_inputlocator), 305
get_calibration_cluster_opt_checkpoint() (cea.inputlocator.InputLocator method),
get_database_air_conditioning_systems() (cea.inputlocator.InputLocator method), 332
get_calibration_cluster_opt_checkpoint_folder() (cea.inputlocator.InputLocator method),
get_database_envelope_systems() (cea.inputlocator.InputLocator method), 332
get_calibration_clustering_clusters_folder() (cea.inputlocator.InputLocator method),
get_database_lca_buildings() (cea.inputlocator.InputLocator method), 332
get_calibration_clustering_folder() (cea.inputlocator.InputLocator method),
get_database_lca_mobility() (cea.inputlocator.InputLocator method), 332
get_calibration_clustering_plots_folder() (cea.inputlocator.InputLocator method),
get_database_standard_schedules() (cea.inputlocator.InputLocator method), 332
get_calibration_clusters_names() (cea.inputlocator.InputLocator method),
get_database_standard_schedules_use() (cea.inputlocator.InputLocator method), 332
get_calibration_folder() (cea.inputlocator.InputLocator method),
get_database_supply_systems() (cea.inputlocator.InputLocator method), 332
get_calibration_gaussian_emulator() (cea.inputlocator.InputLocator method),
get_databases_folder() (cea.inputlocator.InputLocator method), 332
get_calibration_posteriors() (cea.inputlocator.InputLocator method),
get_lca_embodied()
(cea.inputlocator.InputLocator method), 333
get_lca_emissions_results_folder()
(cea.inputlocator.InputLocator method), 333
get_lca_mobility()
(cea.inputlocator.InputLocator method), 333
get_lca_operation()
(cea.inputlocator.InputLocator method), 333
get_list_of_digraphs()
(in module cea.utilities.doc_graphviz), 307
get_list_of_uses_in_case_study()
(in module cea.datamanagement.data_helper), 105
get_local_etc_timezone()
(in module cea.utilities.solar_equations), 315
get_max_VCC_unit_size()
(in module cea.technologies.chiller_vapor_compression), 286
get_measurements()
(cea.inputlocator.InputLocator method), 333
get_meta()
(in module cea.tests.trace_inputlocator), 305
get_minimum_spanning_tree()
(cea.inputlocator.InputLocator method), 333
get_minmaxscalar_model()
(cea.inputlocator.InputLocator method), 333
get_minmaxscalar_folder()
(cea.inputlocator.InputLocator method), 333
get_mpc_results_building_definitions_file()
(cea.inputlocator.InputLocator method), 333
get_mpc_results_building_definitions_folder()
(cea.inputlocator.InputLocator method), 333
get_mpc_results_controls()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_district_plot_grid()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_district_plot_streets()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_electric_power()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_folder()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_max_outputs()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_max_temperature()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_max_temperature()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_min_outputs()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_min_temperature()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_outputs()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_predicted_temperature()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_set_temperature()
(cea.inputlocator.InputLocator method), 334
get_mpc_results_states()
(cea.inputlocator.InputLocator method), 334
get_multi_criteria_analysis()
(cea.inputlocator.InputLocator method), 334
get_multi_criteria_results_folder()
(cea.inputlocator.InputLocator method), 334
get_naming()
(cea.inputlocator.InputLocator method), 334
get_network_energy_pumping_requirements_file()
(cea.inputlocator.InputLocator method), 334
get_network_input_paths()
(cea.inputlocator.InputLocator method), 334
get_network_json()
(cea.plots.supply_system.supply_system_map.SupplySystemMapPlot method), 222
get_network_layout_edges_shapefile()
(cea.inputlocator.InputLocator method), 334
get_network_layout_nodes_csv_file()
(cea.inputlocator.InputLocator method), 334
get_network_layout_nodes_shapefile()
(cea.inputlocator.InputLocator method), 334
get_network_layout_pipes_csv_file()
(cea.inputlocator.InputLocator method), 334
get_network_linear_pressure_drop_edges()
(cea.inputlocator.InputLocator method), 335
get_network_linear_thermal_loss_edges_file()
(cea.inputlocator.InputLocator method), 335
get_network_pressure_at_nodes()
(cea.inputlocator.InputLocator method), 335
get_network_temperature_plant()
(cea.inputlocator.InputLocator method), 335
get_potentials_retrofit_folder()
(cea.inputlocator.InputLocator method), 339
get_potentials_solar_folder()
(cea.inputlocator.InputLocator method), 339
get_predefined_hourly_setpoints()
(cea.inputlocator.InputLocator method), 339
get_predefined_hourly_setpoints_folder()
(cea.inputlocator.InputLocator method), 339
get_preprocessing_costs()
(cea.inputlocator.InputLocator method), 339
get_project_path()
(cea.inputlocator.InputLocator method), 339
get_projected_coordinate_system()
(in module cea.utilities.standardize_coordinates), 316
get_prop_age()
(cea.demand.building_properties.BuildingProperties method), 116
get_prop_architecture()
(in module cea.datamanagement.data_helper), 105
get_prop_comfort()
(cea.demand.building_properties.BuildingProperties method), 116
get_prop_envelope()
(cea.demand.building_properties.BuildingProperties method), 116
get_prop_energy() (in module cea.demand.building_properties), 119
get_prop_geometry()
(cea.demand.building_properties.BuildingProperties method), 116
get_prop_hvac()
(cea.demand.building_properties.BuildingProperties method), 116
get_prop_internal_loads()
(cea.demand.building_properties.BuildingProperties method), 116
get_prop_occupancy()
(cea.demand.building_properties.BuildingProperties method), 116
get_prop_rc_model()
(cea.demand.building_properties.BuildingProperties method), 116
get_prop_solar()
(in module cea.demand.building_properties), 119
get_prop_supply_systems()
(cea.demand.building_properties.BuildingProperties method), 116
get_properties_natural_ventilation()
(in module cea.demand.ventilation_air_flows_detailed), 150
get_properties_supply_systems()
(in module cea.demand.building_properties), 119
get_result_building_NN()
(cea.inputlocator.InputLocator method), 339
get_retrofit_filters()
(cea.inputlocator.InputLocator method), 339
get_radiation_building()
(cea.inputlocator.InputLocator method), 339
get_radiation_building_sensors()
(cea.inputlocator.InputLocator method), 339
get_radiation_materials()
(cea.inputlocator.InputLocator method), 339
get_radiation_metadata()
(cea.inputlocator.InputLocator method), 339
get_random_presence()
(in module cea.demand.schedule_make.schedule_maker), 111
get_raw()
(cea.config.Parameter method), 325
get_raw()
(cea.config.ScenarioParameter method), 327
get_representative_week_thermal_network_layout_folder()
(cea.inputlocator.InputLocator method), 339
get_schema_scripts()
(in module cea.scripts), 343
get_schema_variables()
(in module cea.scripts), 343
get_schedule_model_file()
(cea.inputlocator.InputLocator method), 339
get_schedule_model_folder()
(cea.inputlocator.InputLocator method), 339
get_sensitivity_output()
(cea.inputlocator.InputLocator method), 340
get_sensitivity_plots_file()
(cea.inputlocator.InputLocator method), 340
get_sewage_heat_potential()
(cea.inputlocator.InputLocator method),
get_short_list_of_uses_in_case_study() (in module cea.datamanagement.schedule_helper), 106
get_shp_schema() (in module cea.tests.trace_inputlocator), 305
get_site_polygon() (cea.inputlocator.InputLocator method), 340
get_solar() (cea.demand.building_properties.BuildingProperties(cea.inputlocator.InputLocator method), 116
get_solar_radiation_folder() (cea.inputlocator.InputLocator method), 340
get_solar_time() (in module cea.utilities.solar_equations), 315
get_street_network() (cea.inputlocator.InputLocator method), 340
get_surrogate_folder() (cea.inputlocator.InputLocator method), 340
get_surroundings_geometry() (cea.inputlocator.InputLocator method), 340
get_systems_seasonality() (cea.inputlocator.InputLocator method), 340
get_t_in_pvt() (in module cea.technologies.solar.photovoltaic_thermal), 247
get_t_in_sc() (in module cea.technologies.solar.solar_collector), 249
get_t_in_pvt() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 179
get_tech_unit_size_and_number() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 179
get_temperature_setpoints_incl_seasonality() (in module cea.demand.control_heating_cooling_systems), 121
get_temporary_file() (cea.inputlocator.InputLocator method), 340
get_temporary_folder() (cea.inputlocator.InputLocator method), 340
get_test_config_path() (in module cea.tests.test_schedules), 305
get_thermal_demand_csv_file() (cea.inputlocator.InputLocator method), 340
get_thermal_network_edge_list_file() (cea.inputlocator.InputLocator method), 340
get_thermal_network_edge_node_matrix_file() (cea.inputlocator.InputLocator method), 340
get_thermal_network_folder() (cea.inputlocator.InputLocator method), 340
get_thermal_network_from_csv() (module cea.technologies.thermal_network.thermal_network.ThermalNetwork method), 259
get_thermal_network_from_shapefile() (module cea.technologies.thermal_network.thermal_network.ThermalNetwork method), 260
get_thermal_network_from_shapefile() (in module cea.technologies.thermal_network.simplified_thermal_network), 251
get_thermal_network_layout_massflow_edges_file() (cea.inputlocator.InputLocator method), 340
get_thermal_network_layout_massflow_nodes_file() (cea.inputlocator.InputLocator method), 341
get_thermal_network_node_types_csv_file() (cea.inputlocator.InputLocator method), 341
get_thermal_network_plant_heat_requirement_file() (cea.inputlocator.InputLocator method), 341
get_thermal_network_pressure_losses_edges_file() (cea.inputlocator.InputLocator method), 341
get_thermal_network_pressure_losses_edges_file() (cea.inputlocator.InputLocator method), 341
get_thermal_network_substation_loss_file() (cea.inputlocator.InputLocator method), 341
get_thermal_network_velocity_edges_file() (cea.inputlocator.InputLocator method), 341
get_thermal_network_substation_ploss_file() (cea.inputlocator.InputLocator method), 341
get_thermal_network_thermal_network_velocity_edges_file() (cea.inputlocator.InputLocator method), 341
get_thermal_network_velocity_edges_file() (cea.inputlocator.InputLocator method), 341
get_tif_schema() (in module cea.tests.trace_inputlocator), 305
get_timeseries_plots_file() (cea.inputlocator.InputLocator method), 341
get_total_demand() (cea.inputlocator.InputLocator method), 340
get_uncertainty_checkpoint() (cea.inputlocator.InputLocator method), 341
get_uncertainty_db() (cea.inputlocator.InputLocator method), 341
get_uncertainty_results_folder() (cea.inputlocator.InputLocator method), 341
get_value() (cea.interfaces.arcgis.arcgishelper.ScalarParameterInfoBuilder method), 341
get_value() (cea.interfaces.arcgis.arcgishelper.StringParameterInfoBuilder method), 156
get_water_body_potential() (cea.inputlocator.InputLocator method), 341
get_weather() (cea.inputlocator.InputLocator method), 341
get_weather_dict() (cea.inputlocator.InputLocator method), 341
get_weather_file() (cea.inputlocator.InputLocator method), 341
get_weather_folder() (cea.inputlocator.InputLocator method), 341
get_weather_names() (cea.inputlocator.InputLocator method), 341
get_weather_parameter_info() (in module cea.interfaces.arcgis.arcgishelper), 157
get_weather_path_from_parameters() (in module cea.interfaces.arcgis.arcgishelper), 157
gxls_schema() (in module cea.tests.trace_inputlocator), 305
get_yearly_vectors() (in module cea.demand.schedule_maker.schedule_maker), 111
get_zone_building_names() (cea.inputlocator.InputLocator method), 342
get_zone_geometry() (cea.inputlocator.InputLocator method), 342
getParameterInfo() (cea.interfaces.arcgis.arcgishelper.CeaTool method), 154
GHP_op_cost() (in module cea.technologies.heatpumps), 291
GHP_Op_max() (in module cea.technologies.heatpumps), 291
glossary() (in module cea.api), 319
graphviz() (in module cea.api), 319

ha (cea.utilities.solar_equations.SunProperties attribute), 312
has_3for2_cooling_system() (in module cea.demand.control_heating_cooling_systems), 122
has_ceiling_cooling_system() (in module cea.demand.control_heating_cooling_systems), 122
has_central_ac_heating_system() (in module cea.demand.control_heating_cooling_systems), 122
has_central_ac_heating_system() (in module cea.demand.re_model_SIA), 139
has_central_ac_cooling_system() (in module cea.demand.control_heating_cooling_systems), 122
has_cooling_system() (in module cea.demand.control_heating_cooling_systems), 122
has_data_load() (in module cea.demand.datacenter_loads), 125
has_floor_cooling_system() (in module cea.demand.control_heating_cooling_systems), 122
has_floor_heating_system() (in module cea.demand.control_heating_cooling_systems), 123
has_heating_system() (in module cea.demand.control_heating_cooling_systems), 123
has_heating_demand() (in module cea.demand.re_model_SIA), 139
has_hot_water_technical_system() (in module cea.demand.hotwater_loads), 129
has_local_ac_cooling_system() (in module cea.demand.control_heating_cooling_systems), 123
has_mechanical Ventilation() (in module cea.demand.control_ventilation_systems), 124
has_mechanical_ventilation_economizer() (in module cea.demand.control_ventilation_systems), 124
has_mechanical_ventilation_heat_recovery() (in module cea.demand.control_ventilation_systems), 124
has_night_flushing() (in module cea.demand.control_ventilation_systems), 124
has_radiator_heating_system() (in module cea.demand.control_heating_cooling_systems), 123
has_refrigeration_load() (in module cea.demand.refrigeration_loads), 139
has_sensible_cooling_demand() (in module cea.demand.re_model_SIA), 139
has_sensible_heating_demand() (in module cea.demand.re_model_SIA), 139
has_window_ventilation() (in module cea.demand.control_ventilation_systems), 124
heating_reset_schedule()
import_solar_thermal_data() (in module cea.optimization.slave.seasonal_storage.Import_Network_Data_functions), 182

heating_reset_schedule() (in module cea.plots.demand.heating_reset_schedule), 204

heating_source_activator() (in module cea.optimization.slave.heating_resource_activation), 191

heating_system_is_active() (in module cea.demand.control_heating_cooling_systems), 123

HeatingResetSchedulePlot (class in cea.plots.demand.heating_reset_schedule), 204

hourly_heat_loss (cea.plots.thermal_networks.ThermalNetworksPlotBase attribute), 223

hourly_loads (cea.plots.demand.DemandPlotBase attribute), 198

hourly_mass_flow_calculation() (in module cea.technologies.thermal_network.thermal_network), 270

hourly_thermal_calculation() (in module cea.technologies.thermal_network.thermal_network), 271

HourlyDemandWriter (class in cea.demand.demand_writers), 126

HourlyThermalResults (class in cea.technologies.thermal_network.thermal_network), 256

HP_air_air() (in module cea.technologies.heatsources), 293

HPLake_op() (in module cea.technologies.heatsources), 292

HPLake_op_cost() (in module cea.technologies.heatsources), 292

HPsrew_op_cost() (in module cea.technologies.heatsources), 292

html() (in module cea.api), 319

HttpWorkerStream (class in cea.utilities.workerstream), 317

hum_ratio_from_p_w_and_p() (in module cea.plots.demand.comfort_chart), 201

I_sol (cea.demand.building_properties.SolarProperties attribute), 118

id() (cea.plots.base.PlotBase class method), 227

identify_surfaces_type() (in module cea.resources.radiation_daysim.geometry_generator), 232

ignore_restrictions() (cea.config.Configuration method), 323

import_solar_thermal_data() (in module cea.optimization.slave.seasonal_storage.InstalledCapacities), 220

invalid_occupancy_name_exception() (in module cea.interfaces.dashboard.base.routes), 161

InvestmentCostsPlot (class in cea.config), 324

internal_error() (in module cea.interfaces.dashboard.base.routes), 161

individual_to_barcode() (in module cea.optimization.master_generation), 169

initialize() (cea.config.BuildingsParameter method), 322

initialize() (cea.config.ChoiceParameter method), 325

initialize() (cea.config.FileParameter method), 324

initialize() (cea.config.IntegerParameter method), 324

initialize() (cea.config.OptimizationIndividualListParameter method), 325

initialize() (cea.config.OptimizationIndividualParameter method), 325

initialize() (cea.config.Parameter method), 325

initialize() (cea.config.PathParameter method), 326

initialize() (cea.config.PlantNodeParameter method), 326

initialize() (cea.config.RealParameter method), 326

initialize() (cea.config.RegionParameter method), 327

initialize() (cea.config.ScenarioNameParameter method), 328

initialize() (cea.config.SingleBuildingParameter method), 328

initialize() (cea.config.SubfoldersParameter method), 328

initialize() (cea.config.WeatherPathParameter method), 328

initialize_inputs() (in module cea.demand.thermal_loads), 145

initialize_result_tables_for_supply_configurations() (in module cea.optimization.preprocessing.decentralized_buildings_cooling), 179

initialize_timestep_data() (in module cea.demand.thermal_loads), 145

InputLocator (class in cea.inputlocator), 329

install_arcsgis() (in module cea.api), 319

install_grasshopper() (in module cea.api), 319

InstalledCapacities (class in cea.plots.supply_system.installed_capacities), 220

IntegerParameter (class in cea.config), 324
preprocess_plot_title() (cea.plots.life_cycle.main.Plots method), 209

preprocessing_annual_costs_scenarios() (cea.plots.comparisons.ComparisonsPlotBase method), 194

preprocessing_building_costs() (cea.plots.life_cycle.main.Plots method), 209

preprocessing_building_demand() (cea.plots.demand.main.Plots method), 207

preprocessing_building_emissions() (cea.plots.life_cycle.main.Plots method), 209

preprocessing_costs_scenarios() (cea.plots.comparisons.old.main.Plots method), 196

preprocessing_demand_scenarios() (cea.plots.comparisons.old.main.Plots method), 196

preprocessing_lca_scenarios() (cea.plots.comparisons.old.main.Plots method), 196

preprocessing_occupancy_type_comparison() (cea.plots.comparisons.old.main.Plots method), 196

preprocessing_supply_scenarios() (cea.plots.comparisons.old.main.Plots method), 196

pressure_at_nodes_Pa (cea.plots.thermal_networks.ThermalNetworksPlotBase attribute), 223

pressure_at_supply_nodes_Pa (cea.technologies.thermal_network.thermal_network.HourlyThermalResults attribute), 258

pressure_loss_substations_kW (cea.technologies.thermal_network.thermal_network.HourlyThermalResults attribute), 258

pressure_loss_supply_edge_kW (cea.technologies.thermal_network.thermal_network.HourlyThermalResults attribute), 258

pressure_loss_system_kW (cea.technologies.thermal_network.thermal_network.HourlyThermalResults attribute), 258

pressure_loss_system_Pa (cea.technologies.thermal_network.thermal_network.HourlyThermalResults attribute), 258

primary_energy() (in module cea.plots.life_cycle.primary_energy), 211

primary_energy_intensity() (in module cea.plots.life_cycle.main.Plots method), 210

primary_energy_intensity() (in module cea.plots.comparisons.old.primary_energy_intensity), 197

primary_energy_intensity() (in module cea.plots.life_cycle.secondary_energy_intensity), 211

PrimaryEnergyIntensityPlot (class in cea.plots.life_cycle.primary_energy_intensity), 211

PrimaryEnergyPlot (class in cea.plots.life_cycle.primary_energy), 210

print_help() (in module cea.interfaces.cli.cea_config), 159

print_help() (in module cea.interfaces.cli.cea_doc), 159

print_help() (in module cea.interfaces.cli.cea_doc), 159

print_missing_input_files() (cea.scripts.CeaScript method), 342

print_progress() (in module cea.demand.demand_main), 126

print_progress() (in module cea.demand.schedule_maker.schedule_maker), 111

print_progress() (in module cea.resources.radiation_daysim.geometry_generator), 232

print_script_configuration() (cea.scripts.CeaScript method), 342

print_valid_script_names() (in module cea.interfaces.cli.cea_doc), 159

print_valid_script_names() (in module cea.interfaces.cli.cea_doc), 159

process_buildings_parameter() (cea.plots.supply_system.SupplySystemPlotBase method), 227

process_connected_capacities_kW() (cea.plots.supply_system.SupplySystemPlotBase method), 227

process_disconnected_capacities_kW() (cea.plots.supply_system.SupplySystemPlotBase method), 227

process_generation_total_performance_halloffame() (cea.plots.optimization.GenerationPlotBase method), 212

process_generation_total_performance_pareto() (cea.plots.optimization.GenerationPlotBase method), 212

process_individual_dispatch_curve_cooling() (cea.plots.supply_system.SupplySystemPlotBase method), 218

Index 397
Index 399
save_generation_dataframes() (in module cea.optimization.master.master_main), 171
save_generation_halloffame_individuals() set() (cea.config.Parameter method), 326
(save in module cea.optimization.master.master_main), set() (cea.config.ScenarioParameter method), 327
save_generation_individuals() (in module cea.optimization.master.master_main), 171
save_generation_pareto_individuals() (in module cea.optimization.master.master_main), 172
save_results() (in module cea.optimization.master.masterevaluation), 168
SC_ET_hourly_aggregated_kW() (cea.plots.solar_technology_potentials.SolarTechnologyPotentialsPlotBase method), 216
SC_FP_hourly_aggregated_kW (cea.plots.solar_technology_potentials.SolarTechnologyPotentialsPlotBase attribute), 217
SC_metadata_results() (cea.inputlocator.InputLocator method), 330
SC_results() (cea.inputlocator.InputLocator method), 330
SC_total_buildings() (cea.inputlocator.InputLocator method), 330
SC_totals() (cea.inputlocator.InputLocator method), 330
ScalarParameterInfoBuilder (class in cea.interfaces.arcgis.arcgishelper), 155
scaler_for_normalization() (in module cea.optimization.master.normalization), 175
ScenarioNameParameter (class in cea.config), 327
ScenarioParameter (class in cea.config), 327
ScenarioParameterInfoBuilder (class in cea.interfaces.arcgis.arcgishelper), 156
schedule_maker() (in module cea.api), 320
schedule_maker_main() (in module cea.demand.schedule_maker.schedule_maker), 111
ScheduleData (class in cea.datamanagement.schedule_helper), 106
schemas() (in module cea.scripts), 343
ScriptNotFoundException, 93
Section (class in cea.config), 327
select_from_previous_population() (in snappy_endings (module cea.optimization.thermal_network.thermal_network_optimization.network_layout.connectivity_potential), 237
sensitivity_demand_analyze() (in module cea.api), 320
sensitivity_demand_samples() (in module cea.api), 320
sensitivity_demand_simulate() (in module cea.api), 320
serialize_geometry() (in module cea.interfaces.cli.shapefile_to_excel), 161
set() (cea.config.Parameter method), 326
(set in module cea.optimization.master.master_main), set() (cea.config.ScenarioParameter method), 327
set_parameter() (in module cea.workflows.workflow), 318
set_scenario() (cea.miscs.Dashboard method), 193
set_up_environment_variables() (in module cea.workflows.workflow), 318
setUpClass() (cea.test.sensitivity_demand_simulate.TestRavelLoadAnalysis class method), 303
setUpClass() (cea.test.sensitivity_demand_simulate.TestRavelLoadAnalysis class method), 304
settlement_potentials() (module cea.api), 321
shallow_geothermal_potential() (module cea.api), 321
shapefile_to_excel() (in module cea.api), 321
shapefile_to_WSG_and_UTM() (in module cea.utilities.standardize_coordinates), 316
shutdown() (in module cea.interfaces.dashboard.base.routes), 162
simplify_liness_accuracy() (in module cea.technologies.network_layout.connectivity_potential), 237
simplify_points_accuracy() (in module cea.technologies.network_layout.connectivity_potential), 237
simulate_demand_batch() (in module cea.analysis.sensitivity.sensitivity_demand_simulate), 102
simulate_demand_sample() (in module cea.analysis.sensitivity.sensitivity_demand_simulate), 103
single_process_wrapper() (in module cea.utilities.parallel), 310
SingleBuildingParameter (class in cea.config), 328
SingleBuildingParameterInfoBuilder (class in cea.interfaces.arcgis.arcgishelper), 156
SlaveData (class in cea.optimization.slave_data), 193
snap_points() (in module cea.technologies.network_layout.connectivity_potential), 237
sobol_analyze_function() (in module cea.analysis.sensitivity.sensitivity_demand_analyze), 99
solar_collector() (in module cea.api), 321
solar_hourly_aggregated_kW() (cea.plots.solar_potential.SolarPotentialPlotBase
City Energy Analyst Documentation, Release 2.29.0

method), 216
solar_potential_folder()
(cea.inputlocator.InputLocator method), 342
SolarPotentialPlotBase (class in
cea.plots.solar_potential), 215
SolarProperties (class in
cea.demand.building_properties), 118
SolarRadiationPlot (class in
cea.plots.solar_potential.solar_radiation),
216
SolarTechnologyPotentialsPlotBase (class
in cea.plots.solar_technology_potentials), 216
solve_network_temperatures() (in module
cea.technologies.thermal_network.thermal_network), 273
split_line_by_nearest_points() (in module
cea.technologies.network_layout.connectivity_potential),
237
Storage_Charger() (in module
cea.optimization.slave.seasonal_storage.SolarPowerHandler25incl_Losses),
183
Storage_DeCharger() (in module
cea.optimization.slave.seasonal_storage.SolarPowerHandler25incl_Losses),
183
Storage_Design() (in module
cea.optimization.slave.seasonal_storage.design_operation), 185
Storage_Loss() (in module
cea.optimization.slave.seasonal_storage.SolarPowerHandler25incl_Losses),
184
Storage_Operator() (in module
cea.optimization.slave.seasonal_storage.SolarPowerHandler25incl_Losses),
184
storage_optimization() (in module
cea.optimization.slave.seasonal_storage.storage_main), 187
storage_temperature() (in module
cea.optimization.slave.daily_storage.load_leveling.DailyStorage), 182
StorageGateway() (in module
cea.optimization.slave.seasonal_storage.SolarPowerHandler25incl_Losses),
182
store_cached_value()
(cea.plots.cache.MemoryPlotCache method), 228
store_cached_value()
(cea.plots.cache.PlotCache method), 229
stream_from_queue() (in module
cea.utilities.workerstream), 317
stream_poster() (in module cea.worker), 344
streets_helper() (in module cea.api), 321
StringParameter (class in cea.api), 321
StringParameterInfoBuilder (class in
cea.interfaces.arcgis.arcgishelper), 156
SubfoldersParameter (class in cea.config), 328
SubfoldersParameterInfoBuilder (class in
cea.interfaces.arcgis.arcgishelper), 156
substation_HEX_design_main() (in module
cea.technologies.thermal_network.substation_matrix),
255
substation_HEX_sizing() (in module
cea.technologies.thermal_network.substation_matrix),
255
substation_main_cooling() (in module
cea.technologies.substation), 300
substation_main_heating() (in module
cea.technologies.substation), 300
substation_model_cooling() (in module
cea.technologies.substation), 300
substation_model_heating() (in module
cea.technologies.substation), 300
substation_return_model_main() (in module
cea.technologies.thermal_network.substation_matrix),
255
summarize_results_individual() (in module
cea.optimization.master.performance_aggregation),
summary_fuel_electricity_consumption()
(ceau.optimization.master.cost_model), 167
SunProperties (class in
cea.utilities.solar_equations), 311
supply_return_ambient_temp_plot() (in module
cea.plots.thermal_networks.5heating_reset_curve),
226
SupplyReturnAmbientCurvePlot (class in
cea.plots.thermal_networks.5heating_reset_curve),
225
SupplySystemMapPlot (class in
cea.plots.supply_system.supply_system_map), 221
supply_system_daily_storage() (class in
cea.plots.supply_system), 217
sumsurroundings_building_records()
(cea.resources.radiation_daysim.geometry_generator.BuildingData),
231
sumsurroundings_helper() (in module cea.api), 321
Sz (cea.utilities.solar_equations.SunProperties attribute), 311
T
T_return_nodes (cea.technologies.thermal_network.thermal_network.attribute), 257
T_supply_nodes (cea.technologies.thermal_network.thermal_network.attribute), 257
module cea.technologies.thermal_network.thermal_network,
275
write_results() (in module cea.demand.thermal_loads), 146
write_substation_temperatures_to_nodes_df()
(in module cea.technologies.thermal_network.thermal_network), 275
write_substation_values_to_nodes_df()
(in module cea.technologies.thermal_network.thermal_network), 276
write_to_csv() (cea.demand.demand_writers.HourlyDemandWriter
method), 126
write_to_csv() (cea.demand.demand_writers.MonthlyDemandWriter
method), 127
write_to_csv() (cea.demand.demand_writers.YearlyDemandWriter
method), 127
write_to_hdf5() (cea.demand.demand_writers.HourlyDemandWriter
method), 126
write_to_hdf5() (cea.demand.demand_writers.MonthlyDemandWriter
method), 127
write_to_hdf5() (cea.demand.demand_writers.YearlyDemandWriter
method), 127

X
xls_to_dbf() (in module cea.utilities.dbf), 307

Y
yaml_constructors
(cea.utilities.yaml_ordered_dict.OrderedDictYAMLLoader
attribute), 317
yearly_loads (cea.plots.demand.DemandPlotBase
attribute), 198
YearlyDemandWriter (class in
cea.demand.demand_writers), 127

Z
zone_helper() (in module cea.api), 322
zone_helper() (in module
cea.datamanagement.zone_helper), 109